

código.py

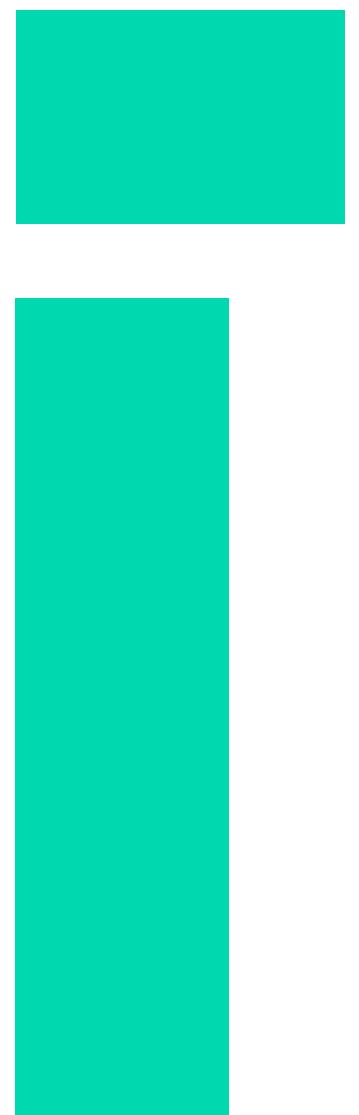
GALÁXIA 11

Factor
Investing



Introdução

Olá, seja bem-vindo à Galáxia 11 sobre Factor Investing. Nesta galáxia vamos explorar modelos de investimento vencedores no Brasil, e entender sobre os fundamentos para construir um modelo de Factor Investing. Além disso, entender a história por trás do Factor Investing, vantagens e desvantagens, sobre os diversos modelos que podem ser aplicados no Brasil. E, também, a estatísticas por trás de cada modelo e como realizar a Regressão Linear para explicar o retorno dos investimentos.



Mundo 1

1.1. – O que é Factor Investing

Factor Investing é uma metodologia, e busca explicar a seguinte questão:

“Qual a origem do retorno das ações? É possível prever o preço de uma ação?”

Portanto, é um modelo que busca alocar os ativos em uma carteira, com base em fatores que afetam o desempenho do ativo escolhido.

Assim, é possível automatizar o processo de decisão ao investir em ações, selecionando alguns fatores para alocar os ativos em uma carteira. Através de um código no Python, é possível vasculhar uma base de dados inteira, juntamente com indicadores fundamentalistas, e assim, escolher as melhores ações dentro desse universo, conforme seus critérios e indicadores.

1.2. – Como surgiu o Factor Investing

O que você precisa entender é que no mercado de ações, risco é mais importante que retorno. Afinal, o risco é a causa, e o retorno é a consequência. Logo, para descobrir a origem do retorno, é necessário entender o risco que você correu até chegar nesse retorno.

E foi assim que, Markowitz e Shape criaram uma relação na década de 50. A relação risco-retorno. Esse foi o primeiro modelo matemático criado para tentar modelar o mercado de ações, surgindo um novo campo de estudo no mercado financeiro. E assim, Markowitz e Shape provaram matematicamente que existe uma relação entre o risco e o retorno, que você conhece hoje.

Na década de 60, foi criado o primeiro modelo de Factor Investing, o Capital Asset Pricing Model (CAPM). E foi através desse modelo, que surgiu o primeiro fator, que está presente até os dias atuais em qualquer modelo, que é o famoso Beta.

O que é o Beta? Esse fator consegue mensurar a volatilidade de uma ação com seu benchmark. Portanto, se o Beta da Petrobras for 2, quando o Ibovespa subir 10%, a ação necessariamente deve subir 20%.

Todos os fatores são baseados em fatores de risco. Logo, qualquer fator exposto para escolher uma ação, você está escolhendo qual risco quer tomar.

Por conta disso, surgiu o dilema que o Beta explicaria todo o retorno de uma ação.

Contudo, Eugene Fama na década de 70, identificou uma anomalia no Low Beta. Ele conseguiu comprovar estatisticamente que o dilema criado na década de 60 era falso, e que o Beta não explica sozinho, o retorno de uma ação.

Afinal, havia empresas com beta baixo, que estavam conseguindo retorno acima do mercado. É como se o beta fosse 0.5, e enquanto o mercado subia 10%, a ação necessariamente deveria subir 5%, mas estava subindo 20%.

Com isso, o Eugene Fama tentou identificar outros fatores que contribuem para o crescimento de uma ação, além do Beta.

Após 20 anos, na década de 90, ele lançou o modelo de 3 fatores. Nesse modelo, o Beta ainda permanece, e é acrescentado outros 2 fatores: fator tamanho e fator valor. O fator tamanho diz que, quanto menor a empresa, mais arriscada ela é, portanto, maior deve ser o retorno. O fator valor diz que, quanto mais barata a empresa, maior deve ser o retorno.

Ainda na década de 90, surgiu o Momentum, um fator de análise técnica, mais especificamente de Trend Following. Esse fator foi criado por, Narasimham Jegadeesh e Sheridan Titman, um modelo de tendência. Basicamente, o fator diz que, empresas que estão subindo, tendem a subir.

Em 2015, Eugene Fama ganhou o prêmio Nobel de Economia, após lançar o modelo de 5 fatores, que é capaz de explicar 95% do retorno de uma ação. Nesse modelo, foram acrescentados o fator qualidade e o fator investimento. O fator qualidade diz que, quanto maior a rentabilidade da empresa, maior deve ser o retorno.

O fator investimento diz que, quanto menos alavancada é a empresa, maior deve ser o seu retorno.

Factor Investing é investir baseado em fatores de risco. Quanto mais rico você correr, maior será o seu retorno.

Mundo 2

2.1. – Vantagens e desvantagens do Factor Investing (data_feed.py)

As vantagens do modelo advém da sistematização do seu processo de decisão.

Os investidores criam teses de investimento, que não são totalmente objetivas. Isso faz com que a narrativa possa acabar influenciando sua decisão de compra ou venda. Com o Factor Investing, isso não ocorre, afinal, ele é 100% sistematizado, sabendo exatamente onde ocorrem os erros e acertos.

Com isso, é possível eliminar o viés comportamental na hora de comprar ou vender uma ação. Além de uma facilidade e praticidade de objetividade.

Porém, a maior vantagem de usar Factor é ter uma garantia de retorno, academicamente comprovada, conforme os artigos do Mundo 1.

“Nem tudo são flores”

Todo modelo existem prós e contras, e não seria aqui diferente.

Uma das desvantagens é ser frio na tomada de decisão. Evite contrariar o modelo, apesar de soar à primeira vista um péssimo negócio na compra de algumas ações. Às vezes, são essas ações que, no imaginário popular são péssimas, mas que geram um retorno maior quando dão a volta por cima. Portanto, se o modelo mandou comprar, compre!

Outra desvantagem é ter que acreditar na metodologia e aguentar ter um desempenho de -30% à -50%. Afinal, o modelo também erra, ele não é perfeito. Por que seria uma desvantagem? Pois você terá que acreditar fielmente ao seu modelo e estar convicto.

Em algumas decisões pode haver queda de 50%. É nesse momento que seu modelo terá uma virada de chave para uma alta rentabilidade. Caso contrário, você poderá não surfar nessa alta e terminará num prejuízo gigante.

Mundo 3

3.1. – O que não deve ser feito no Factor Investing

Factor Investing se resume a fatores de risco, e deve ser representado de uma forma prática, e não teórica.

Vamos supor que um modelo decida se expor ao fator tamanho. Logo, o modelo deve comprar empresas pequenas. E, para selecionar essas empresas, é necessário um indicador financeiro para representá-las. Por exemplo, através do Market Cap. Ao chegar nesse indicador financeiro, deve ser aplicado em uma amostra.

Afinal, o Factor Investing é uma metodologia que utiliza amostragem estatística. Portanto, se expor a fatores de risco do mercado é uma observação da média. E, na média, empresas menores se multiplicam mais do que empresas maiores.

Essa afirmação é com base em um grupo de ações, ou seja, com base em uma amostra. E, não com base em ações individuais. Portanto, nunca utilize Factor Investing para análise de ações individuais como algo determinístico.

O mercado financeiro é mais probabilístico do que determinístico. Então, é necessário uma amostra.

Por exemplo:

- “Essa ação é uma Small Cap, então ela vai se multiplicar mais que a Ambev”
- “Essa ação tem PL=2, logo, é melhor do que qualquer outra ação de outro setor com PL=19”

Repetindo, nunca utilize Factor Investing para análise de ações individuais, utilize sempre comparando uma amostragem com outra amostragem.

Mundo 4

4.1. – Modelo de 5 fatores e o Momentum

Por qual motivo o Factor Investing sempre funcionará?

Pois é uma metodologia que sempre irá assumir riscos, e para que não funcione, o mercado financeiro teria que parar de remunerar ações mais arriscadas em relação às ações menos arriscadas, onde todas as ações renderam igualmente. E isso, não faz o menor sentido econômico.

Fator Beta

Nesse fator, você está correndo mais risco por estar em renda variável. Portanto, o mercado possui um prêmio de risco em relação à renda fixa. Afinal, a volatilidade gera uma insegurança maior.

Fator Tamanho

Esse fator diz que, quanto menor uma empresa, maior a probabilidade dela dobrar de tamanho. Pois, empresas menores têm mais facilidade para alocar capital. E também, possui assimetria de informação, pois há menos analistas para cobrir esse setor.

Para representar este fator, basta calcular o Market Cap das empresas.

Esse fator não vai parar de funcionar, por conta que os fundos e grandes empresas teriam que conseguir comprar small caps. Além disso, seria necessário uma grande quantidade de analistas.

Fator Valor

No longo prazo, esse fator diz que a empresa vale o fluxo de caixa que ela gera. Se você pagar menos por esse fluxo de caixa, você terá um maior retorno.

Não é possível que esse fator pare de funcionar. Afinal, o Valuation teria que parar de funcionar, assim como o Fluxo de Caixa Descontado.

Além disso, seria necessário o fim de uma lógica econômica. Nessa situação, as empresas baratas não geraram mais retornos que as empresas caras. Somente desta forma, o Fator Valor não funcionaria.

Esse fator pode ser representado por diversos indicadores como:
Preço sobre Lucro, Ev/Evit, Preço sobre Fluxo de Caixa, TIR.

O Eugene Fama costuma utilizar o indicador Preço sobre Valor Patrimonial.

Fator Rentabilidade ou Qualidade

Esse fator diz que, a geração de valor de uma empresa está em alocar capital acima do seu custo de capital. Portanto, se a empresa não consegue projetos com retorno maior que o WACC, essa empresa destrói valor.

Logo, na média, as empresas rentáveis rendem mais.

E para representar esse fator, pode ser usado os indicadores: ROIC, ROI, ROA, EVA, ou até mesmo algumas combinações com margens (Bruta, Líquida, EBITDA).

O Eugene Fama costuma utilizar o indicador ROI para esse fator.

Fator Investimento

Esse é o último fator, é o mais fraco, estatisticamente falando. Tanto que, o Eugene Fama quando lançou o modelo somente com 4 fatores. Nesse fator, empresas com baixa alavancagem operacional têm menos chance de quebrar, pois têm maiores chances de sobreviverem em uma crise.

Para representar esse fator, é utilizado a variação dos ativos.

Momentum

Esse fator se baseia no viés comportamental de manada. É um fator mais comportamental que econômico, diferentemente do modelo de 5 fatores.

Conforme a ação sobe, teses vão se confirmando, e a ação se populariza mais ainda, convergindo todo o fluxo de dinheiro. A Magazine Luiza (MGLU3) é o maior exemplo de momentum na história do mercado financeiro brasileiro.

Como diz o ditado: "O importante é chegar cedo na festa e sair antes da confusão".

Aqui tem outro ponto também: o retorno associado a fatos novos não é instantaneamente incorporado em uma ação. É impossível prever com exatidão o futuro e uma ação pode ficar dias, meses ou anos absorvendo uma tese de investimento e alterando expectativas.

Em todos os fatores nós teremos os prêmios de risco, que é a diferença entre o primeiro e o último quartil. Então, após a coleta das amostras, é necessário dividir as empresas em quartis.

Ou seja, quanto foi remunerado a mais por estar exposto a este fator. Após isso, é comparada a rentabilidade entre o último e o primeiro quartil

Esse prêmio de risco deve ser positivo ao longo do tempo.

Mundo 5

5.1. – A estatística do Factor Investing

Antes de entrar na parte mais técnica de estatística, é preciso ter uma coisa muito clara:

Entender a intuição por trás da estatística e do modelo é mais importante do que decorar as fórmulas.

Sempre é possível consultar as fórmulas posteriormente, pois a internet pode oferecer essa informação. O que realmente importa é a compreensão intuitiva sobre o porquê e o momento adequado para aplicar cada uma delas.

5.1.1. – Distribuições de Probabilidade

Em estatística, uma distribuição de probabilidade descreve a maneira como os valores de uma variável são distribuídos, e o intervalo desses valores.

Existem 2 tipos de distribuições:

- Contínuo: Existe um número infinito de possibilidades. Por exemplo, a rentabilidade de uma ação.
- Discreto: Existe um número finito de possibilidades. Por exemplo, jogar um dado ou uma moeda.

Em 99% dos casos, vamos lidar com eventos contínuos. Afinal, o Factor Investing busca explicar o retorno de uma ação, que pode ir +- infinito.

A estatísticas descritivas de cada distribuição são chamadas de momentos. Por exemplo: média, variância, desvio padrão, covariância e correlação.

A média que você conhece e que lhe foi ensinada na escola é só uma das formas de calcular a média de uma amostra. Dependendo da distribuição, a média pode ser calculada diferente da soma dos dados dividido pelo tamanho da amostra.

Portanto, a média nada mais é do que o Valor Esperado ao escolher um valor aleatório em uma amostra.

A média em distribuições discretas e contínuas pode ser calculada através dessa fórmula.

$$\mu = E[X] = \sum_{\Omega} x P[X = x]$$

$$\int_{\Omega} x f(x) dx$$

Em estatística, o desvio padrão é conhecido pelo σ e a média pelo μ . A integral é uma forma de calcular a área de probabilidade de algum evento acontecer.

Através dessa função, e da coleta de dados das amostras, será gerado um gráfico de distribuição normal.

5.1.2. – Distribuição Normal

Essa é a distribuição mais comum que existe em estatística. Altura, peso, QI entre outras, seguem uma distribuição normal em uma população.

Nesse exemplo acima, a distribuição normal tem média igual a 0 e desvio padrão igual a 1.

A área em azul escuro representa 68% dos dados, que estão entre ± 1 desvio padrão.

Isso quer dizer que, por exemplo: ao abordar uma pessoa aleatória na rua, existe a probabilidade do peso dessa pessoa estar entre ± 1 desvio padrão da população.

Portanto, se a média do peso da população brasileira for 80kg, o desvio é de $\pm 20\text{kg}$. Logo, 68% de chance do peso dessa pessoa estar entre 60kg à 100kg.

É dessa forma que funciona em estatística, e similarmente ocorre em finanças.

Outro exemplo através da distribuição normal acima, no mundo das finanças.

A distribuição de retornos diários de uma ação se aproxima bastante de uma distribuição normal com média zero. Se o desvio padrão dos retornos diários de uma ação é 1%, 68% dos dias a ação vai cair ou subir entre 1 e -1%. Lembrando que, o desvio padrão depende da volatilidade da ação.

5.1.3. – Regressão Linear

Regressão linear é um modelo estatístico poderosíssimo para prever ou explicar uma variável Y a partir de outras (X).

No caso do Factor Investing, a variável Y é o retorno da ação ou de uma carteira e as variáveis X são os fatores!

$$\hat{Y}_t = \alpha + r_{f,t} + \beta_1 M_{ercado,t} + \beta_2 HML + \beta_3 WML + \beta_4 SMB + \epsilon_t$$

\hat{Y} é uma variável predita, isso quer dizer que, o valor dela não é de fato o que ocorreu.

Na fórmula acima:

- α (alfa): Significa que o retorno esperado da ação está sendo explicado pelos fatores (x). O ideal é um $\alpha=0$, caso contrário, significa que há um retorno adicional que não é explicado pelos fatores (x).

- RF: É o retorno livre de risco.
 - β_1 Mercado: É o beta do mercado.
 - β_2 HML: É o beta high minus low. Esse é o fator valor, ou seja, empresas baratas menos empresas caras.
 - β_3 WML: É o beta winner minus loser. Esse é o fator momentum, ou seja, empresas em tendência de alta menos empresas em tendência de baixa.
 - β_4 SMB: É o beta small minus bigger. Esse é o fator tamanho, ou seja, empresas pequenas menos empresas grandes.
 - ϵ : Esse é o erro estatístico. Afinal, todo modelo tem um erro, nenhum acerta 100%, e esse erro é normalmente distribuído para o funcionamento da regressão. Assim, é possível descobrir o motivo do erro, como dito no Mundo 2.
 - β_4 SMB = negativo, isso significa que a empresa tem uma exposição negativa ao fator tamanho, ou seja, a empresa é uma Blue Chip.
- Modelos mais complexos que esses, podem aumentar a chance de um overfitting.

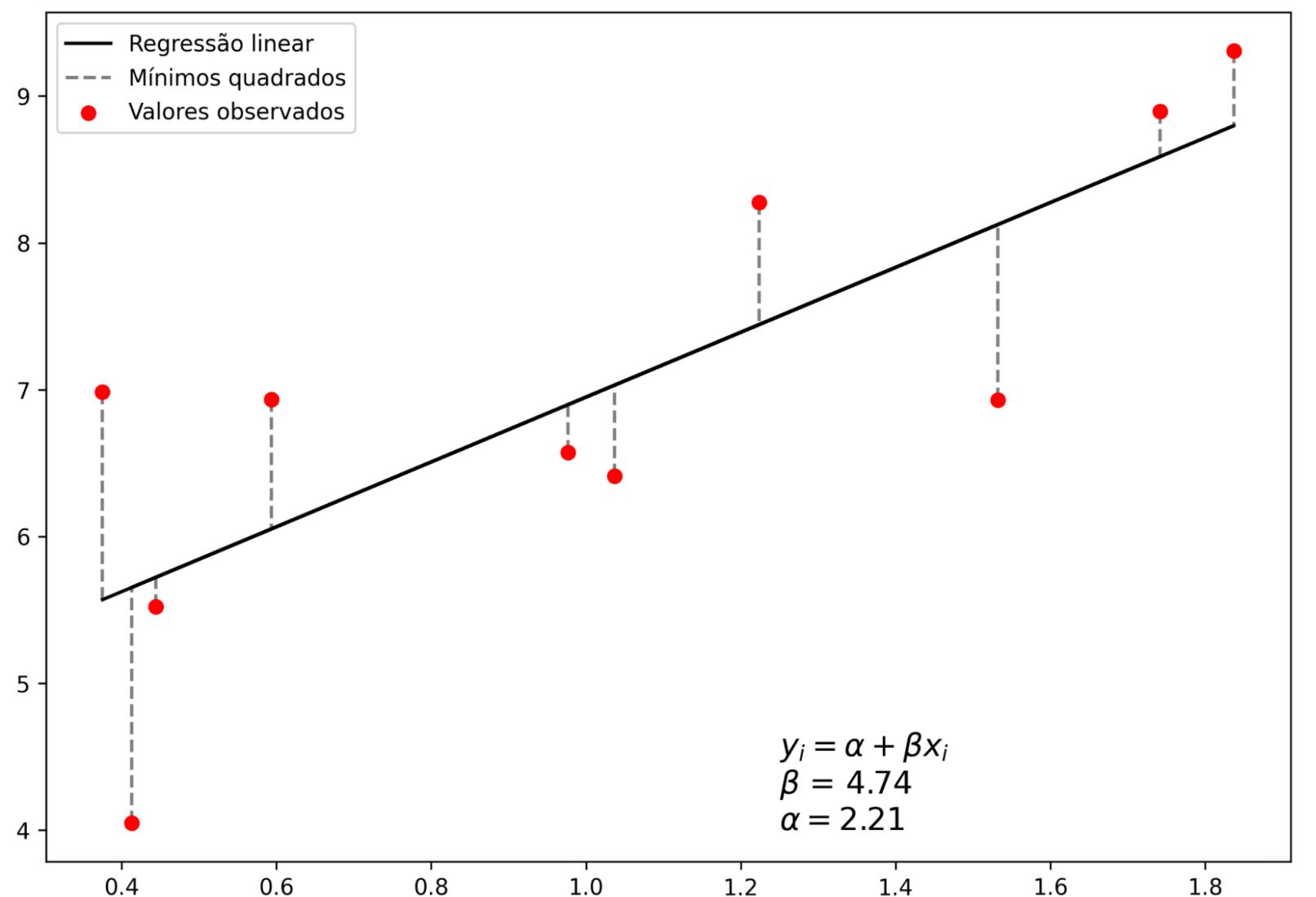
Aplicando essa função em uma empresa da bolsa, como a Ambev (ABEV3), para um exemplo ilustrativo:

- β_2 HML = 2, isso significa que a empresa tem uma exposição positiva ao fator valor, ou seja, a empresa está barata.
- β_3 WML = -1, isso significa que a empresa tem uma exposição negativa ao fator tendência, ou seja, a empresa está em tendência de

5.1.4. – Estimando o Beta

Uma das formas de estimar os betas usando a regressão linear, é utilizar o método de mínimos quadrados.

