MVP SPRINT 3 - ENGENHARIA DE DADOS
ALUNO: IGOR MIRANDA EISENLOHR

Descrição

O trabalho fará todo o processo de ETL:

1. a busca por dados do mercado financeiro utilizando a linguagem Python.
2. Posteriormente, deve-se levar os dados para um ambiente Cloud (escolhido: Google Cloud);
3. Realizar as transformações necessárias dos dados;
4. Por fim, realizar análises que gerem valor aos dados coletados.

Objetivo

O objetivo é consguir coletar os dados de um importante indice economico, como o CDI, e dados dos principais ativos presentes na bolsa brasileira. Após esse processo, temos interesse de armazenar os dados em um Data Warehouse/Data Lake (como o Google BigQuery).

Com uma tabela referente ao cadastro de ativos, outra referente a informações extras de cadastro e uma última contendo os preços históricos de 2 anos para os ativos encontrados.

A ideia é remover possíveis cadastros duplicados, assim como, preços ausentes,

Detalhamento

1. Busca pelos dados

A busca dos dados foi feita com as bibliotecas **yfinance** e **investpy** e leitura de json de url fornecida pelo banco central do brasil. A escolha dos dados teve como motivo principal a familiaridade com o tema, visto que tenho atuado em uma empresa focada no mercado financeiro.

2. Coleta

A coleta dos dados foi feita por meio de script python, buscando lista de ativos cadastrados na bolsa brasileira, informações adicionais sobre os ativos e preços históricos dos últimos 2 anos.

3. Modelagem

O banco de dados possui 2 tabelas, todas relacionadas pelo “ticker” do ativo. Exemplo: “PETR4.SA” para identificação da empresa Petrobras.

A primeira tabela armazena o cadastro dos ativos com as colunas:

Link para visualização dos dados: cadastro

Coluna	**Tipo**	**Descrição**
country	STRING	País de negociação do ticket (“brazil”)
name	STRING	Nome da companhia
full_name	STRING	Nome completo da companhia
isin	STRING	Código isin (cadastro na b3)
currency	STRING	Moeda
ticker	STRING	Simbolo/Ticker da empresa na bolsa
city	STRING	Cidade de localização da empresa
state	STRING	Estado de localização da empresa
industry	STRING	Indústria de atuação da empresa
sector	STRING	Setor de atuação da empresa

A segunda tabela armazena o preço histórico dos ativos com as colunas:

Link para visualização de parte dos dados: Amostra de prices

Coluna	**Tipo**	**Descrição**
date	STRING	Data da cotação
year	INTEGER	Ano da cotação
month	INTEGER	Mês da cotação