O método de mínimos quadrados é uma técnica em estatística para encontrar a melhor reta, ou curva, que se ajuste aos dados. É muito utilizado para estimar a relação entre as variáveis. O objetivo é minimizar a somatório do ϵ^2 para traçar a reta mais próxima dos dados reais históricos.



Nessa regressão existe um único beta, assim como no CAPM. O eixo X é o retorno do mercado, e o eixo Y é o retorno da ação. Através da intersecção dos dados, é gerado um ponto correspondente.

Através dos pontos correspondentes gerados, é traçada uma reta. E, nessa reta, existem um beta e um alfa, tal qual a fórmula: $Y = \alpha + \beta X$.

- Y é a variável dependente, que está sendo prevista. (retorno esperado da ação)
- α é o valor do intercepto no eixo Y, ou seja, valor de Y quando X é igual a 0.
- β é o valor da inclinação da reta (coeficiente angular), que determina a força da relação entre as variáveis.
- X é a variável independente, usada para prever Y. (prêmio de risco)

Portanto, a cada valor X, haverá um Y preditivo. Supondo que prêmio de risco seja 10 (X = 10). De acordo com a fórmula, o retorno preditivo da ação deve ser 49,61.

5.1.5. – Avaliação da Reta

Nos tópicos anteriores, foi assumido que o beta segue uma distribuição normal. Afinal, é uma estimativa, não conhecemos o beta verdadeiro. Portanto, como avaliar a reta?

- **R²** → Quanto a variação de Y é explicada pelo modelo. $0 \leq R^2 \leq 1$
 - se for 0,7, quer dizer que o beta explica 70% do retorno da ação.
- Desvio padrão dos Betas → Se for muito alto, você não sabe o valor real.
 - Há 68% de chance do verdadeiro beta estar em ± 1 Desvio Padrão do valor estimado.
 - Há 95% de chance do verdadeiro beta estar em ± 2 Desvio Padrão do valor estimado.
 - Se o beta for 0.5 e o desvio for 1, esse beta, ou seja, esse fator, é um lixo.
- P-valor → Qual a probabilidade do valor encontrado ser pura sorte?
 - O valor mais aceito é o p-valor ser menor que 5%. Isso significa que só existe 5% de chance do seu beta ser zero.
- Valor F → Avalia todos os betas serem zero, conjuntamente. Quanto maior o F, melhor.
- No caso específico do Factor Investing, nós queremos que o alfa, ou intercepto, seja ZERO.

Queremos explicar todo o retorno por fatores econômicos.

Mundo 6

6.1. – Funcionamento dos códigos

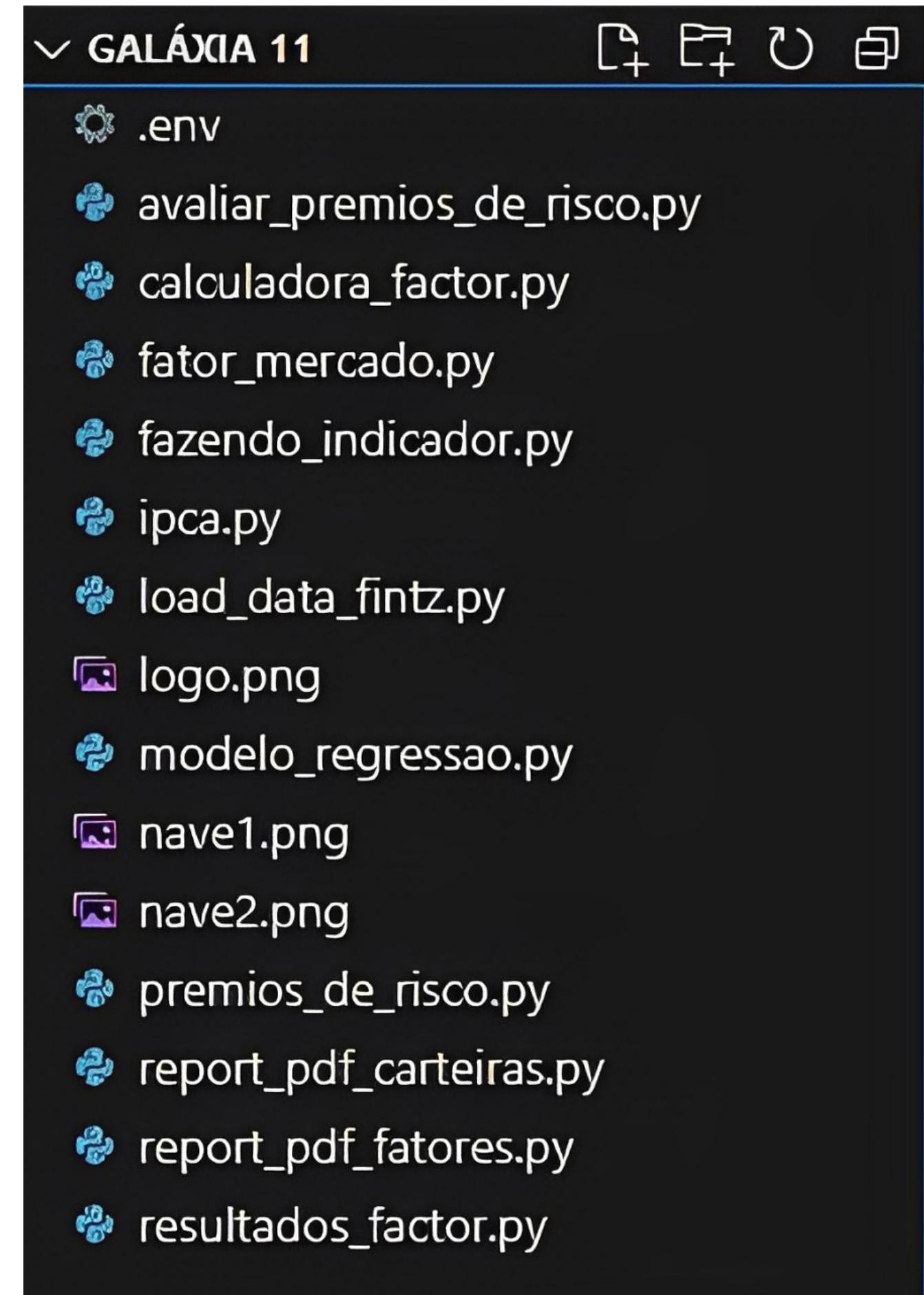
A partir deste mundo, todo o ouro sobre o conhecimento está sendo entregue.

Para isso, foram programadas 2 bibliotecas de backtest, uma para esta Galáxia de Factor Investing, e outra para a próxima Galáxia de Análise Técnica.

Através dessas bibliotecas, pode ser utilizado qualquer indicador para realizar o backtesting em questão de segundos, seja indicador fundamentalista ou técnico. Sendo possível também, testar qualquer tese de investimento

Nessa Galáxia, vamos ensinar todo o funcionamento deste sistema, a lógica por trás dos códigos e como eles conversam entre si.

Esses são os códigos para a Galáxia 11:



O primeiro passo é através do 'load_data_fintz', onde será coletado os dados mais atualizados em nossa base de dados da Fintz. Depois, serão feitos os indicadores em 'fazendo_indicadores'.

Depois dessas 2 etapas, é possível calcular os prêmios de risco no Brasil, além de avaliar o funcionamento. E finalmente, será criado o 'modelo_regressao' para definir se um modelo é bom ou ruim.

Após toda a teoria, é hora de colocar em prática, com o arquivo 'calculadora_factor', implementando a estratégia e avaliando seus resultados.

Mundo 7

7.1. – Coleta de dados na Fintz

Finalmente, será iniciada a explicação de todo o arcabouço do código.

Como foi explicado no Mundo 6, acerca do funcionamento da dinâmica do código, assim serão os próximos mundos.

O primeiro código é o “load_data_fintz”, é através desse código que será feito o acesso a API da Fintz para a coleta de todos os dados do mercado financeiro, que serão necessários para rodar o Factor Investing.

Sendo assim, o primeiro passo é inicializar o código. E através do init, será possível pegar a chave API dentro do seu arquivo ‘.env’, para liberar a autenticação com a base de dados na Fintz. Essa classe ‘dados_fintz’ possui um argumento obrigatório que é o ‘caminho_dados’, que será onde todos os parquets gerados neste código, serão salvos.

```
import requests
import pandas as pd
import os
import urllib.request

class dados_fintz:

    def __init__(self, caminho_dados):
        self.chave_api = os.getenv("API_FINTZ")
        self.headers = {'accept': 'application/json',
                       'X-API-Key': self.chave_api}
        os.chdir(caminho_dados)
```

Na classe 'dado_fintz' existem outros 4 métodos, que são para coletar dados do CDI, Ibovespa, cotações, e arquivos contábeis (demonstração ou indicadores). Os métodos são todos construídos na mesma estrutura: request na API da Fintz, coleta e tratamento dos dados, e depois é retornado um arquivo parquet

```
def cdi(self):
    response = requests.get('https://api.fintz.com.br/taxas/historico?codigo=12&dataInicio=2000-01-01&ordem=ASC',
                           headers=self.headers)

    cdi = pd.DataFrame(response.json())

    cdi = cdi.drop(['dataFim', 'nome'], axis=1)

    cdi.columns = ['data', 'retorno']

    cdi['retorno'] = cdi['retorno']/100

    cdi.to_parquet('cdi.parquet', index=False)
```

```
def ibov(self):
    response = requests.get('https://api.fintz.com.br/indices/historico?indice=IBOV&dataInicio=2000-01-01',
                           headers=self.headers)

    df = pd.DataFrame(response.json())

    df = df.sort_values('data', ascending=True)

    df.columns = ['indice', 'data', 'fechamento']

    df = df.drop('indice', axis=1)

    df.to_parquet('ibov.parquet', index=False)
```

```
def pegar_cotacoes(self):
    response = requests.get(f'https://api.fintz.com.br/bolsa/b3/avista/cotacoes/historico/arquivos?classe=ACOES&preencher=true',
                           headers=self.headers)
    link_download = (response.json()['link'])

    urllib.request.urlretrieve(link_download, f"cotacoes.parquet")

    df = pd.read_parquet('cotacoes.parquet')

    colunas_pra_ajustar = ['preco_abertura', 'preco_maximo', 'preco_medio', 'preco_minimo']

    for coluna in colunas_pra_ajustar:

        df[f'{coluna}_ajustado'] = df[coluna] * df['fator_ajuste']

    df['preco_fechamento_ajustado'] = df.groupby('ticker')['preco_fechamento_ajustado'].transform('ffill')

    df = df.sort_values('data', ascending=True)

    df.to_parquet('cotacoes.parquet', index=False)

def pegando_arquivo_contabil(self, demonstracao=False, indicadores=False, nome_dado=''):
    if demonstracao:
        try:
            response = requests.get(f'https://api.fintz.com.br/bolsa/b3/tm/demonstracoes/arquivos?item={nome_dado}',
                                   headers=self.headers)
        except:
            print("Demonstração não encontrada!")
            exit()
        link_download = (response.json()['link'])
        urllib.request.urlretrieve(link_download, f"{nome_dado}.parquet")
    elif indicadores:
        try:
            response = requests.get(f'https://api.fintz.com.br/bolsa/b3/tm/indicadores/arquivos?indicador={nome_dado}',
                                   headers=self.headers)
        except:
            print("Indicador não encontrado!")
            exit()
        link_download = (response.json()['link'])
        urllib.request.urlretrieve(link_download, f"{nome_dado}.parquet")
    else:
        print("Escolha uma demonstração ou indicador.")
```

O fim do código é basicamente selecionar um dos métodos escolhidos, e exportá-lo para uma pasta de arquivos em seu computador.

```
if __name__ == "__main__":
    lendo_dados = dados_fintz(caminho_dados=r'C:\Users\lsiqu\dev\base_dados_br')
    lendo_dados.cdi()
    lendo_dados.ibov()
    lendo_dados.pegar_cotacoes()
```

Para exportar o método ‘pegando_arquivo_contabil’, é um pouco diferente. Antes de exportar, é necessário escolher as demonstrações e os indicadores através das listas.

```
if __name__ == "__main__":
    lendo_dados = dados_fintz(caminho_dados=r'C:\Users\lsiqu\dev\base_dados_br')

    lista_demonstracoes = ['Ebit12m', 'DividaBruta', 'DividaLiquida', 'Ebit12m', 'LucroLiquido12m', 'PatrimonioLiquido', 'ReceitaLiquida12m']
    lista_indicadores = ['EBIT_EV', 'L_P', 'ROE', 'ROIC', 'ValorDeMercado']

    for demonstracao in lista_demonstracoes:
        print(demonstracao)
        lendo_dados.pegando_arquivo_contabil(demonstracao=True, nome_dado = demonstracao)

    for indicador in lista_indicadores:
        print(indicador)
        lendo_dados.pegando_arquivo_contabil(indicadores=True, nome_dado = indicador)
```

Mundo 8

8.1. – Criação de Indicadores

A próxima etapa desta Galáxia, é no código “fazendo_indicadores”, que é um dos arquivos bases para que o modelo funcione. É nessa etapa que os indicadores são configurados.

Esse modelo de backtest segue algumas premissas, e uma delas é: a coluna com o indicador tem que se chamar “valor”. Caso contrário, o código ficará bugado.

Toda classe deve iniciar com o ‘init’, sempre.

```
class MakeIndicator():
    def __init__(self, caminho_dados):
        os.chdir(caminho_dados)
```

8.1.1. – Indicador de momento

Neste indicador, e em todos os outros, iniciam da mesma forma: coleta de dados atualizados na base da Fintz.

Para o indicador de momento, há a coleta das cotações e depois é criado um retorno, conforme os meses que você irá escolher. Será utilizado como base a coluna do preço do fechamento ajustado.

Toda vez que o valor for zero ou infinito, será substituído por “N/A”, e depois é feito um ‘dropna()’ para a exclusão desses dados.

Essa estrutura segue para todos os indicadores.

```
def fazer_indicador_momento(self, meses):
    cotacoes = pd.read_parquet('cotacoes.parquet')
    cotacoes['data'] = pd.to_datetime(cotacoes['data']).dt.date
    cotacoes = cotacoes[['data', 'ticker', 'preco_fechamento_ajustado']]
    cotacoes['valor'] = cotacoes.groupby('ticker')['preco_fechamento_ajustado'].pct_change(periods = (meses * 21))
    cotacoes.loc[cotacoes['valor'] == 0, 'valor'] = pd.NA
    cotacoes.loc[cotacoes['valor'] == np.inf, 'valor'] = pd.NA
    cotacoes = cotacoes.dropna()
    valor = cotacoes[['data', 'ticker', 'valor']]
    valor.to_parquet(f'momento_{meses}_meses.parquet', index = False)
```

8.1.2. – Volume mediano

Esse indicador de volume mediano é importante para criar um filtro de liquidez, afinal, o modelo pode ser enganado.

Imagine uma situação de uma empresa de baixa liquidez, em que algum fator externo influencia a cotação para que ela decole ou desabe, sendo um dia totalmente fora da curva. Supondo que a cotação subiu bastante, isso influenciará no seu modelo, exercendo a ordem de compra.

Depois de coletar e tratar os dados, este indicador calcula a mediana do volume para cada ação ao longo de uma janela móvel de 21 dias. Pois, assim, será capturado somente o volume mais repetido.

```
def volume_mediano(self):
    cotacoes = pd.read_parquet('cotacoes.parquet')
    cotacoes['data'] = pd.to_datetime(cotacoes['data']).dt.date

    cotacoes = cotacoes[['data', 'ticker', 'volume_negociado']]
    cotacoes['volume_negociado'] = cotacoes.groupby('ticker')['volume_negociado'].fillna(0)
    cotacoes['valor'] = cotacoes.groupby('ticker')['volume_negociado'].rolling(21).median().reset_index(0,drop=True)
    cotacoes = cotacoes.dropna()
    valor = cotacoes[['data', 'ticker', 'valor']]

    valor.to_parquet(f'volume_mediano.parquet', index = False)
```

8.1.3. – Ebit/Dívida Líquida

Esse é um indicador econômico informando ao investidor que, quanto maior o EBIT/DÍV. LÍQ, melhor é a saúde financeira da empresa. Portanto, aqui é necessário uma adaptação do código para essa interpretação.

Nesse indicador, quando a dívida líquida é zero ou negativa, a empresa não possui dívida. Nesta situação, o código irá transformar a empresa com o melhor EBIT/DÍV. LIQ.

Do outro lado, quando o EBIT é igual a zero ou negativo, a empresa não possui resultado operacional. Nesse caso, o código irá transformar a empresa com o melhor EBIT/DÍV. LIQ.

Nessa situação, foi inserido o valor "999" e "-999", para que a empresa seja posicionada no topo ou fundo do ranking, respectivamente. Contudo, esse valor pode ser modificado, desde que mantenha a mesma lógica e interpretação do indicador.

```
df_indicadores.loc[df_indicadores['divida'] <= 0, 'ebit_DL'] = 999
df_indicadores.loc[df_indicadores['ebit'] <= 0, 'ebit_DL'] = -999
```

Caso não ocorra essas particularidades, o indicador será calculado, dividindo o EBIT pela DÍV. LIQ.

```
def ebit_divida_liquida(self):
    df_ebit = pd.read_parquet('Ebit12m.parquet')
    df_ebit['id_dado'] = df_ebit['ticker'].astype(str) + "_" + df_ebit['data'].astype(str)
    df_ebit['valor'] = df_ebit['valor'].astype(float)
    df_ebit = df_ebit[['ticker', 'data', 'id_dado', 'valor']]
    df_ebit.columns = ['ticker', 'data', 'id_dado', 'ebit']

    df_divida_liquida = pd.read_parquet('DividaLiquida.parquet')
    df_divida_liquida['id_dado'] = df_divida_liquida['ticker'].astype(str) + "_" + df_divida_liquida['data'].astype(str)
    df_divida_liquida['valor'] = df_divida_liquida['valor'].astype(float)
    df_divida_liquida = df_divida_liquida[['id_dado', 'valor']]
    df_divida_liquida.columns = ['id_dado', 'divida']

    df_indicadores = pd.merge(df_ebit, df_divida_liquida, how = 'inner', on = 'id_dado')
    df_indicadores['ebit_DL'] = pd.NA
    df_indicadores.loc[df_indicadores['divida'] <= 0, 'ebit_DL'] = 999
    df_indicadores.loc[df_indicadores['ebit'] <= 0, 'ebit_DL'] = -999

    df_indicadores.loc[df_indicadores['ebit_DL'].isna(), 'ebit_DL'] = (df_indicadores[df_indicadores['ebit_DL'].isna()]['ebit']/
    df_indicadores[df_indicadores['ebit_DL'].isna()]['divida'])
    df_indicadores = df_indicadores[['data', 'ticker', 'ebit_DL']]
    df_indicadores.columns = ['data', 'ticker', 'valor']

    df_indicadores.to_parquet(f'ebit_dl.parquet', index = False)
```

8.1.4. – Patrimônio Líquido/Dívida Bruta

Esse indicador funciona na mesma dinâmica do indicador anterior, porém, com uma interpretação diferente.

Quanto menor for o número desse indicador, pior é a empresa no quesito endividamento e alavancagem. O Patrimônio Líquido pela Dívida Bruta mostra a relação entre o passivo e o ativo de uma empresa.

Se o Patrimônio Líquido for igual a zero, é a pior situação possível para o indicador. Afinal, a empresa possui mais passivos do que ativos.

```
df_indicadores.loc[df_indicadores['patrimonio_liquido'] <= 0, 'PL_DB'] = 0
```

```
def pl_divida_bruta(self):
    df_pl = pd.read_parquet('PatrimonioLiquido.parquet')
    df_pl = df_pl.dropna()
    df_pl['id_dado'] = df_pl['ticker'].astype(str) + "_" + df_pl['data'].astype(str)
    df_pl['valor'] = df_pl['valor'].astype(float)
    df_pl = df_pl[['data', 'ticker', 'valor', 'id_dado']]
    df_pl.columns = ['data', 'ticker', 'patrimonio_liquido', 'id_dado']

    df_divida_bruta = pd.read_parquet('DividaBruta.parquet')
    df_divida_bruta[df_divida_bruta['valor'] == '0.0'] = pd.NA
    df_divida_bruta = df_divida_bruta.dropna()
    df_divida_bruta = df_divida_bruta.assign(id_dado = df_divida_bruta['ticker'].astype(str) + "_" + df_divida_bruta['data'].astype(str))
    df_divida_bruta['valor'] = df_divida_bruta['valor'].astype(float)
    df_divida_bruta = df_divida_bruta[['id_dado', 'valor']]
    df_divida_bruta.columns = ['id_dado', 'divida']

    df_indicadores = pd.merge(df_pl, df_divida_bruta, how = 'inner', on = 'id_dado')
    df_indicadores['PL_DB'] = pd.NA
    df_indicadores.loc[df_indicadores['patrimonio_liquido'] <= 0, 'PL_DB'] = 0
    df_indicadores.loc[df_indicadores['PL_DB'].isna(), 'PL_DB'] = (df_indicadores[df_indicadores['PL_DB'].isna()]['patrimonio_liquido']/
    df_indicadores[df_indicadores['PL_DB'].isna()]['divida'])
    df_indicadores = df_indicadores[['data', 'ticker', 'PL_DB']]
    df_indicadores.columns = ['data', 'ticker', 'valor']

    df_indicadores.to_parquet('pl_db.parquet', index = False)
```

8.1.5. Volatilidade

Nessa parte do código sobre o indicador de volatilidade, observe que há um período mínimo para que a volatilidade seja calculada. Portanto, a empresa deve ter dados pelo menos em 80% do período. Caso contrário, a volatilidade não é calculada.

```
cotacoes['valor'] = cotacoes.groupby('ticker')['retorno'].rolling(window=int(252 * anos),
min_periods=int(252 * anos * 0.8)).std().reset_index(0,drop=True)
```

```
def volatilidade(self, anos):
    cotacoes = pd.read_parquet('cotacoes.parquet')
    cotacoes['data'] = pd.to_datetime(cotacoes['data']).dt.date
    cotacoes = cotacoes[['data', 'ticker', 'preco_fechamento_ajustado']]
    cotacoes['retorno'] = cotacoes.groupby('ticker')['preco_fechamento_ajustado'].pct_change()
    cotacoes.loc[cotacoes['retorno'] == 0, 'retorno'] = pd.NA
    cotacoes.loc[cotacoes['retorno'] == np.inf, 'retorno'] = pd.NA
    cotacoes['valor'] = cotacoes.groupby('ticker')['retorno'].rolling(window=int(252 * anos),
min_periods=int(252 * anos * 0.8)).std().reset_index(0,drop=True)
    cotacoes = cotacoes.dropna()
    cotacoes['valor'] = cotacoes['valor'] * np.sqrt(252)
    valor = cotacoes[['data', 'ticker', 'valor']]
    valor.to_parquet(f'vol_{int(252 * anos)}.parquet', index = False)
```

8.1.6. – Beta

Nessa função será calculado o Beta, através de uma Regressão Linear contra o Ibovespa, assim como explicado nos mundos anteriores.

Assim como no indicador da volatilidade, há um período mínimo para ser calculado. Logo, a empresa deve ter dado pelo menos em 80% do período.

```
def beta(self, anos):

    cotacoes = pd.read_parquet('cotacoes.parquet')
    cotacoes_ibov = pd.read_parquet('ibov.parquet')

    cotacoes_ibov.loc['5846'] = ['2023-08-10', 118349.60]

    cotacoes_ibov['retorno_ibov'] = cotacoes_ibov['fechamento'].pct_change()
    cotacoes_ibov = cotacoes_ibov[['data', 'retorno_ibov']]
    cotacoes_ibov['data'] = pd.to_datetime(cotacoes_ibov['data']).dt.date

    cotacoes['data'] = pd.to_datetime(cotacoes['data']).dt.date
    cotacoes = cotacoes[['data', 'ticker', 'preco_fechamento_ajustado']]
    cotacoes['retorno'] = cotacoes.groupby('ticker')[['preco_fechamento_ajustado']].pct_change()
    cotacoes.loc[cotacoes['retorno'] == 0, 'retorno'] = pd.NA
    cotacoes.loc[cotacoes['retorno'] == np.inf, 'retorno'] = pd.NA

    dados_totais = pd.merge(cotacoes, cotacoes_ibov, on='data', how='inner')

    empresas = dados_totais['ticker'].unique()
    dados_totais = dados_totais.set_index('ticker')
    lista_df_betas = []

    for empresa in empresas:

        dado_empresa = dados_totais.loc[empresa]

        if dado_empresa.dropna().empty == False:

            if len(dado_empresa) > int(252 * anos):

                datas = dado_empresa.data.values
                exog = sm.add_constant(dado_empresa.retorno_ibov)
                model = RollingOLS(endog=dado_empresa.retorno.values, exog=exog,
                                    window=int(252 * anos), min_nobs = int(252 * anos * 0.8))
                betas = model.fit()
                betas = betas.params
                dado_empresa = betas.reset_index()
                dado_empresa['data'] = datas
                dado_empresa.columns = ['ticker', 'const', 'valor', 'data']
                dado_empresa = dado_empresa[['data', 'ticker', 'valor']]
                dado_empresa = dado_empresa.dropna()
                lista_df_betas.append(dado_empresa)

    betas = pd.concat(lista_df_betas)
    betas.to_parquet(f'beta_{int(252 * anos)}.parquet', index = False)
```

8.1.7. – Média móvel proporção

Esse indicador é uma junção da análise técnica com a análise fundamentalista, usado como base o modelo de média móvel de 7 com 40 períodos.

Sua explicação é simples: divide-se média curta pela média longa.

Se o valor for acima de 1, significa que a ação está em tendência de alta. Além disso, quanto maior esse indicador, melhor.

Do contrário, suponha que a média curta seja R\$15 e a média longa R\$20. Portanto, esse valor está abaixo de 1, significa que a ação está em tendência de baixa.

8.1.8. – Criação dos indicadores

E por fim, os indicadores serão retornados, através da mesma lógica de funcionamento como foi no ‘load_data_fintz’.

```
if __name__ == "__main__":  
  
    indicador = MakeIndicator(caminho_dados=r'C:\Users\lsiqu\dev\base_dados_br')  
  
    indicador.fazer_indicador_momento(meses=12)  
    indicador.fazer_indicador_momento(meses=1)  
    indicador.fazer_indicador_momento(meses=6)  
    indicador.volume_mediano()  
    indicador.media_movel_proporcao(7, 40)  
    indicador.beta(3)  
    indicador.volatilidade(1)  
    indicador.pl_divida_bruta()
```

8.2 – Comentários

Sempre evite ao máximo utilizar loops em seus códigos. No máximo, utilize loops curtos. Como é possível verificar, foi utilizado groupby em diversos casos para evitar a utilização de loops.

O motivo é que os loops deixam o código extremamente mais lento, afinal, o python foi criado para ser utilizado através de vetores.

E lembre-se, o ouro está sendo entregue. Logo, você está totalmente livre para realizar modificações em qualquer linha do código.

Mundo 9

9.1. – Como calcular o prêmio de risco

Antes de iniciarmos a explicação dessa etapa do código de “premios_de_risco”, é necessário criar uma pasta dentro da pasta que estão armazenados os parquets. Dessa forma, é feita uma separação entre os parquets dos indicadores com os parquets dos prêmios de risco.

Esse arquivo está totalmente estruturado em uma forma sequencial. Ou seja, para um método ocorrer, tem que ocorrer o método anterior.

Por exemplo: para a coleta das possíveis datas e filtragem do volume, é necessário coletar as cotações. Por isso, o cálculo do prêmio de risco é a última etapa.

9.2. – Init

Como todo código, sempre faça o init. Dessa vez, a inicialização capta os indicadores criados em ‘fazendo_indicadores’ e fará a ordenação.

Assim como em ‘load_data_fintz’ e ‘fazendo_indicadores’, é necessário passar o caminho dos dados para salvar os arquivos.

```
import pandas as pd
import numpy as np
from dateutil.relativedelta import relativedelta
import os

class premio_risco:

    def __init__(self, indicadores_dict, nome_premio, liquidez=0, caminho_dados = None, caminho_salvar_arquivo = '.'):
        self.indicadores = list(indicadores_dict.keys())
        self.ordem_indicadores = list(indicadores_dict.values())
        self.liquidez = liquidez
        self.nome_premio = nome_premio
        self.caminho_salvar_arquivo = caminho_salvar_arquivo

        if caminho_dados != None:
            os.chdir(caminho_dados)
```

9.3. – Dados da cotação

A primeira etapa dessa função ‘pegando_dados_cotacoes’ é ler o parquet das cotações, gerado em ‘load_data_fintz’. Posteriormente, será criado um ID para todos os dados de cotações. Esse ID será composto com o ticker+data, sendo assim, uma identidade única. Afinal, uma cotação só pode ser de apenas um ticker numa determinada data.

O objetivo da criação do ‘id_dado’ é para que seja possível realizar um merge durante todo o código.

Por fim, é aconselhável sempre transformar uma coluna de data em datetime, para confirmar se realmente é uma data. O motivo é que a biblioteca do pandas nem sempre lê como uma data, e pode ocorrer de bugar o código por completo por transformar automaticamente em uma string.

```
def pegando_dados_cotacoes(self):
    self.cotacoes = pd.read_parquet("cotacoes.parquet")
    self.cotacoes['id_dado'] = self.cotacoes['ticker'].astype(str) + " " + self.cotacoes['data'].astype(str)
    self.cotacoes['data'] = pd.to_datetime(self.cotacoes['data']).dt.date
```

9.4. – Possíveis datas

Na função ‘pegando_datas_possiveis’, a ideia é pegar o último dia útil de cada mês no prêmio de risco, para captar o retorno mensal, e assim ranquear as ações conforme o indicador selecionado.

Isso é construído de forma bem simples. Foi escolhida a ação da Petrobras por ser uma empresa que sempre existiu na bolsa, e provavelmente sempre será negociada. Então, o código acaba sendo dependente da Petrobras, e caso a empresa venha decretar falência, o código quebra. E convenhamos, isso é quase impossível.

Portanto, nesse trecho do código, será coletado as ações da Petrobras, feito um groupby por ano e mês. Isso faz com que colete a cotação do último dia útil de cada mês. Portanto, será coletado o último dia útil de todos os meses da bolsa, por conta que a Petrobras sempre existiu e sempre existirá.

```
def pegando_datas_possiveis(self):  
  
    cot_petr = self.cotacoes[self.cotacoes['ticker'] == 'PETR4']  
  
    cot_petr = cot_petr.sort_values('data', ascending = True)  
    cot_petr = cot_petr.assign(year = pd.DatetimeIndex(cot_petr['data']).year)  
    cot_petr = cot_petr.assign(month = pd.DatetimeIndex(cot_petr['data']).month)  
    datas_final_mes = cot_petr.groupby(['year', 'month'])['data'].last()  
    datas_final_mes = datas_final_mes.reset_index()  
  
    self.datas_final_mes = datas_final_mes
```

9.5. – Filtragem de volume

Como dito anteriormente, o código é construído sequencialmente. Portanto, após a coleta das possíveis datas, chegou a hora de filtrar o volume.

Assim como todo método, aqui é feito uma leitura do parquet gerado pelo indicador ‘volume_mediano’.

E assim como em ‘pegando_dados_cotacoes’, será criado um ‘id_dado’ único. Logo, será feito um merge e criada uma tabela com esses dados.

Por fim, será criado um filtro para a liquidez que será escolhida. Dessa forma, é possível escolher somente empresas com alta liquidez.

```
def filtrando_volume(self):  
  
    dados_volume = pd.read_parquet("volume_mediano.parquet")  
    dados_volume['id_dado'] = dados_volume['ticker'].astype(str) + "_" + dados_volume['data'].astype(str)  
    dados_volume = dados_volume[['id_dado', 'valor']]  
    dados_volume.columns = ['id_dado', 'volumeMediano']  
    self.cotacoes = pd.merge(self.cotacoes, dados_volume, how = 'inner', on = 'id_dado')  
    self.cotacoes = self.cotacoes[self.cotacoes['volumeMediano'] > self.liquidez]
```

9.6. – Indicadores

Esse método ‘pegando_indicadores’ é bem fácil. Para cada indicador escolhido, será feito uma leitura do arquivo parquet gerado, e será acrescentado na lista dos indicadores escolhidos (‘self.df_indicadores’).

```
def pegando_indicadores(self):  
  
    self.df_indicadores = []  
    for indicador in self.indicadores:  
        try:  
            df = pd.read_parquet(f'{indicador}.parquet')  
            df['data'] = pd.to_datetime(df['data']).dt.date  
        except:  
            df = None  
        self.df_indicadores.append(df)
```

9.7. – Descobrindo o mês inicial

O método 'descobrindo_mes_inicial' descobre automaticamente o mês em que o backtest tem que iniciar. Afinal, cada indicador começa em um mês diferente, por razões de cálculo. Em outros casos, é simplesmente por conta que as empresas demoram para liberar os dados depois que o indicador surge.

Então, após a coleta dessa data mínima, será acrescentado 2 meses. O motivo é que, alguns indicadores fundamentalistas não possuem informações completas inicialmente para serem calculados.

Por exemplo, o ROIC possui na base de dados desde outubro 2011, mas como estava em período de divulgação de balanços, é necessário aguardar o fim dessa divulgação. Caso contrário, só vai haver os dados do ROIC para as empresas que divulgaram o balanço até aquele momento.

Esse exemplo anterior ilustra bem a importância de acrescentar 2 meses.

E por fim, será filtrado as cotações somente acima da data mínima, otimizando o código.

```
def descobrindo_mes_inicial(self):
    datas = []
    for df in self.df_indicadores:
        indicador_sem_na = df.dropna()
        petr_indicador = indicador_sem_na.query('ticker == "PETR4"') #eu usei a petro como parametro pra saber a primeira data de qlqr indicador.
        data_minima = min(petr_indicador['data'])
        datas.append(data_minima)
    data_minima_geral = max(datas) #maximo porque queremos a MAIOR data mínima entre os indicadores.
    self.data_minima_geral = data_minima_geral + relativedelta(months=+2)
    #vou colocar 2 meses depois da primeira empresa ter o indicador. isso pq os indicadores fundamentalistas no
    #1 mes da base podem ser muito escassos devido a maioria das empresas não terem soltado resultado ainda.
    self.lista_datas_final_mes = (self.datas_final_mes.query('data >= @self.data_minima_geral'))['data'].to_list()
    self.cotacoes_filtrado = self.cotacoes.query('data >= @self.data_minima_geral')
```

Mundo 10

10.1. – Preparação Inicial

Vamos terminar de explicar o arquivo “premio_de_risco”, e este mundo será específico somente para a explicação do método ‘calculando_premios’.

O objetivo do método ‘calculando_premios’ é criar um DataFrame dos prêmios de risco, dividido entre quartis e universo. Essas colunas criadas no DataFrame representam a rentabilidade de cada quartil, enquanto em universo é a rentabilidade de todas as empresas.

A importância do universo é para que seja possível comparar com os quartis, afinal, o Ibovespa na maioria das vezes será superado e não faz sentido compará-lo.

Essa preparação inicial é essencial, pois cria um lugar bem organizado para guardar os resultados dos prêmios que serão calculados mais adiante. É como montar uma estrutura sólida antes de começar a adicionar os detalhes importantes.

Isso permite que, à medida que o código avança e calcula os prêmios, todas as informações necessárias tenham um lugar específico para serem armazenadas, facilitando muito a organização e entendimento dos resultados.

O código começa preparando um espaço para armazenar informações importantes. Ele cria o DataFrame ‘df_premios’ para guardar dados sobre prêmios que serão calculados mais tarde.

```
def calculando_premios(self):  
    colunas = ['primeiro_quartil', 'segundo_quartil', 'terceiro_quartil',  
    'quarto_quartil', 'universo']  
    df_premios = pd.DataFrame(columns=colunas,  
    index=self.lista_datas_final_mes)  
    lista_dfs = [None] * 4 # inicializa lista para guardar dataframes de quartis
```

10.2. – Calculando a rentabilidade da carteira – Parte 1

O próximo trecho do código tem a finalidade de calcular o retorno de uma carteira de investimentos em datas anteriores. Ele começa verificando se não estamos na primeira iteração do loop sobre as datas. Caso não seja a primeira iteração, ou seja, já temos informações de datas anteriores, ele executa o cálculo para determinar o retorno da carteira.

O código então calcula o retorno médio de cada quartil e armazena esses valores na tabela `df_premios` para a data correspondente. Por fim, é criado o retorno do universo, que consiste na média do retorno de todas as empresas.

```
if i != 0:  
    df_vendas = df_infoPontuais[['ticker', 'preco_fechamento_ajustado']]  
    df_vendas.columns = ['ticker', 'preco_fechamento_ajustado_posterior']  
    lista_retornos = []  
  
    for zeta, df in enumerate(lista_dfs):  
        df_quartil = pd.merge(df, df_vendas, how='inner', on='ticker')  
        df_quartil['retorno'] = df_quartil['preco_fechamento_ajustado_posterior'] / df_quartil['preco_fechamento_ajustado'] - 1  
        retorno_quartil = df_quartil['retorno'].mean()  
        lista_retornos.append(retorno_quartil)  
        df_premios.loc[data, colunas[zeta]] = retorno_quartil  
  
    df_premios.loc[data, 'universo'] = np.mean(np.array(lista_retornos))
```

10.3. – Calculando a rentabilidade da carteira - Parte 2

Basicamente, esse trecho de código na imagem abaixo, está incorporando os valores dos indicadores financeiros ao DataFrame 'df_info_pontuais'. Ele itera sobre todos os indicadores disponíveis, seleciona os valores desses indicadores para a data específica e os adiciona ao DataFrame principal 'df_info_pontuais' usando a coluna ticker como chave para o merge.

Dessa forma, o DataFrame 'df_info_pontuais' será expandido para conter informações sobre os indicadores financeiros relevantes para as ações presentes nesse DataFrame, facilitando a geração do ranking final com base nesses indicadores.

```
#pegando as novas carteiras
df_info_pontuais['ranking_final'] = 0

for alfa, indicador in enumerate(self.df_Indicadores):
    indicador_na_data = indicador.loc[indicador['data'] == data, ['ticker', 'valor']].dropna()
    indicador_na_data.columns = ['ticker', f'indicador_{self.Indicadores[alfa]}']
    df_info_pontuais = pd.merge(df_info_pontuais, indicador_na_data, how='inner', on='ticker')
```

10.4. – Filtrando empresas líquidas

Ainda no método “calculando_premios”, essa parte do código visa sómente excluir empresas repetidas e manter o ticker com maior volume. Afinal, estamos interessados na empresa, e não nos códigos de negociação.

```
#filtrando empresas com varios tickers pro mais líquido
df_info_pontuais['comeco_ticker'] = df_info_pontuais['ticker'].astype(str).str[0:4]
df_info_pontuais.sort_values('volume_negociado', ascending=False, inplace=True)
df_info_pontuais.drop_duplicates('comeco_ticker', inplace=True)
df_info_pontuais.drop('comeco_ticker', axis=1, inplace=True)
```

Portanto, o código irá identificar as 4 primeiras letras do ticker e realizar um ‘sort.values()’ na coluna de volume negociado. Depois disso, haverá um ‘drop_duplicates’ para toda vez que um ticker se repetir.

Na imagem abaixo, por exemplo, será mantido ‘VALE5’ e ‘PETR4’.

		index	ticker	preco...	volum...	rankin...	indicador_ROIC
	75	75	VALE5	33.01	1170959...	0	0.23689
	25	25	PETR4	8.42	720461220	0	0.07731
	32	32	OGXP3	1655	528720699	0	-0.02706
	41	41	VALE3	24.33	303159918	0	0.23689
	38	38	GFS3	516.22	196269310	0	0.05119
	104	104	ITSA4	3.42	180152440	0	-0.50976
	17	17	PETR3	10.65	165612560	0	0.07731

10.5. – Ranqueamento das empresas

Em resumo, o código percorre os indicadores financeiros, calcula o ranking de cada empresa com base nesses indicadores e soma os rankings individuais para criar um ranking geral. Em seguida, organiza as empresas em ordem crescente desse ranking.

Posteriormente, divide as empresas em quartis, agrupando-as em quatro conjuntos com base nesse ranking combinado.

Isso permite uma classificação das empresas em relação aos indicadores financeiros e facilita a segmentação em grupos para análises subsequentes.

```
for alfa, ordem in enumerate(self.ordem_indicadores):
    crescente_condicao = ordem.lower() == 'crescente'
    df_info_pontuais[f'ranking_{alfa}'] = df_info_pontuais[f'indicador_{self.indicadores[alfa]}'].rank(ascending=crescente_condicao)
    df_info_pontuais['ranking_final'] += df_info_pontuais[f'ranking_{alfa}']

df_info_pontuais.sort_values('ranking_final', ascending=True, inplace=True)

empresas_por_quartil = len(df_info_pontuais) // 4
sobra_empresas = len(df_info_pontuais) % 4
```

10.6. – Rentabilidade em quartis

Nesse fragmento do código da imagem abaixo, os quartis são criados como uma lista de DataFrame.

```
# dividindo o dataframe em quartis
lista_dfs[0] = df_info_pontuais.iloc[0: empresas_por_quartil]
lista_dfs[1] = df_info_pontuais.iloc[empresas_por_quartil: (empresas_por_quartil * 2)]
lista_dfs[2] = df_info_pontuais.iloc[(empresas_por_quartil * 2): (empresas_por_quartil * 3)]
lista_dfs[3] = df_info_pontuais.iloc[(empresas_por_quartil * 3): ((empresas_por_quartil * 4) + sobra_empresas)]
```

10.7. – Criação do DataFrame

Em resumo, essa parte final adiciona informações importantes ao DataFrame de prêmios, como nome do prêmio, nível de liquidez e identificadores únicos para cada linha, preparando os dados de maneira organizada antes de salvá-los ou utilizá-los em outras análises.

```
df_premios['nome_premio'] = self.nome_premio
df_premios['liquidez'] = self.liquidez
df_premios.reset_index(names='data', inplace=True)
df_premios['id_premio'] = \
    df_premios['nome_premio'].astype(str) + "_" + df_premios['liquidez'].astype(str) + "_" + df_premios['data'].astype(str)
df_premios.dropna(inplace=True)
self.df_premios = df_premios
```

10.8. – Gerando o parquet

Em uma linha de código, todo o backtest criado será exportado em um parquet, conforme definido o caminho dos dados.

```
def colocando_premio_na_base(self):
    self.df_premios.to_parquet(f'{self.caminho_salvar_arquivo}/{self.nome_premio}_{self.liquidez}.parquet',
                               index = False)
```

10.9. – Rodando os prêmios de risco

Na última parte do código, estão os detalhes acerca das informações complementares para rodar o backtest. Note que, é criado um indicador por vez e é extremamente rápido.

Ao criar os indicadores, também será criado um dicionário como argumento. Esse dicionário deve ter exatamente o mesmo nome do arquivo parquet gerado, e a ordem que o indicador deve ser ordenado, seja crescente ou decrescente. Por exemplo: indicadores_dict = {'ROE': 'decrescente'}

Depois, escolha a liquidez. Nesse caso, foi inserida a liquidez acima de 1 milhão para captar somente empresas líquidas. E por fim, coloque o nome do prêmio de risco.

```
if __name__ == "__main__":
    indicadores_dict = {'ROE': 'decrescente',
    }

    premio = premio_risco (indicadores_dict, liquidez = 1000000, nome_premio = 'QUALITY_ROE',
    caminho_dados= r'C:\Users\lsiqu\dev\base_dados_br',
    caminho_salvar_arquivo= r'C:\Users\lsiqu\dev\premios_risco_br'
    )

    premio.pegando_dados_cotacoes ()
    premio.pegando_datas_posseveis()
    premio.filtrando_volume ()
    premio.descobrindo_mes_inicial ()
    premio.calculando_premios ()
    premio.colocando_premio_na_base ()
```

Mundo 11

11.1. – Backtest de todos indicadores

Neste mundo, será ensinado a rodar o backtest com todos os indicadores criados.

É possível estar rodando um backtest com mais de um indicador. Para isso, acrescente o indicador no dicionário de indicadores ('indicadores_dict').

```
if __name__ == "__main__":
    indicadores_dict = {'ROIC': 'decrescente',
                        'ROE': 'decrescente'}
    premio = premio_risco(indicadores_dict, liquidez = 1000000, nome_premio = 'QUALITY_ROE',
                          caminho_dados=r'C:\Users\lsiqu\dev\base_dados_br',
                          caminho_salvar_arquivo=r'C:\Users\lsiqu\dev\base_dados_br\premios_risco_br'
                          ) #não pode ter \ no nome!!
    premio.pegando_dados_cotacoes()
    premio.pegando_datas_posseveis()
    premio.filtrando_volume()
    premio.pegando_indicadores()
    premio.descobrindo_mes_inicial()
    premio.calculando_premios()
    premio.colocando_premio_na_base()
```

```
if __name__ == "__main__":
    indicadores_dict = {'EBIT_EV': 'decrescente'}
```

Mundo 12

12.1. – Inicialização das variáveis

Para rodar esse arquivo “avaliar_premios_de_risco”, deve ser feita a inicialização das variáveis

```
class MakeResultsPremium:

    def __init__(self, dicionario_fatores, data_final_analise, caminho_imagens, nome_arquivo =
'premios_de_risco.pdf',
                 caminho_premios_de_risco = '.'):
        self.dicionario_fatores = dicionario_fatores
        self.lista_nome_fatores = []
        self.liquidez = []
        self.caminho_premios_de_risco = caminho_premios_de_risco

        for key, item in dicionario_fatores.items():

            self.lista_nome_fatores.append(key)
            self.liquidez.append(item)

        self.data_final_analise = (datetime.datetime.strptime(data_final_analise, '%Y-%m-%d')).date()
        self.caminho_imagens = caminho_imagens
        self.nome_arquivo = nome_arquivo
        os.chdir(caminho_imagens)
```

12.2. – Coletando dados

O método ‘puxando_dados’ é essencialmente para a coleta de dados, e realiza diversas operações sobre os dados de prêmio de risco, armazenado em arquivos parquet.

Ele começa um loop que percorre cada prêmio de risco presente na lista.

Além de identificar a data inicial geral para análise, considerando a data mais recente entre todas as datas iniciais dos prêmios. Em seguida, o código filtra o DataFrame ‘self.premios_de_risco’ para restringir as datas entre essa data inicial e ‘self.data_final_analise’.

Por fim, em ‘self.premios_de_risco’, é calculado o prêmio de risco (retorno do 1º quartil menos o retorno 4º quartil).

Posteriormente, é criado um DataFrame dos prêmios de risco que serão divididos em quartis e o universo.

```
def puxando_dados(self):  
  
    lista_dfs = []  
    data_inicial = []  
  
    for i, nome_premio in enumerate(self.lista_nome_fatores):  
        df = pd.read_parquet(f'{self.caminho_premios_de_risco}/{nome_premio}_{self.liquidez[i]}.parquet')  
        df['data'] = pd.to_datetime(df['data']).dt.date  
  
        lista_dfs.append(df)  
        data_inicial.append(min(df['data']))  
  
    self.premios_de_risco = pd.concat(lista_dfs)  
    data_inicial = max(data_inicial)  
  
    self.premios_de_risco = self.premios_de_risco[(self.premios_de_risco['data'] >= data_inicial) &  
                                                 (self.premios_de_risco['data'] <= self.data_final_analise)]  
  
    self.premios_de_risco = self.premios_de_risco.assign(premio_fator =  
                                         (1 + self.premios_de_risco['primeiro_quartil'])/(1 + self.premios_de_risco['  
                                         quarto_quartil']))  
  
    self.premios_de_risco['primeiro_quartil'] = 1 + self.premios_de_risco['primeiro_quartil']  
    self.premios_de_risco['segundo_quartil'] = 1 + self.premios_de_risco['segundo_quartil']  
    self.premios_de_risco['terceiro_quartil'] = 1 + self.premios_de_risco['terceiro_quartil']  
    self.premios_de_risco['quarto_quartil'] = 1 + self.premios_de_risco['quarto_quartil']  
    self.premios_de_risco['universo'] = 1 + self.premios_de_risco['universo']
```

Mundo 13

13.1. – Criação de PDF

Este mundo é bem curto. Basicamente é a explicação do método para fazer o pdf.

Esse código é responsável por criar um arquivo PDF que contém um relatório com múltiplas páginas. Esse relatório inclui imagens geradas a partir de diferentes análises e visualizações relacionadas aos fatores de risco de investimento.

A classe `MakePDF` é responsável pela criação do PDF. Ela inicializa o documento PDF, define cabeçalhos e rodapés, e adiciona páginas específicas para cada tipo de análise.

```
from fpdf import FPDF

class PDF(FPDF):
    def header(self):
        self.image('logo.png', 10, 8, 40)
        self.set_font('Arial', 'B', 20)
        self.ln(15)
        self.set_draw_color(35, 155, 132) #cor RGB
        self.cell(0, 15, f'Relatório dos Fatores',
                 border = True, ln = True, align = "C")
        self.ln(5)

    def footer(self):
        self.set_y(-15) #espaço ate o final da folha
        self.set_font('Arial', 'I', 10)
        self.cell(0, 10, f'{self.page_no()}/{self.nb}', align = "C")

    class MakePDF():
        def __init__(self, fatores, liquidez, matriz_correl,
                    nome_arquivo = "premios_de_risco.pdf", caminho_imagens = None):
            self.lista_nome_fatores = fatores
            self.liquidez = liquidez
            self.matriz_correl = matriz_correl

            self.nome_arquivo = nome_arquivo
            self.caminho_imagens = caminho_imagens

            self.pdf = PDF("P", "mm", "Letter")
            self.pdf.alias_nb_pages()
            self.pdf.add_page()
            self.pdf.set_fill_color(255, 255, 255)
            self.pdf.set_draw_color(35, 155, 132)

            self.primeira_pagina()
            self.segunda_pagina()

            for i, fator in enumerate(self.lista_nome_fatores):
                self.pagina_fator(nome_fator= fator, liquidez= self.liquidez[i])

            self.pdf.output(f'{self.nome_arquivo}')
```

Para facilitar a explicação, vamos dividir o PDF em 2 partes:

Página Inicial e de Análises Gerais: A primeira página contém duas imagens relacionadas à comparação dos primeiros quartis e prêmios de risco.

Na segunda página são exibidas as análises de matriz de correlação entre os fatores de risco, onde são exibidas as relações entre os fatores.

```
def primeira_pagina(self):
    self.pdf.image(f"{self.caminho_imagens}/comparando_1Q.png", w = 180, h = 90, x = 13, y = 60)
    self.pdf.image(f"{self.caminho_imagens}/comparando_premios.png", w = 180, h = 90, x = 13, y = 160)

def segunda_pagina(self):
    self.pdf.add_page()
    self.pdf.set_font('Arial', 'B', 9)
    self.pdf.cell(0, 15, 'Matriz de correlação', ln = True, align = 'C')
    self.pdf.cell(0, 5, ln = True)

    tamanho_tabela = len(self.matriz_correl)
    if tamanho_tabela > 20:
        print("Não é possível ter mais de 20 fatores no relatório!")
        exit()
```

Páginas de Análise Individual por Fator de Risco: Para cada fator de risco na lista `self.lista_nome_fatores`, são criadas páginas separadas com quatro imagens. Estas representam diferentes análises para o respectivo fator, incluindo gráficos de barras e linhas que mostram dados específicos e o comportamento do prêmio de risco ao longo do tempo.

Essas imagens são adicionadas às páginas do PDF usando a função `self.pdf.image()`, que especifica o caminho das imagens e suas posições dentro das páginas.

```
def pagina_fator(self, nome_fator, liquidez):
    self.pdf.add_page()
    self.pdf.image(f"{self.caminho_imagens}/barras_quartis_{nome_fator}_{liquidez}.png", w = 95, h = 80, x = 10, y = 60)
    self.pdf.image(f"{self.caminho_imagens}/linha_quartis_{nome_fator}_{liquidez}.png", w = 95, h = 80, x = 110, y = 60)
    self.pdf.image(f"{self.caminho_imagens}/movel_12m_premio_de_risco_{nome_fator}_{liquidez}.png", w = 95, h = 80, x = 10, y = 160)
    self.pdf.image(f"{self.caminho_imagens}/premio_de_risco_{nome_fator}_{liquidez}.png", w = 95, h = 80, x = 110, y = 160)
```

O PDF é criado e salvo no arquivo `self.nome_arquivo`, contendo informações detalhadas e visualizações referentes aos fatores de risco e seus respectivos prêmios em diferentes páginas para análises posteriores.

Mundo 14

14.1. – Comparando todos os fatores

Neste mundo, o intuito é ensinar e auxiliar na interpretação do relatório criado dos fatores. O objetivo não é falar qual fator é melhor ou pior, afinal, isso pode ser feito através dos backtests.

É com esses relatórios gerados que é possível saber quais fatores funcionam ou não no Brasil, mesmo sem ainda ter feito a Regressão Linear. Acontece que, por exemplo, temos o Momento 7 com 40, e pelo relatório, é um fator que funcionará, mesmo sem ainda ter feito a Regressão Linear. Isso porque, a Regressão Linear será um complemento dessas estatísticas.

É importante saber que, qualquer modelo de fatores vai ganhar do Ibovespa. Isso acontece por conta que o Ibovespa sempre pertencerá ao 4º quartil, que seria o pior. Afinal, as empresas que estão no Ibovespa são as maiores do Brasil, e segundo o fator tamanho: quanto maior a empresa, menos rentável ela tende a ser. Por conta disso, a rentabilidade do 4º quartil geralmente é puxada para baixo.

Após gerar o relatório dos fatores para analisar os prêmios de risco, opte primeiro por escolher um momentum para rodar com outros fatores. Ou seja, evite escolher "Momento R1M" e "Momento R12M" para rodar juntos. Portanto, escolha o melhor e rode com os fatores.

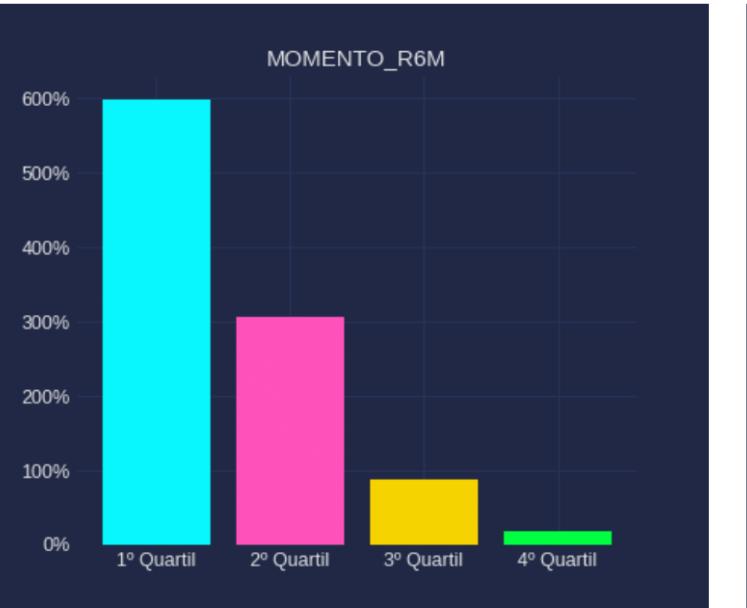
Esse pdf gerado pode ser dividido em 2 análises: Matriz de Correlação e a Distribuição de retorno ao longo dos quartis.

Em Factor Investing é necessário ter fatores descorrelacionados. Por exemplo, não é correto colocar "Momento R6M" juntamente com o "Momento MM 7 40", pois possuem uma correlação de 0.95, retratando as mesmas características. Portanto, evite correlação maior que 0.7.

A importância de selecionar fatores descorrelacionados é visando a diversificação e a diluição dos riscos. Afinal, se 2 fatores descorrelacionados geram retornos em momentos diferentes. Então, enquanto alguns ativos sobem, outros caem, e no final, a carteira estará sempre em ascensão.

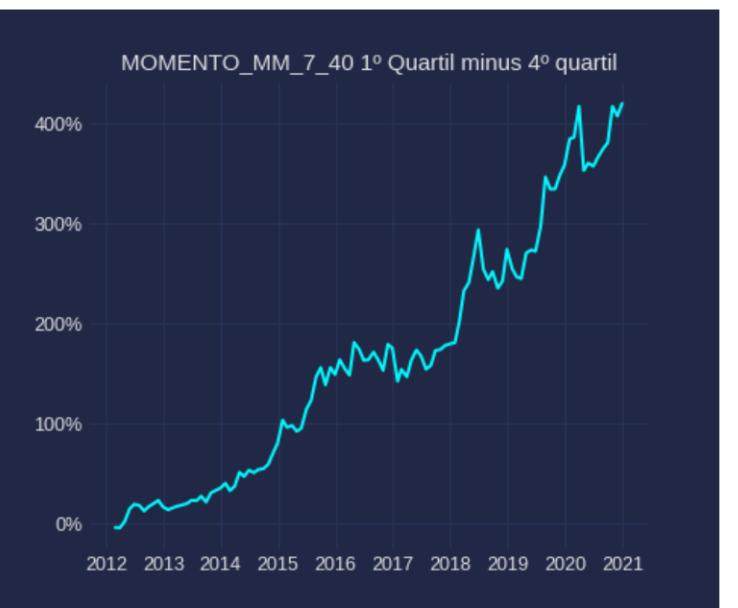
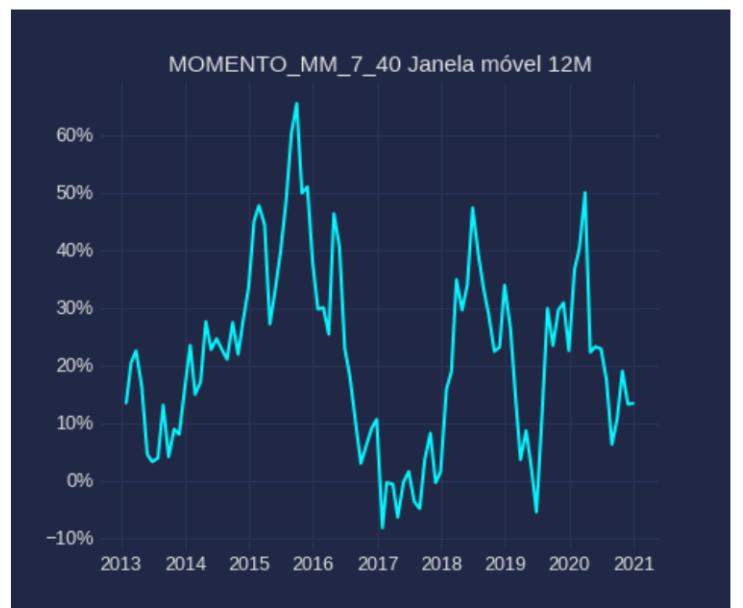
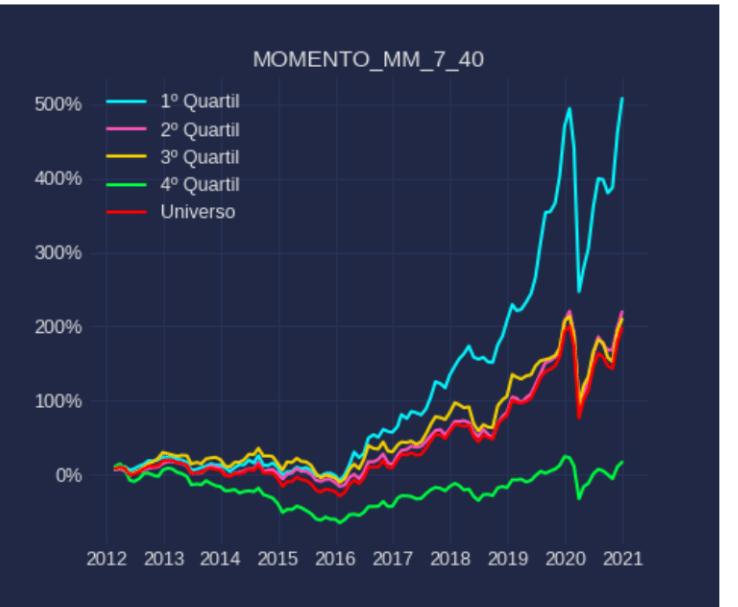
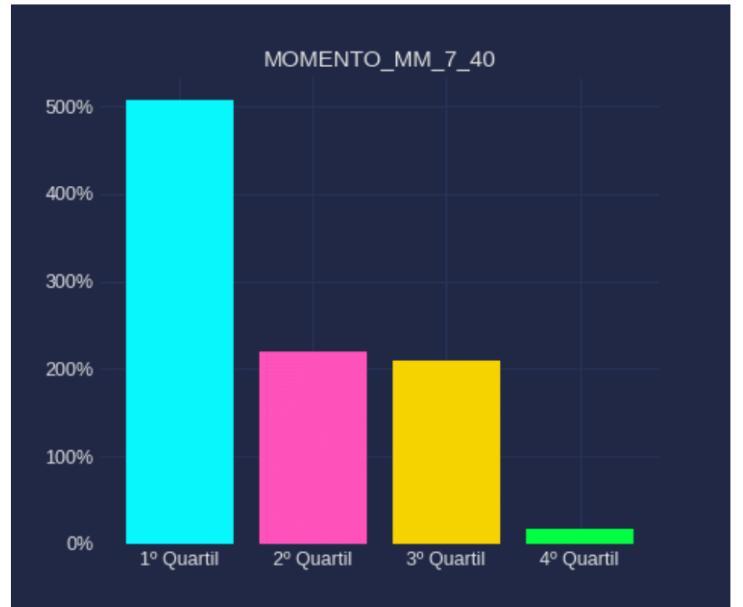
Quanto à distribuição de retorno ao longo dos quartis, é importante verificar os padrões de retorno. Os quartis devem respeitar uma ordem, como se fosse escadinha. Além disso, deve haver uma consistência nos retornos através dos gráficos, como por exemplo:

Relatório dos Fatores



O “Momento R6M” é um ponto fora da curva, como se fosse exemplo de um livro, seria o ideal para que ocorresse. Acontece que, isso não acontecerá com todos os fatores, mas havendo uma discrepância entre o 1º e o 4º quartil, e uma consistência nos retornos, igual no “Momento MM 7 40”, o modelo tem muito valor e funciona.

Relatório dos Fatores



Mundo 15

15.1. – Fator Beta

Este mundo é totalmente reservado para explicar uma peculiaridade que ocorre com o fator mercado (Beta) aqui no Brasil.

Como vem sendo demonstrado, o Beta é o fator fundamental no Factor Investing e está presente desde os primeiros modelos, iniciando no CAPM na década de 50.

Essa é a fórmula, e para calcular o Beta, é preciso ter o retorno do mercado menos o retorno da renda fixa.

$$R_i = R_f + \beta_i * (R_m - R_f)$$

Através desse cálculo, que seria o Ibovespa menos o CDI, é possível chegar no Market Premium, que é o prêmio de mercado ao longo do tempo. Contudo, esse prêmio é igual a zero, sendo um grande problema para realizar modelagem de Factor Investing. Isso acontece porque o Beta possui um valor estranho. Afinal, ao fazer a diferença entre o retorno do Ibovespa com o retorno do CDI, é retornado um valor negativo.

Mas, a modelagem será feita mesmo assim, pois não há nada a ser feito.

O código “fator_mercado” que calcula o market premium’. O método ‘calculando_premio’ inicia lendo os dados do CDI e do Ibovespa a partir de arquivos no formato parquet. Em seguida, o código calcula o retorno acumulado do CDI e do Ibovespa e cria um DataFrame com os retornos mensais.

Depois, é feita a junção desses DataFrames pela coluna 'data', calculando o market premium, que representa a diferença entre os retornos do Ibovespa e do CDI. O market premium é então armazenado em um novo DataFrame chamado ‘df_dados_mercado’.

Por fim, esse DataFrame é exportado como um parquet.

```
def calculando_premio(self):

    cdi = pd.read_parquet(f'{self.caminho_dados}/cdi.parquet')
    cdi['cota'] = (1 + cdi['retorno']).cumprod() - 1
    ibov = pd.read_parquet(f'{self.caminho_dados}/ibov.parquet')

    ibov_datas = ibov.sort_values('data', ascending = True)
    ibov_datas = ibov_datas.assign(year = pd.DatetimeIndex(ibov_datas['data']).year)
    ibov_datas = ibov_datas.assign(month = pd.DatetimeIndex(ibov_datas['data']).month)
    datas_final_mes = ibov_datas.groupby(['year', 'month'])['data'].last()
    dias_final_de_mes = datas_final_mes.to_list()

    ibov = ibov[ibov['data'].isin(dias_final_de_mes)]
    cdi = cdi[cdi['data'].isin(dias_final_de_mes)]
    ibov['retorno_ibov'] = ibov['fechamento'].pct_change()
    cdi['retorno_cdi'] = cdi['cota'].pct_change()
    ibov['data'] = ibov['data'].astype(str)
    cdi['data'] = cdi['data'].astype(str)

    df_dados_mercado = pd.merge(ibov, cdi, how = 'inner', on = "data")
    df_dados_mercado['mkt_premium'] = (1 + df_dados_mercado['retorno_ibov'])/(1 + df_dados_mercado['retorno_cdi']) - 1
    df_dados_mercado = df_dados_mercado.dropna()
    df_dados_mercado = df_dados_mercado[['data', 'mkt_premium']]
    df_dados_mercado['data'] = pd.to_datetime(df_dados_mercado['data']).dt.date

df_dados_mercado.to_parquet(f'{self.caminho_salvar_arquivo}/market_premium.parquet', index = False)
```

Mundo 16

16.1. – Modelo de 5 fatores do Fama French

Este mundo será focado em explicar o funcionamento da regressão linear para que possa ser realizado o modelo de 5 fatores do Fama French, e como ele pode ser aplicado no Brasil.

16.2. – Coleta de dados

Essa parte de código, sobre o método “puxando_dados_premios”, é basicamente a coleta e tratamento de dados, e a leitura dos parquets.

```
def puxando_dados_premios(self):
    df_premios = pd.read_parquet(f'{self.caminho_premios_de_risco}/market_premium.parquet')
    df_premios['data'] = pd.to_datetime(df_premios['data'])

    for i, nome_premio in enumerate(self.lista_nome_fatores):
        df = pd.read_parquet(f'{self.caminho_premios_de_risco}/{nome_premio}_{self.liquidez[i]}.parquet')
        df['data'] = pd.to_datetime(df['data'])

        df = df.assign(premio_fator = (1 + df['primeiro_quartil']) / (1 + df['quarto_quartil']) - 1)
        universo_fator = df[['data', f'universo']]
        universo_fator.columns = ['data', f'universo_{nome_premio}']

        if i == 0:
            universo = universo_fator

        else:
            universo = pd.merge(universo, universo_fator, on = 'data')

        df = df[['data', 'premio_fator']]
        df.columns = ['data', nome_premio]

    df_premios = pd.merge(df, df_premios, how = 'inner', on = 'data')

    df_premios = df_premios.drop(df_premios.index[0], axis = 0) #tirando a primeira linha por um detalhe no calculo do universo
    df_premios = df_premios.set_index('data')
    df_premios = df_premios[df_premios.index < self.data_final_analise]
    universo = universo[universo['data'] < self.data_final_analise]

    self.df_premios = df_premios
    self.universo = universo
```

16.3. – Cálculos para a Regressão Linear.

Existem milhões de formas para definir se um modelo é bom ou ruim, em cada paper existente que você ler durante sua jornada, haverá um tipo diferente de realizar a regressão.

O Eugene Fama realiza uma regressão de um jeito bem complexo, e não será ensinado neste mundo. Portanto, será feito uma regressão linear de um jeito mais simplificado, conforme pesquisadores brasileiros da FGV encontraram dessa forma. Essa forma é simplesmente regredir o retorno dos fatores e do prêmio de risco calculado, contra o retorno do “universo”.

O método “calculando_universo” tem como finalidade, calcular um indicador de universo relativo ao retorno de risco em relação a uma taxa livre de risco.

Cada fator possui seu próprio universo. A ideia aqui é juntar o universo dos 5 fatores, e assim calcular a média de todos os universos, chegando no “universo do universo”.

Isso significa que está sendo calculado o retorno médio de todas as ações, ou seja, o retorno médio do mercado.

Posteriormente, o DataFrame do CDI é ajustado para calcular a taxa de retorno livre de risco ('rf') com base na variação acumulada diária ('cota').

A ideia é chegar na fórmula abaixo, para que seja calculada a diferença entre o retorno do mercado e o retorno da renda fixa, para a regressão.

Fama – French Five-Factor Model

$$R_{it} - R_{Ft} = a_i + b_i(R_{Mt} - R_{Ft}) + s_iSMB_t + h_iHML_t + r_iRMW_t + c_iCMA_t + e_{it}$$

```
def calculando_universo(self):  
  
    universo = self.universo  
    universo = universo.set_index('data')  
    universo['universo_medio'] = universo.mean(axis = 1)  
    universo = universo.reset_index()  
    universo = universo[['data', 'universo_medio']]  
  
    cdi = pd.read_parquet(f'{self.caminho_cdi}/cdi.parquet')  
    cdi['data'] = pd.to_datetime(cdi['data'])  
    cdi['cota'] = (1 + cdi['retorno']).cumprod() - 1  
    cdi = cdi[cdi['data'].isin(universo['data'].to_list())]  
    cdi['rf'] = cdi['cota'].pct_change()  
    cdi = cdi.dropna()  
    cdi = cdi[['data', 'rf']]  
  
    universo = pd.merge(universo, cdi, how = 'inner', on = 'data')  
    universo['U_RF'] = (1 + universo['universo_medio'])/ (1 + universo['rf']) - 1  
    universo = universo.set_index('data')  
    self.universo = universo
```

16.4. – Regressão Linear

Esse trecho de código em Python contém um método chamado “regressao”, que realiza uma análise de regressão linear usando a biblioteca statsmodels.

As incógnitas X e Y, são representadas como os prêmios de riscos, e o universo, respectivamente.

Após isso, é criado um modelo de regressão ordinária dos mínimos quadrados (OLS) utilizando sm.OLS(Y, X_C), onde, “Y” representa a variável dependente e “X_C” a matriz de variáveis independentes, incluindo a constante adicionada.

O método fit() é então aplicado ao modelo, realizando a regressão propriamente dita, e o resultado é armazenado na variável resultado. Por fim, é impresso um resumo estatístico do resultado da regressão, utilizando resultado.summary() para mostrar estatísticas importantes como coeficientes, p-values, R-quadrado, entre outras métricas, que ajudam a entender a relação entre as variáveis e a qualidade do modelo de regressão linear

```
def regressao(self):  
    Y = self.universo['U_RF']  
    X = self.df_premios  
  
    X_C = sm.add_constant(X)  
  
    model = sm.OLS(Y, X_C)  
    resultado = model.fit()  
    print(resultado.summary())
```

Mundo 17

17.1. – Calculadora do Factor Investing

Nesse mundo será explicado o funcionamento da calculadora factor.

Como todo código de programação orientado à objeto, é necessário realizar o método `init` para inicialização.

Dessa vez, há algumas particularidades a serem explicadas, pois esse “`init`” é um pouco diferente dos “`init’s`” vistos até aqui.

Como é possível notar, existe um `**kargs`. O `**kargs` (abreviação de “keyword arguments”) é uma funcionalidade muito útil em Python que permite passar um dicionário contendo várias características como argumentos para uma função. Isso é especialmente útil quando se deseja testar diferentes indicadores ou qualquer outra quantidade que seja escolhida, sem a necessidade de criar múltiplos argumentos para cada indicador.

A variável `self.dinheiro_inicial` representa o dinheiro fictício utilizado para calcular os retornos. É um componente crucial para evitar distorções existentes em modelo que assume ter dinheiro infinito. Isso é importante porque pode gerar resultados enganosos ao realizar operações financeiras em um cenário irreal.

Por exemplo, ao executar um modelo que sempre realiza ordens de compra quando o mercado está em queda, se esse modelo pressupõe que há dinheiro infinito, ele pode gerar a ilusão de ser lucrativo. No entanto, essa suposição não condiz com a realidade, pois ao considerar um capital limitado, a rentabilidade poderá ser negativa, especialmente quando o mercado estiver em alta.

```
class backtest_indicators():

    def __init__(self, data_final, filtro_liquidez, balanceamento, numero_ativos, nome_indicador = None, ordem_indicador = None,
                 corretagem = 0, impacto_mercado = 0, data_inicial = None, nome_arquivo = 'backtest.pdf',
                 caminho_imagens = None, caminho_dados = None, **kargs):

        self.nome_arquivo = nome_arquivo
        self.caminho_imagens = caminho_imagens
        self.caminho_dados = caminho_dados

        if nome_indicador == None:
            self.indicadores = kargs
        else:
            self.indicadores = {'carteiral': {
                'indicadores': {
                    nome_indicador: {'caracteristica': ordem_indicador},
                    'peso': 1
                }
            }}

        self.balanceamento = balanceamento
        self.liquidez = filtro_liquidez

    try:
        data_inicial = datetime.datetime.strptime(data_inicial, "%Y-%m-%d").date()
        data_final = datetime.datetime.strptime(data_final, "%Y-%m-%d").date()
    except:
        data_final = datetime.datetime.strptime(data_final, "%Y-%m-%d").date()

        self.data_inicial = data_inicial
        self.data_final = data_final
        self.numero_ativos = numero_ativos
        self.corretagem = corretagem
        self.impacto_mercado = impacto_mercado
        self.dinheiro_inicial = 10000

        os.chdir(caminho_dados)
```

17.2. – Dicionário de carteirast

O processo de realização de backtest de implementação pode ocorrer de duas maneiras distintas para a construção da carteira:

1. Agregação em um único ranking supremo:

Nesta abordagem, os indicadores individuais, como EBIT dividido pelo lucro líquido, valor de mercado e média móvel de 7 com 40, são combinados em um único ranking, somando as classificações de cada indicador. Por exemplo:

$$\text{Ranking final} = \text{Ranking EBIT} + \text{Ranking Valor de Mercado} + \text{Ranking MM740}$$

Isso resulta em uma única carteira composta por, por exemplo, os 10 melhores ativos conforme a classificação final.

2. Combinação de rankings individuais:

Nessa abordagem, cada indicador é tratado separadamente, gerando rankings individuais. Por exemplo:

- Ranking do EBIT: 10 melhores ativos.
- Ranking da MM740: 10 melhores ativos.
- Ranking do Valor de Mercado: 10 melhores ativos.

Posteriormente, uma carteira é construída com um número variável de ativos, que não necessariamente será de 30 (soma dos ativos dos três rankings), pois pode haver sobreposição de empresas entre os rankings. Além disso, diferente da primeira forma, pesos diferentes podem ser atribuídos aos rankings para influenciar a seleção dos ativos na carteira final. Por exemplo, é possível definir o peso de cada ranking, como 0.2, 0.3, 0.5, para EBIT/MM740/Valor de Mercado, respectivamente, para ajustar a importância relativa de cada indicador na composição da carteira.

Assim, na segunda abordagem, há a flexibilidade de selecionar os pesos para cada ranking, permitindo uma ponderação diferenciada, potencialmente atribuindo maior peso a empresas presentes em múltiplos rankings simultaneamente, ao invés de utilizar pesos iguais para todos os indicadores como na primeira forma.

```
if __name__ == "__main__":
    dicionario_carteira = {
        'carteira1': {
            'indicadores': {
                'ROE': {'caracteristica': 'decrescente'},
            },
            'peso': 1
        },
    }
```

17.3. – Coleta e tratamento de dados

No método “pegando_dados”, será coletado os dados de cotações, e o volume mediano para aplicar um filtro. Além disso, será adquirido os indicadores, lendo os parquets e juntando-os, através de um merge. E por fim, um DataFrame será criado.

```
def pegando_dados(self):

    cotacoes = pd.read_parquet('cotacoes.parquet')
    cotacoes['data'] = pd.to_datetime(cotacoes['data']).dt.date
    cotacoes['ticker'] = cotacoes['ticker'].astype(str)
    self.cotacoes = cotacoes.sort_values('data', ascending=True)
    volume_mediano = pd.read_parquet('volume_mediano.parquet')
    volume_mediano['data'] = pd.to_datetime(volume_mediano['data']).dt.date
    volume_mediano['ticker'] = volume_mediano['ticker'].astype(str)
    volume_mediano = volume_mediano[['data', 'ticker', 'valor']]
    volume_mediano.columns = ['data', 'ticker', 'volume']

    lista_dfs = []

    lista_dfs.append(self.cotacoes)
    lista_dfs.append(volume_mediano)
    lista_indicadores_sem_rep = []

    for carteira in self.indicadores.values():
        indicadores = carteira['indicadores']

        for indicador in indicadores.keys():

            if indicador in lista_indicadores_sem_rep:
                pass

            else:

                lista_indicadores_sem_rep.append(indicador)
                lendo_indicador = pd.read_parquet(f'{indicador}.parquet')
                lendo_indicador['data'] = pd.to_datetime(lendo_indicador['data']).dt.date
                lendo_indicador['ticker'] = lendo_indicador['ticker'].astype(str)
                lendo_indicador['valor'] = lendo_indicador['valor'].astype(float)
                lendo_indicador = lendo_indicador[['data', 'ticker', 'valor']]
                lendo_indicador.columns = ['data', 'ticker', 'indicador']
                lista_dfs.append(lendo_indicador)

    df_dados = lista_dfs[0]

    for df in lista_dfs[1:]:
        df_dados = pd.merge(df_dados, df, how='inner', left_on=['data', 'ticker'], right_on=['data', 'ticker'])

    self.df_dados = df_dados.dropna()
```

No método “filtrando_datas”, essa etapa envolve filtrar o intervalo de datas desejado. Se não for especificada uma data inicial, pode-se adotar um método que defina a data inicial como “relative delta + 2”, que seria dois meses após da data final dos indicadores, garantindo um intervalo adequado para coletar os dados da carteira.

```
def filtrando_datas(self):  
    df_dados = self.df_dados  
  
    if self.data_inicial != None:  
        df_dados = df_dados[df_dados['data'] >= self.data_inicial]  
  
    else:  
        df_dados = df_dados[df_dados['data'] >= (min(df_dados['data']) + relativedelta(months=+2))]  
  
    df_dados = df_dados[df_dados['data'] < self.data_final]  
  
    self.pegando_dias_das_carteiras(df = df_dados)  
    df_dados = self.df_dados[self.df_dados['data'].isin(self.periodos_de_dias)]  
  
    self.df_dados = df_dados
```

No método “pegando_dias_das_carteiras”, é coletado as datas disponíveis que foram definidas no método “filtrando_datas”, e, em seguida, executar um loop conforme o balanceamento escolhido. O balanceamento envolve a definição de períodos específicos, como 21, 20, 60 ou 30 dias úteis, para a composição das carteiras de investimento.

```
def pegando_dias_das_carteiras(self, df):  
    datas_disponiveis = np.sort(df['data'].unique())  
    self.periodos_de_dias = [datas_disponiveis[i] for i in range(0, len(datas_disponiveis), self.balanceamento)]
```

Mundo 18

18.1. – Calculadora do Factor Investing | Parte 2

Continuaremos a explicação do código da calculadora de Factor Investing.

No método “criando_carteiras”, o código em questão executa um processo de criação automatizada de carteiras de investimento com base em indicadores predefinidos.

Primeiramente, os dados são filtrados considerando o critério de liquidez estabelecido (`self.liquidez`). Após essa filtragem inicial, é feita uma seleção dos ativos mais líquidos, considerando somente um ticker por empresa. Por exemplo, entre PETR3 e PETR4, será considerado somente o ativo mais líquido.

Em seguida, o código itera sobre um conjunto de indicadores pré definidos, onde cada indicador possui um conjunto específico de características. Para cada indicador, uma carteira é construída com base nos dados filtrados. Os ativos são classificados em cada período de acordo com os critérios estabelecidos para cada indicador, gerando rankings individuais. Esses rankings são combinados para criar um "ranking final" para cada ativo.

Portanto, conforme explicado em “kargs”, o funcionamento do dicionário de carteiras, aqui, estaremos utilizando a 2^a forma da construção de carteira.

Então, os ativos são selecionados para compor a carteira com base em um número determinado (`self.numero_ativos`), usando essa classificação. O peso de cada ativo na carteira é calculado de acordo com as proporções estabelecidas para cada carteira. Essas carteiras são armazenadas em uma lista para posterior consolidação.

Ao final, todas as carteiras são concatenadas em um único DataFrame chamado `carteira_por_periodo`. Os ativos são agrupados por data e ticker, somando os pesos de cada ativo em cada período.

O resultado final é um DataFrame que indica a composição das carteiras em diferentes períodos, mostrando a data, o ticker dos ativos e seus respectivos pesos nas carteiras. Esse procedimento automatizado permite criar estratégias de investimento baseadas em indicadores definidos e analisar o desempenho dessas estratégias ao longo do tempo.

```
def criando_carteiras(self):  
    df_dados = self.df_dados  
    df_dados = df_dados[df_dados['volume'] > self.liquidez]  
    # Mantém o maior volume se as 4 primeiras letras do ticker forem iguais  
    df_dados = df_dados.assign(TICKER_PREFIX = df_dados['ticker'].str[:4])  
    df_dados = df_dados.loc[df_dados.groupby(['data', 'TICKER_PREFIX'])['volume'].idxmax()]  
    df_dados = df_dados.drop('TICKER_PREFIX', axis = 1)  
  
    lista_df_carteiras = []  
  
    for nome_carteira, carteira in self.indicadores.items():  
        df_carteiras = df_dados.copy()  
        df_carteiras[f'RANK_FINAL_{nome_carteira}'] = 0  
        indicadores = carteira['indicadores']  
  
        for indicador, ordem in indicadores.items():  
            crescente_condicao = (ordem['caracteristica'] == 'crescente')  
            # Crie os rankings para os indicadores  
            df_carteiras[f'{indicador}_RANK_{nome_carteira}'] = df_carteiras.groupby('data')[indicador].rank(ascending = crescente_condicao)  
            df_carteiras[f'RANK_FINAL_{nome_carteira}'] = df_carteiras[f'RANK_FINAL_{nome_carteira}'] + df_carteiras[f'{indicador}_RANK_{nome_carteira}']  
  
        df_carteiras[f'posicao_{nome_carteira}'] = df_carteiras.groupby('data')[f'RANK_FINAL_{nome_carteira}'].rank()  
        portfolio = df_carteiras[df_carteiras[f'posicao_{nome_carteira}'] <= self.numero_ativos]  
        portfolio = portfolio.assign(peso = carteira['peso']/(portfolio.groupby('data').transform('size')))  
        lista_df_carteiras.append(portfolio)  
  
    carteira_por_periodo = pd.concat(lista_df_carteiras, ignore_index=True)  
    carteira_por_periodo = carteira_por_periodo.sort_values('data', ascending=True)[['data', 'ticker', 'peso']]  
  
    carteira_por_periodo = carteira_por_periodo.groupby(['data', 'ticker'])['peso'].sum()  
  
    self.carteira_por_periodo = carteira_por_periodo.reset_index()
```

Mundo 19

19.1. – Calculadora do Factor Investing | Parte 3

Este código executa um cálculo de retorno diário para um portfólio de investimentos. A função `calculando_retorno_diario` inicialmente seleciona as cotações de ativos dentro do intervalo de tempo definido pelo primeiro dia da carteira (`self.carteira_por_periodo.iloc[0, 0]`) até a data final (`self.data_final`).

Posteriormente, é criado um DataFrame chamado `df_retornos` com 4 colunas (data, dinheiro, nº trade e retorno).

Após isso, algumas variáveis são criadas ou ajustadas para serem usadas posteriormente. `cotacoes` é atualizado com uma nova coluna chamada `var_fin`, que representa a variação financeira dos preços ajustados de fechamento dos ativos.

A variável `retorno_fin` é definida como um DataFrame contendo as colunas 'data', 'ticker' e 'var_fin'.

Em seguida, os índices dos DataFrames `retorno_fin`, `carteiras`, e `cotacoes_rebalanceamento` são ajustados para usar as colunas 'data' e 'ticker' como índices, o que simplifica a manipulação e fusão dos dados com base nesses rótulos. Esses ajustes são realizados usando o método `set_index` do pandas.

Em seguida, é feita uma iteração pelas datas presentes nesse intervalo, onde o código calcula o retorno financeiro diário da carteira. A cada dia, ele verifica se há datas de rebalanceamento da carteira. Se houver, o algoritmo ajusta a composição da carteira, comprando ou vendendo ativos de acordo com a nova alocação definida nas datas de rebalanceamento.

Durante a iteração diária, ele calcula o retorno financeiro do portfólio, considerando as variações diárias dos preços dos ativos, atualizando o valor financeiro da carteira e a composição dos ativos de acordo com as proporções estabelecidas. Essa simulação considera o custo de transação (corretagem e impacto de mercado) ao fazer as operações de compra e venda de ativos.

19.2. – Criando e rebalanceando a carteira

Na última etapa do código, é realizado o rebalanceamento da carteira

A carteira é montada e comprada no mercado no dia seguinte ao fechamento, ou seja, em D+1. Esse atraso na execução da carteira é conhecido como "delay" e nesse caso específico, existe a linha de código "delay = delay + 1".

Na estrutura de condição "else", o primeiro cálculo determina o valor disponível em dinheiro por ação dentro da carteira. Essa quantia é calculada multiplicando o peso de cada ativo na carteira pelo valor total de dinheiro disponível para investir. Além disso, são considerados fatores como custos de corretagem e possíveis impactos no mercado.

Após isso, é criado o DataFrame `carteira_vigente` combinando os dados da carteira na data de rebalanceamento com as cotações dos ativos nessa mesma data. Isso é realizado utilizando a função `pd.merge`, essencial para juntar esses conjuntos de dados.

Com base nos valores obtidos, é calculada a quantidade de ações que podem ser adquiridas para cada ativo. Essa quantidade é obtida dividindo o valor disponível em dinheiro por ação pelo preço de fechamento ajustado do ativo.

Por fim, o código atualiza o contador de carteira, indicando que houve uma troca de carteira nesse ponto do código, e define a variável `trocar_carteira` como `False`, sinalizando que não há necessidade de alterar a composição da carteira até a próxima data de rebalanceamento.

```

if data in datas_rebalanceamento:
    carteira_na_data = carteiras.loc[data].copy()
    trocar_carteira = True
    delay = 0

if trocar_carteira:
    if delay == 0:
        delay = delay + 1 #eu vou simular que eu só compro as ações no preço de fechamento do dia seguinte.

    else:
        carteira_na_data["dinheiro_por_acao"] = (carteira_na_data["peso"] * df_retornos.iloc[i, 1]) * (1 - self.corretagem) * (1 - self.impacto_mercado)
        cotacoes_na_data = cotacoes_rebalanceamento.loc[data]
        carteira_vigente = pd.merge(carteira_na_data, cotacoes_na_data, left_index=True, right_index=True)
        carteira_vigente["quantidade_acoes"] = carteira_vigente["dinheiro_por_acao"] / carteira_vigente["preco_fechamento_ajustado"]
        carteira += 1
        trocar_carteira = False

```

Ao final, o código calcula o retorno percentual do portfólio para cada dia com base nos valores financeiros diários. O resultado é armazenado em um DataFrame chamado `df_retornos`, contendo informações sobre a data, o valor financeiro, o número de trades realizados e o retorno percentual do portfólio para cada dia.

```

def calculando_retorno_diario(self):
    cotacoes = self.cotacoes[(self.cotacoes['data'] >= self.carteira_por_periodo.iloc[0, 0]) &
                               (self.cotacoes['data'] <= self.data_final)]
    datas_carteira = cotacoes['data'].unique()
    df_retornos = pd.DataFrame(columns=['data', 'dinheiro', 'numero_trade'], index = list(range(0, len(datas_carteira))))
    carteira = 0
    df_retornos.iloc[1, 0] = self.carteira_por_periodo.iloc[1, 0] #segunda data (a primeira foi pra gerar o sinal)
    df_retornos.iloc[1, 1] = self.dinheiro_inicial
    df_retornos.iloc[1, 2] = carteira

    cotacoes = cotacoes.assign(var_fin = cotacoes.groupby('ticker')['preco_fechamento_ajustado'].diff())
    retorno_fin = cotacoes[['data', 'ticker', 'var_fin']]
    carteiras = self.carteira_por_periodo.copy()
    datas_rebalanceamento = carteiras['data'].unique()
    cotacoes_rebalanceamento = cotacoes[['ticker', 'data', 'preco_fechamento_ajustado']]
    retorno_fin.set_index(["data", "ticker"], inplace=True)
    carteiras.set_index(["data", "ticker"], inplace=True)
    cotacoes_rebalanceamento.set_index(["data", "ticker"], inplace=True)

    for i, data in enumerate(datas_carteira):

        if i not in [0, 1]:
            retorno_fin_dia = retorno_fin.loc[data]
            var_patrimonio_no_dia = (carteira_vigente["quantidade_acoes"] * retorno_fin_dia["var_fin"]).sum()
            df_retornos.iloc[i, 0] = data
            df_retornos.iloc[i, 1] = df_retornos.iloc[i - 1, 1] # Inicializando com o valor do dia anterior
            df_retornos.iloc[i, 1] += var_patrimonio_no_dia # Agora a operação de adição deve funcionar corretamente
            df_retornos.iloc[i, 2] = carteira

        if data in datas_rebalanceamento:
            carteira_na_data = carteiras.loc[data].copy()
            trocar_carteira = True
            delay = 0

        if trocar_carteira:
            if delay == 0:
                delay = delay + 1 #eu vou simular que eu só compro as ações no preço de fechamento do dia seguinte.

            else:
                carteira_na_data["dinheiro_por_acao"] = (carteira_na_data["peso"] * df_retornos.iloc[i, 1]) * (1 - self.corretagem) * (1 - self.impacto_mercado)
                cotacoes_na_data = cotacoes_rebalanceamento.loc[data]
                carteira_vigente = pd.merge(carteira_na_data, cotacoes_na_data, left_index=True, right_index=True)
                carteira_vigente["quantidade_acoes"] = carteira_vigente["dinheiro_por_acao"] / carteira_vigente["preco_fechamento_ajustado"]
                carteira += 1
                trocar_carteira = False

        df_retornos = df_retornos.assign(retorno = df_retornos['dinheiro'].pct_change())
        df_retornos = df_retornos.drop(0, axis = 0)

    self.df_retornos = df_retornos

```

Mundo 20

Neste mundo, será destrinchado o funcionamento do pdf. Apesar de ser um código bem extenso, com aproximadamente 500 linhas, são apenas formatação de dados, e é bem intuitivo.

Esse código está estruturado em 2 etapas: cálculo das estatísticas e a plotagem de gráficos e imagens.

20.1. – Estatísticas de trade, retorno e risco

A primeira página do PDF é composta pelo período que foi realizado o backtest, juntamente com o gráfico que representa o retorno acumulado do CDI, Ibovespa e do modelo.

Também é composto por diversas estatísticas de retorno e de risco do modelo.

Uma delas é o Turn Over, informando-o que, quanto maior o Turn Over, maior a corretagem. Porém, a corretagem é irrelevante para o modelo, afinal, o modelo continuará lucrativo, e a carteira será rebalanceada somente 1 vez por mês.

codigo.py

**Estatísticas de Retorno e Risco**

Retorno acum. modelo	285.08%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	12.61%
Vol 252d	22.35%
Índice Sharpe	0.13
VAR diário 95%	-2.3%
Turn over carteira	10.88%
Drawdown máximo	-57.33%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	58.39%
% Operações perdedoras	41.61%
Média de ganhos	5.75%
Média de perdas	-4.91%
Expec. matemática por trade	1.31%
Maior sequência de vitória	5
Maior sequência de derrotas	4
% Meses carteira > IBOV	55.07%

Para calcular o dia inicial, final e dias finais do modelo, é através do método “periodo_backtest”.

```
def periodo_backtest(self):
```

```
    self.dia_inicial = self.df_trades['data'].iat[0]
    self.dia_final = self.df_trades['data'].iat[-1]

    self.dias_totais_backtest = len(self.df_trades)
```

Esses foram os métodos utilizados para calcular as estatísticas de retorno e risco: “retorno_risco”, “turnover_carteira”, “drawdown”, “calcula_drawdown”.

```

def retorno_risco(self):

    self.retorno_acum_modelo = self.df_trades['dinheiro'].iat[-1]/self.df_trades['dinheiro'].iat[0] - 1
    self.retorno_acum_cdi = ((self.cdi['retorno'] + 1).cumprod() - 1).iat[-1]
    self.retorno_acum_ibov = ((self.ibov['retorno'] + 1).cumprod() - 1).iat[-1]

    self.retorno_aa_modelo = (1 + self.retorno_acum_modelo) ** (252/self.dias_totais_backtest) - 1
    self.retorno_aa_cdi = (1 + self.retorno_acum_cdi) ** (252/self.dias_totais_backtest) - 1

    self.vol_ultimo_ano = ((self.df_trades['retorno'].iloc[-253:-1]).std()) * np.sqrt(252)
    self.vol_periodo = (self.df_trades['retorno'].std()) * np.sqrt(252)

    self.sharpe = (self.retorno_aa_modelo - self.retorno_aa_cdi)/self.vol_periodo

    dia_95 = int(self.dias_totais_backtest * 0.05)
    self.var_diario = self.df_trades['retorno'].sort_values(ascending = True).iat[dia_95]

```

```

def turnover_carteira(self):
    # Cria um DataFrame com a data, ticker e um contador de empresas
    turnover_df = self.carteiras[['ticker']].copy()
    turnover_df['contador'] = 1

    # Transforma a data no índice e faz um 'shift' para comparar a presença de empresas em datas consecutivas
    #turnover_df.set_index('data', inplace=True)
    turnover_df = turnover_df.groupby(['data', 'ticker']).count().unstack(fill_value=0)

    turnover_anterior = turnover_df.shift()
    entrou = (turnover_df - turnover_anterior) > 0
    saiu = (turnover_df - turnover_anterior) < 0

    # Calcula o turnover, que é a soma das empresas que saem e entram, dividido pelo total de empresas
    total_anterior = turnover_anterior.sum(axis=1)
    total_atual = turnover_df.sum(axis=1)
    turnover_df = (saiu.sum(axis=1) + entrou.sum(axis=1)) / (total_anterior + total_atual)

    self.turn_over_medio = turnover_df.mean()

```

```

def drawdown(self):
    self.dd_all = self.calcula_drawdown(len(self.df_trades)) # all time

def calcula_drawdown(self, janela):
    df = self.df_trades.copy()

    df['maximo'] = df['dinheiro'].rolling(janela, min_periods=1).max()
    df['drawdown'] = (df['dinheiro'])/df['maximo'] - 1
    df['drawdown_max'] = df['drawdown'].rolling(janela, min_periods=1).min()

    self.drawdowns = df['drawdown']
    self.drawdowns.index = df['data']

    return min(df['drawdown_max'])

```

E assim foi estruturado o método “estatisticas_de_trade”, que calcula justamente as estatísticas do trade.

As operações vencedoras e perdedoras, bem como a média de ganho e perda, são elementos essenciais para calcular a expectativa matemática de um modelo de investimento. Esta expectativa precisa ser positiva para validar a viabilidade do modelo. Se a expectativa matemática for negativa, é um sinal de que o modelo não é adequado e possivelmente está sofrendo de overfitting. Mesmo que o modelo aparente ser lucrativo, se a expectativa matemática for negativa, é recomendado descartá-lo. Isso ocorre porque um modelo com expectativa negativa pode parecer eficaz no curto prazo, mas é improvável que funcione consistentemente no longo prazo.

```
def estatisticas_de_trade(self):  
    self.numero_trades = self.df_trades['numero_trade'].max()  
    self.df_trades['retorno_plus_one'] = self.df_trades['retorno'] + 1  
    self.df_trades['retorno_por_trade'] = self.df_trades.groupby('numero_trade')['retorno_plus_one'].cumprod() - 1  
    trades_acum = (self.df_trades.groupby(['numero_trade']).tail(1))['retorno_por_trade']  
    trades = trades_acum.dropna().unique()  
    self.operacoes_vencedoras = np.sum(np.where(trades > 0, True, False))/self.numero_trades  
    self.operacoes_perdedoras = 1 - self.operacoes_vencedoras  
    self.media_ganhos = trades[trades > 0].mean()  
    self.media_perdas = trades[trades < 0].mean()  
    self.expectativa_matematica = (self.operacoes_vencedoras * self.media_ganhos) - (self.operacoes_perdedoras * abs(self.media_perdas))  
    self.maior_sequencia_vitorias = len(max([(list(g) if k else [] for k, g in groupby(trades.tolist(), key=lambda i: i > 0)), key=len]))  
    self.maior_sequencia_derrotas = len(max([(list(g) if k else [] for k, g in groupby(trades.tolist(), key=lambda i: i < 0)), key=len]))  
    ibov = self.ibov.copy()  
    ibov.columns = ['data', 'fechamento', 'retorno_ibov']  
    df_trades_ibov = pd.merge(self.df_trades, ibov, on='data', how='inner')  
    df_trades_ibov['retorno_acum_ibov'] = (df_trades_ibov.groupby('numero_trade', group_keys=False)['retorno_ibov'].apply(lambda x: (1 + x).cumprod() - 1))  
    df_trades_ibov['retorno_por_trade'] = (df_trades_ibov.groupby('numero_trade', group_keys=False)['retorno'].apply(lambda x: (1 + x).cumprod() - 1))  
    retorno_ibov_por_trade = df_trades_ibov.groupby('numero_trade')['retorno_acum_ibov'].last()  
    retorno_por_trade = df_trades_ibov.groupby('numero_trade')['retorno_por_trade'].last()  
    superou_trades = retorno_por_trade > retorno_ibov_por_trade  
    self.percentual_cart_supera_ibov = superou_trades.mean()
```

2O.2. – Eventos de estresse, períodos sem lucro e retornos do modelo

O gráfico mais importante desse relatório é o gráfico de underwater, o gráfico dos azarados. Cada ponto no gráfico representa uma máxima histórica, mostrando o desempenho de investimentos hipotéticos feitos no ponto mais alto registrado pelo modelo. Esse gráfico é conhecido como "gráfico dos azarados" e serve para calcular quanto dinheiro teria sido perdido se alguém tivesse investido exatamente nas máximas históricas identificadas pelo modelo. Quando o gráfico atinge 0%, isso indica que todos os investidores teriam lucrado com o modelo ou carteira em questão.

Essa análise oferece uma visão sobre o desempenho do modelo ao longo do tempo. Por exemplo, suponha que alguém tenha investido em 2013 e mantido esse investimento até 2016 sem ver lucros. Isso proporciona uma noção mais realista do risco associado ao modelo.

Essa análise também aborda uma das questões mais cruciais para o modelo: qual é a probabilidade de alguém investir no modelo exatamente no dia inicial? Geralmente, essa probabilidade é próxima de zero. Portanto, ao investir, a pessoa pode ter o azar de investir em um ponto que representa uma máxima histórica, o que poderia resultar em perdas significativas.

Ou seja, o que acontece se investir no modelo em um dia completamente aleatório?

Portanto, está contabilizado os períodos em que o modelo esteve sem lucro.

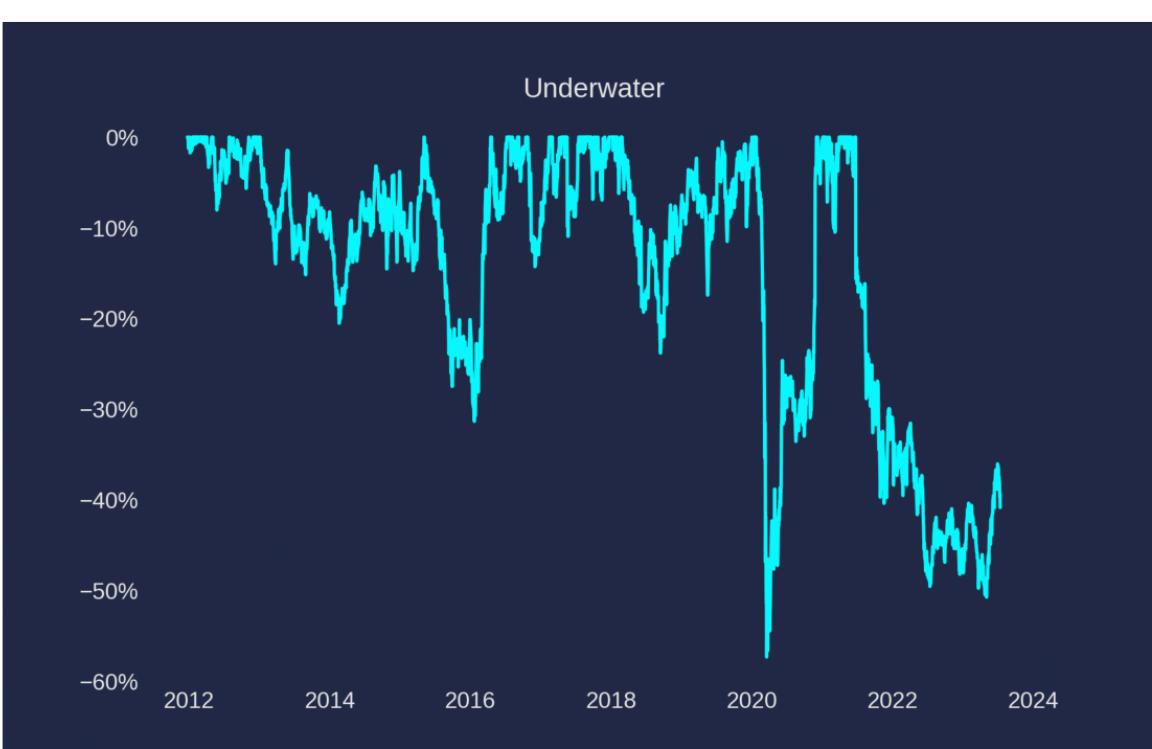
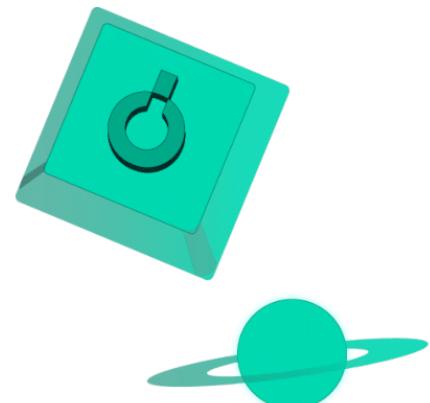
Além disso, há os eventos de estresse, em que a bolsa caiu drasticamente, como no Joesley Day, Auge da Pandemia, Greve dos caminhoneiros e precatórios.

código.py

Relatório do Modelo

Eventos de estresse	
Joesley Day - 18/05/2017	-7.96%
Auge pandemia: Março - 2020	-41.46%
Boa sorte Day - 10/11/2022	-3.81%
Greve dos caminhoneiros - 2018	-8.05%
Precatórios - ago/nov 2021	-24.55%
Crise de 2008	-

Períodos sem lucro	
Pior período de 1 mês	-53.78%
Pior período de 3 meses	-53.17%
Pior período de 6 meses	-52.99%
Pior período de 1 ano	-53.94%
Pior período de 2 anos	-54.11%
Pior período de 3 anos	-29.96%



Na 3º página do relatório, é basicamente o retorno do modelo ano a ano, e mês a mês. Acontece que, o modelo pode ganhar do bovespa, em 7 de 12 anos, mas e daí?

código.py

Relatório do Modelo

Retorno ano a ano

	MODELO	CDI	IBOV
2011	-0.568%	0.246%	-1.04%
2012	29.9%	8.4%	7.4%
2013	-7.37%	8.06%	-15.5%
2014	4.43%	10.8%	-2.91%
2015	-15.7%	13.2%	-13.3%
2016	33%	14%	38.9%
2017	86.9%	9.93%	26.9%
2018	4.17%	6.42%	15%
2019	7.62%	5.96%	31.6%
2020	11.5%	2.76%	2.92%
2021	37.6%	4.42%	-11.9%
2022	-21.7%	12.4%	4.69%
2023	9.22%	7.04%	7.27%

Retorno mês a mês

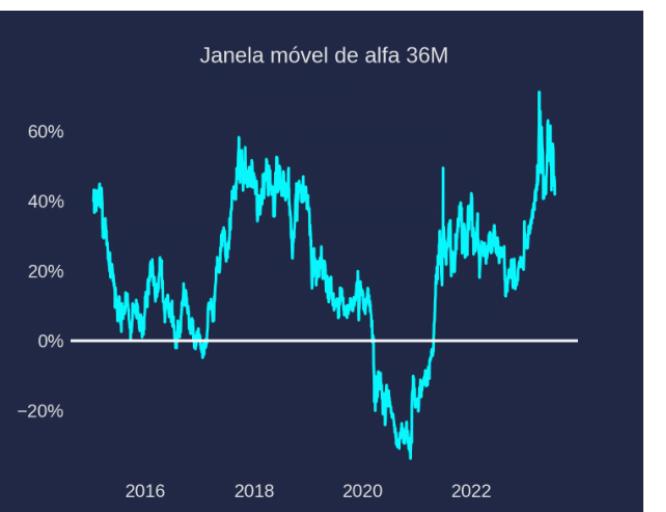
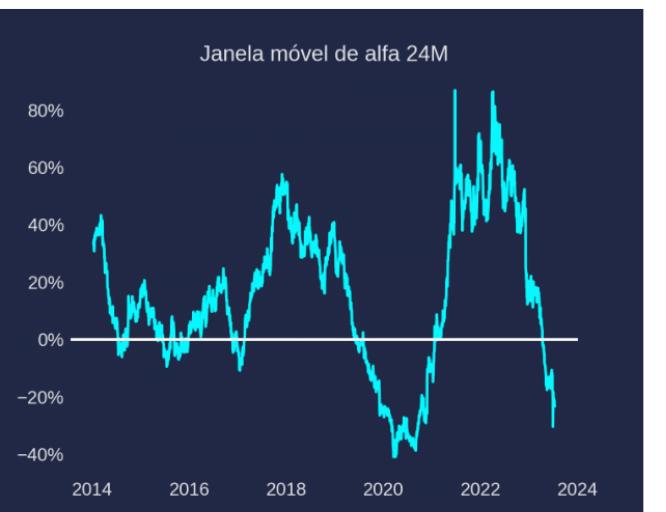
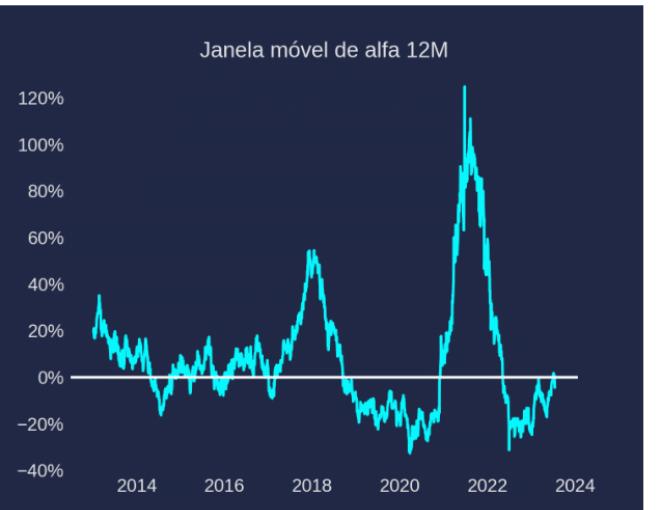
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2011	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2012	-3%	9.1%	4.4%	2.7%	-4.8%	4.6%	1.1%	-0.7%	-2.3%	1.3%	5%	3.6%
2013	-4.3%	-3%	-2.5%	5%	2.6%	-7.1%	0.068%	-3%	6.4%	0.53%	-1.2%	-0.1%
2014	-8.6%	-2.9%	5.3%	0.89%	1.9%	4.1%	-2.8%	8.6%	-4.8%	1.1%	2.8%	0.0019%
2015	-9.3%	2.5%	-1.1%	10%	-1.7%	-1.7%	-3.2%	-7.7%	-11%	4%	-1.6%	5.5%
2016	-6.8%	6.3%	15%	6.9%	-5.6%	8.1%	7.4%	1.7%	1.4%	4%	-9.7%	2.6%
2017	-4.6%	12%	-0.9%	6.4%	0.25%	0.078%	12%	12%	7.9%	3%	-0.58%	9.4%
2018	-9.6%	2.1%	-1.6%	-3.7%	-7.2%	-4.1%	4.4%	-4.7%	-4.4%	13%	3.4%	-0.46%
2019	-2.2%	1.2%	-1.5%	0.012%	-2%	1.4%	5.6%	-5.7%	2.4%	2.4%	2.4%	-0.55%
2020	-0.93%	-15%	-39%	20%	1.5%	18%	2.2%	-5.7%	0.17%	1.9%	40%	9.9%
2021	-4.7%	-3.3%	11%	20%	16%	8.8%	-3.9%	-9.9%	-3.6%	-15%	0.54%	14%
2022	-7.4%	-2.7%	7.3%	-6.8%	-1.1%	-16%	5.8%	1.5%	-2.3%	5.3%	-2.2%	-3%
2023	-10%	-6.1%	-5.8%	-4.2%	11%	14%	-7.5%	0%	0%	0%	0%	0%

O que realmente importa são as análises baseadas nas janelas móveis, que indicam em que porcentagem do tempo o modelo superou o desempenho do Ibovespa.

Em outras palavras, a métrica significativa é determinar a frequência em que o modelo se mantém acima de 0% durante uma janela móvel de 12 meses, por exemplo. Isso oferece uma visão mais precisa e detalhada do desempenho relativo do modelo em relação ao Ibovespa ao longo do tempo, considerando períodos contínuos de 12 meses em que o modelo supera ou fica abaixo do desempenho do índice de referência.

Essa análise das janelas móveis ajuda a entender não apenas a capacidade do modelo de superar o Ibovespa em um período específico, mas também a consistência desse desempenho ao longo do tempo, oferecendo uma métrica mais robusta para avaliar a eficácia e a confiabilidade do modelo em comparação com o benchmark de mercado.

Relatório do Modelo



Mundo 21

21.1. – Capacity e backtest de preço/lucro

Finalmente será iniciado os backtest dos modelos.

Contudo, é necessário explicar sobre a capacity dos modelos de Factor Investing, que é “Quanto dinheiro o meu modelo aguenta?”.

Para responder essa pergunta, é necessário responder outras 5, que definem a fórmula da capacity.

- Qual é meu filtro de liquidez? Supondo que seja 1 milhão.
- Qual é o número de ativos da carteira? Supondo que sejam 10 ativos.
- Qual é a porcentagem desses ativos? Supondo que o peso de cada ativo seja 10%.
- Quantos dias eu quero demorar para comprar o modelo? Supondo que seja 5 dias.
- Quanto eu aceito negociar do volume diário de uma ação por dia? O máximo é 20%.

Capacity = (Filtro Liquidez * % do volume negociado * dias que demora para comprar o modelo) / % em cada ativo

$$1M * 0,2\% = 200k/0,1 = 2M * 5 = 10M$$

Com os números utilizados como exemplo, o modelo aguentaria 10 milhões de reais.

Outro detalhe importante, é o Trade-Off entre o número de ativos e a liquidez.

Se você possui 10 ativos e uma liquidez de 1 milhão de reais, no pior cenário, ou seja, se a pior empresa representar 10% da carteira, então o valor total da carteira será de, no máximo, 10 milhões de reais. Nesse contexto, é fundamental observar que não é possível realizar negociações que ultrapassem o valor total disponível na carteira.

Além disso, existe uma limitação em relação ao volume de negociação diária. Se considerarmos que apenas 20% da negociação do dia é permitida, então, nos cálculos, seria possível negociar até 200 mil reais por dia, com base em uma carteira de 2 milhões de reais.

Esses limites são fundamentais para garantir que as operações estejam dentro das capacidades financeiras disponíveis e que as negociações não excedam os limites estabelecidos pela liquidez.

Rodando o backtest do modelo do Preço sobre Lucro com 10 ativos e filtro de liquidez de 1 milhão, esse foi o resultado:

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	125.15%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	7.41%
Vol 252d	25.43%
Índice Sharpe	-0.06
VAR diário 95%	-2.6%
Turn over carteira	21.31%
Drawdown máximo	-67.2%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	54.01%
% Operações perdedoras	45.99%
Média de ganhos	6.97%
Média de perdas	-5.95%
Expec. matemática por trade	1.03%
Maior sequência de vitória	9
Maior sequência de derrotas	7
% Meses carteira > IBOV	50.72%

Mundo 22

22.1. – Backtest EV/EBIT

Nesse backtest, será testado o modelo utilizando o indicador Enterprise Value (EV) / EBIT.

Esse é o resultado deste modelo:

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	396.6%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	15.17%
Vol 252d	26.18%
Índice Sharpe	0.22
VAR diário 95%	-2.47%
Turn over carteira	22.99%
Drawdown máximo	-56.42%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	56.2%
% Operações perdedoras	43.8%
Média de ganhos	7.01%
Média de perdas	-5.4%
Expec. matemática por trade	1.58%
Maior sequência de vitória	12
Maior sequência de derrotas	9
% Meses carteira > IBOV	54.35%

Mundo 23

23.1. – Backtest ROE e ROIC

Nesse backtest, será testado o modelo utilizando o indicador ROE e ROIC, individualmente e ao mesmo tempo.

Quanto menos empresas, melhor será o desempenho do seu modelo, afinal, estará pegando somente as melhores empresas para seu modelo. Contudo, conforme explicado no trade-off, a liquidez será estrangulada, tendo que aumentar.

Por exemplo, ao comparar 2 modelos:

- 10 ativos e 1 milhão de liquidez
- 5 ativos e 2 milhões de liquidez

O segundo modelo apresentará um desempenho melhor, apesar de ambos possuírem a mesma capacity.

Esse é o resultado do ROE com 10 ativos e 1 milhão de liquidez:

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	104.42%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	6.5%
Vol 252d	18.34%
Índice Sharpe	-0.11
VAR diário 95%	-2.08%
Turn over carteira	9.2%
Drawdown máximo	-56.71%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	60.58%
% Operações perdedoras	39.42%
Média de ganhos	4.34%
Média de perdas	-4.71%
Expec. matemática por trade	0.77%
Maior sequência de vitória	7
Maior sequência de derrotas	4
% Meses carteira > IBOV	50.0%

Esse é o resultado do ROE com 5 ativos e 2 milhões de liquidez:

código.py

**Estatísticas de Retorno e Risco**

Retorno acum. modelo	129.74%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	7.6%
Vol 252d	21.77%
Índice Sharpe	-0.06
VAR diário 95%	-2.33%
Turn over carteira	9.34%
Drawdown máximo	-56.64%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	59.12%
% Operações perdedoras	40.88%
Média de ganhos	5.18%
Média de perdas	-5.26%
Expec. matemática por trade	0.91%
Maior sequência de vitória	6
Maior sequência de derrotas	4
% Meses carteira > IBOV	53.62%

Esse é o resultado do ROIC com 10 ativos e 1 milhão de liquidez:

código.py

**Estatísticas de Retorno e Risco**

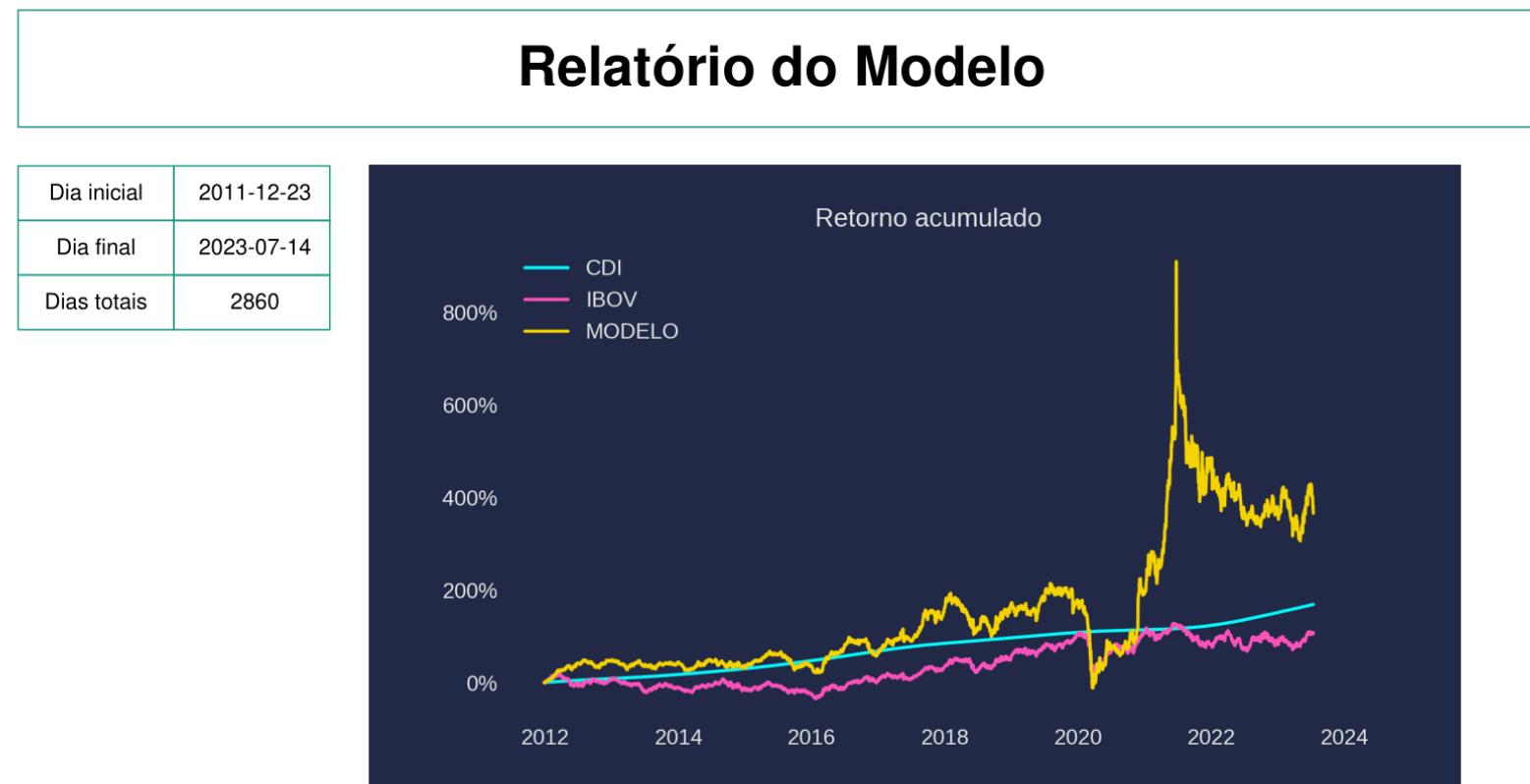
Retorno acum. modelo	285.08%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	12.61%
Vol 252d	22.35%
Índice Sharpe	0.13
VAR diário 95%	-2.3%
Turn over carteira	10.88%
Drawdown máximo	-57.33%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	58.39%
% Operações perdedoras	41.61%
Média de ganhos	5.75%
Média de perdas	-4.91%
Expec. matemática por trade	1.31%
Maior sequência de vitória	5
Maior sequência de derrotas	4
% Meses carteira > IBOV	55.07%

Esse é o resultado do ROIC com 5 ativos e 2 milhões de liquidez:

código.py

**Estatísticas de Retorno e Risco**

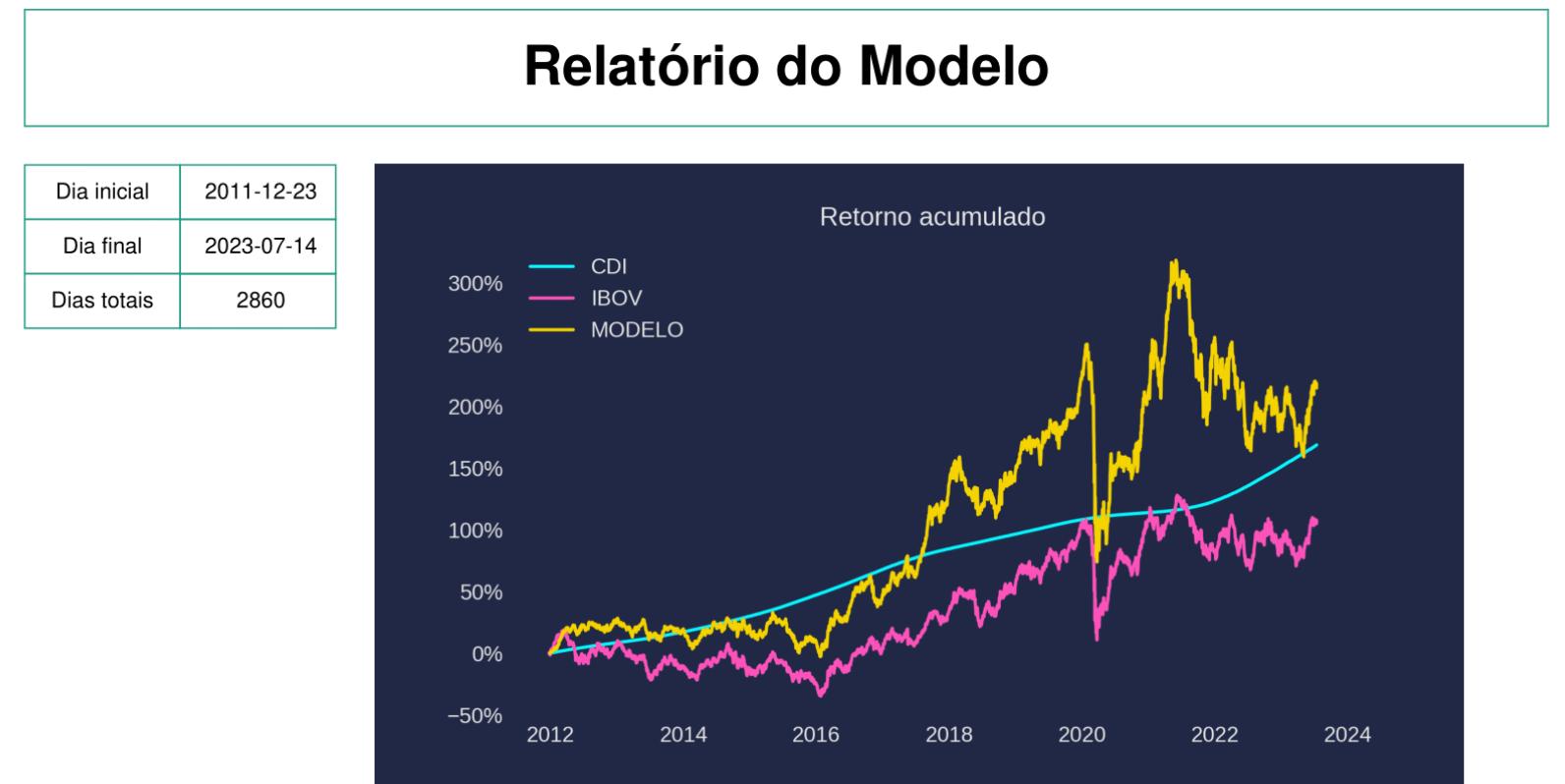
Retorno acum. modelo	365.37%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	14.51%
Vol 252d	25.95%
Índice Sharpe	0.16
VAR diário 95%	-2.71%
Turn over carteira	12.26%
Drawdown máximo	-71.9%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	55.47%
% Operações perdedoras	44.53%
Média de ganhos	7.31%
Média de perdas	-5.39%
Expec. matemática por trade	1.65%
Maior sequência de vitória	6
Maior sequência de derrotas	5
% Meses carteira > IBOV	51.45%

Esse é o resultado do ROE e ROIC:

código.py

**Estatísticas de Retorno e Risco**

Retorno acum. modelo	214.84%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	10.63%
Vol 252d	20.69%
Índice Sharpe	0.07
VAR diário 95%	-2.11%
Turn over carteira	10.09%
Drawdown máximo	-50.34%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	61.31%
% Operações perdedoras	38.69%
Média de ganhos	4.67%
Média de perdas	-4.63%
Expec. matemática por trade	1.07%
Maior sequência de vitória	6
Maior sequência de derrotas	5
% Meses carteira > IBOV	53.62%

Mundo 24

24.1. – Backtest Magic Formula

Nesse backtest, serão testadas 2 versões da Magic Formula de Joel Greenblatt. A primeira será apenas o ROIC com EV/EBIT.

A segunda versão é um pouco diferente, contendo 2 carteiras com apenas 6 ativos cada.

Carteira 1: EV/EBIT com peso de 0.5.

Carteira 2: ROIC com peso de 0.5.

Esse é o resultado do ROIC com EV/EBIT:

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	434.78%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	15.92%
Vol 252d	21.98%
Índice Sharpe	0.26
VAR diário 95%	-2.37%
Turn over carteira	18.36%
Drawdown máximo	-54.84%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	59.85%
% Operações perdedoras	40.15%
Média de ganhos	6.1%
Média de perdas	-5.21%
Expec. matemática por trade	1.56%
Maior sequência de vitória	15
Maior sequência de derrotas	6
% Meses carteira > IBOV	52.9%

Esse é o resultado da segunda versão, com 2 carteiras separadas:

código.py



Mundo 25

25.1. – Backtest Momentum 1, 6 e 12 meses

Nesse backtest, será testado o modelo utilizando o momentum com 1, 6 e 12 meses.

Esse é o resultado do Momentum 1 mês:

código.py

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

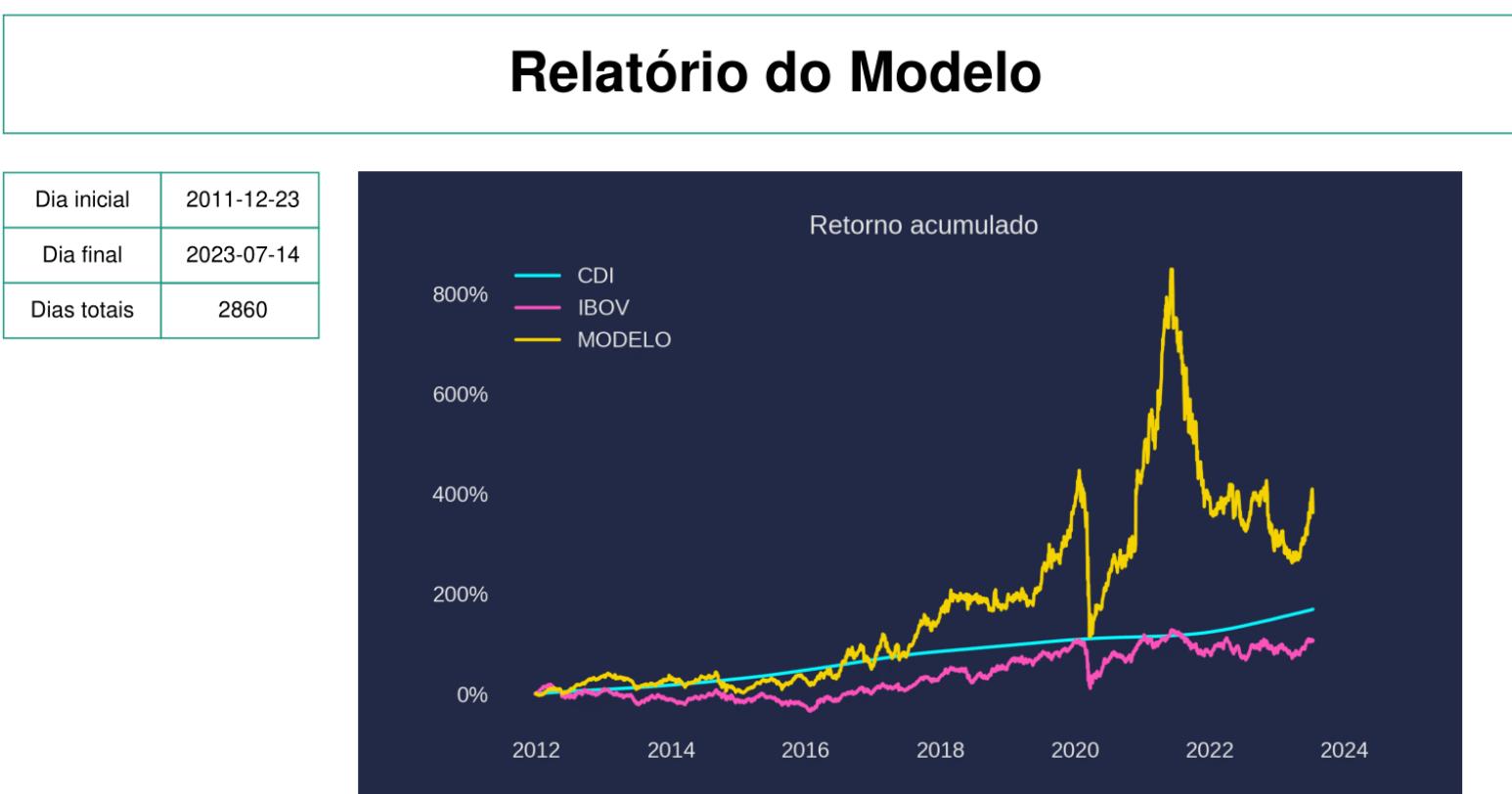
Retorno acum. modelo	197.08%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	10.07%
Vol 252d	34.25%
Índice Sharpe	0.03
VAR diário 95%	-2.74%
Turn over carteira	89.64%
Drawdown máximo	-75.18%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	48.18%
% Operações perdedoras	51.82%
Média de ganhos	9.37%
Média de perdas	-6.19%
Expec. matemática por trade	1.31%
Maior sequência de vitória	8
Maior sequência de derrotas	10
% Meses carteira > IBOV	49.28%

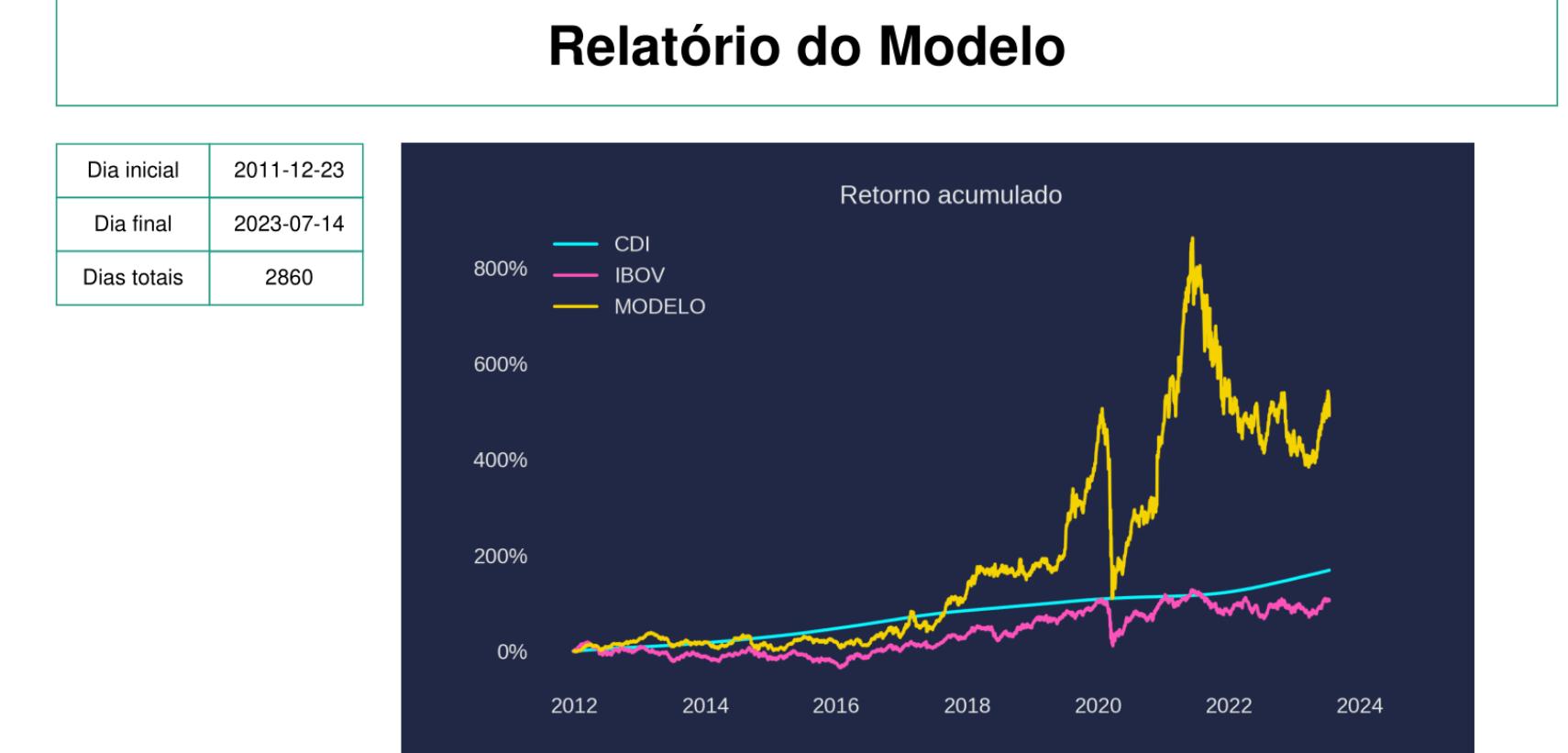
Esse é o resultado do Momentum 6 meses:

código.py



Esse é o resultado do Momentum 12 meses:

código.py



Estatísticas de Retorno e Risco

Retorno acum. modelo	361.77%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	14.43%
Vol 252d	26.83%
Índice Sharpe	0.18
VAR diário 95%	-2.55%
Turn over carteira	43.8%
Drawdown máximo	-61.88%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	59.12%
% Operações perdedoras	40.88%
Média de ganhos	7.13%
Média de perdas	-6.5%
Expec. matemática por trade	1.56%
Maior sequência de vitória	9
Maior sequência de derrotas	8
% Meses carteira > IBOV	56.52%

Estatísticas de Retorno e Risco

Retorno acum. modelo	490.99%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	16.95%
Vol 252d	23.47%
Índice Sharpe	0.27
VAR diário 95%	-2.33%
Turn over carteira	32.55%
Drawdown máximo	-65.32%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	60.58%
% Operações perdedoras	39.42%
Média de ganhos	6.67%
Média de perdas	-5.86%
Expec. matemática por trade	1.73%
Maior sequência de vitória	8
Maior sequência de derrotas	7
% Meses carteira > IBOV	57.97%

Mundo 26

26.1. – Backtest EV/EBIT e Valor de Mercado

Nesse backtest, será testado o modelo utilizando o EV/EBIT com Valor de Mercado, captando small caps de valor.

Esse é o resultado deste modelo:

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	1463.81%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	27.42%
Vol 252d	31.32%
Índice Sharpe	0.61
VAR diário 95%	-2.67%
Turn over carteira	28.04%
Drawdown máximo	-60.09%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	58.39%
% Operações perdedoras	41.61%
Média de ganhos	8.66%
Média de perdas	-6.0%
Expec. matemática por trade	2.56%
Maior sequência de vitória	9
Maior sequência de derrotas	8
% Meses carteira > IBOV	58.7%

Mundo 27

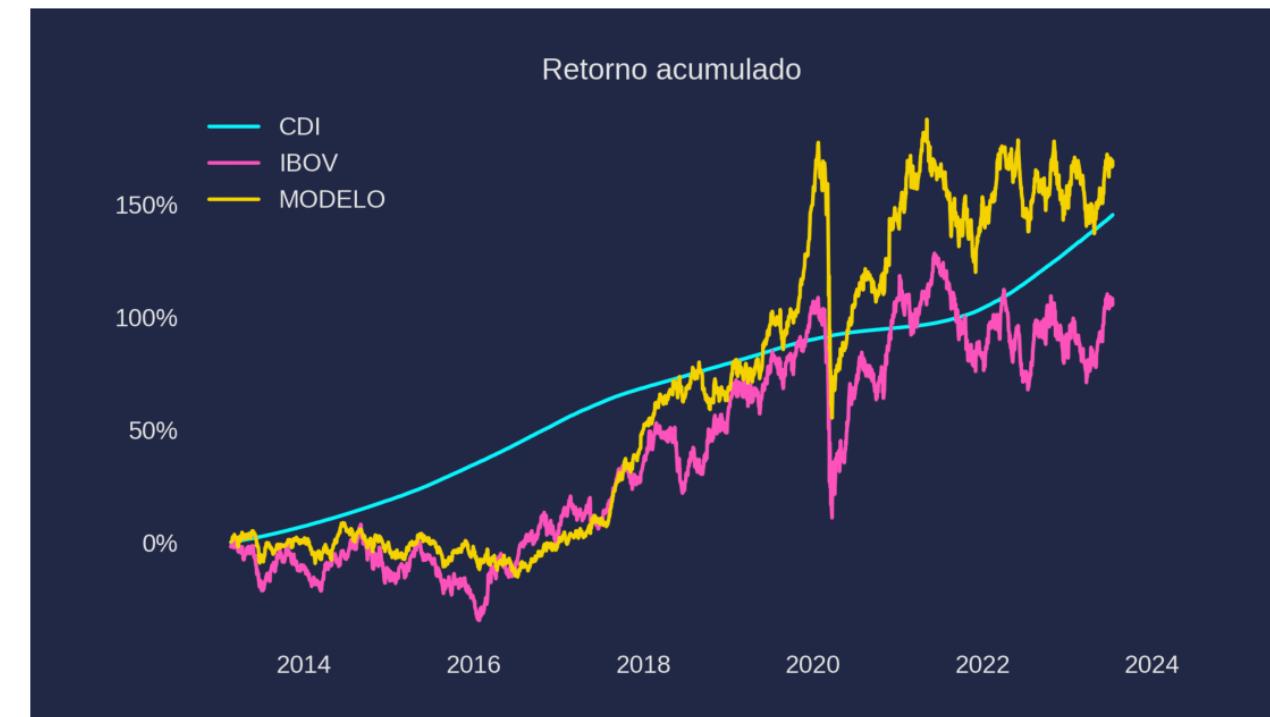
27.1. – Backtest anomalia do Low Beta

Nesse backtest, será testado o modelo utilizando a anomalia do Low Beta, que é empresas com Beta baixo.

Esse é o resultado do backtest da anomalia do Low Beta.

Relatório do Modelo

Dia inicial	2013-02-20
Dia final	2023-07-14
Dias totais	2577



Estatísticas de Retorno e Risco

Retorno acum. modelo	166.94%
Retorno acum. CDI	145.2%
Retorno acum. IBOV	105.38%
Retorno a.a. modelo	10.08%
Vol 252d	16.44%
Índice Sharpe	0.05
VAR diário 95%	-1.54%
Turn over carteira	12.85%
Drawdown máximo	-44.04%

Estatísticas de Trade

Número de carteiras	123
% Operações vencedoras	58.54%
% Operações perdedoras	41.46%
Média de ganhos	4.41%
Média de perdas	-3.94%
Expec. matemática por trade	0.95%
Maior sequência de vitória	10
Maior sequência de derrotas	4
% Meses carteira > IBOV	53.23%

Mundo 28

28.1. – Backtest com Fator Volatilidade

Como controlar o risco-retorno da carteira de Small-Caps? Será utilizado a calculadora de Factor Investing para ajudar a melhor o risco-retorno da carteira através do Fator Volatilidade.

Nesse backtest, será testado o modelo utilizando o Fator Volatilidade para controlar o risco da carteira.

Esse é o resultado do backtest:

código.py

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	362.97%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	14.46%
Vol 252d	21.82%
Índice Sharpe	0.27
VAR diário 95%	-1.83%
Turn over carteira	25.0%
Drawdown máximo	-45.44%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	55.47%
% Operações perdedoras	44.53%
Média de ganhos	5.36%
Média de perdas	-3.7%
Expec. matemática por trade	1.33%
Maior sequência de vitória	6
Maior sequência de derrotas	6
% Meses carteira > IBOV	57.97%

Mundo 29

29.1. – Backtest de análise técnica com Fac-

Nesse backtest, será testado o modelo utilizando o Fator Investing juntamente com a análise técnica, utilizando o momentum 7 com 40.

Esse é o resultado do backtest:

Relatório do Modelo

Dia inicial	2011-12-23
Dia final	2023-07-14
Dias totais	2860



Estatísticas de Retorno e Risco

Retorno acum. modelo	121.2%
Retorno acum. CDI	168.8%
Retorno acum. IBOV	105.26%
Retorno a.a. modelo	7.25%
Vol 252d	31.97%
Índice Sharpe	-0.06
VAR diário 95%	-2.63%
Turn over carteira	82.92%
Drawdown máximo	-74.24%

Estatísticas de Trade

Número de carteiras	137
% Operações vencedoras	54.74%
% Operações perdedoras	45.26%
Média de ganhos	7.39%
Média de perdas	-6.54%
Expec. matemática por trade	1.08%
Maior sequência de vitória	9
Maior sequência de derrotas	9
% Meses carteira > IBOV	46.38%

Está entregue todo o ouro de Factor Investing, agora a ferramenta de backtest de Factor Investing é sua, totalmente personalizável. Seja um alquimista e explore ao máximo a combinação de vários fatores.

Realize inúmeros testes, mas com responsabilidade. E, se possível, sempre me refcrcie e cite como o mentor, pois isso me auxilia bastante.

Façam testes vencedores!

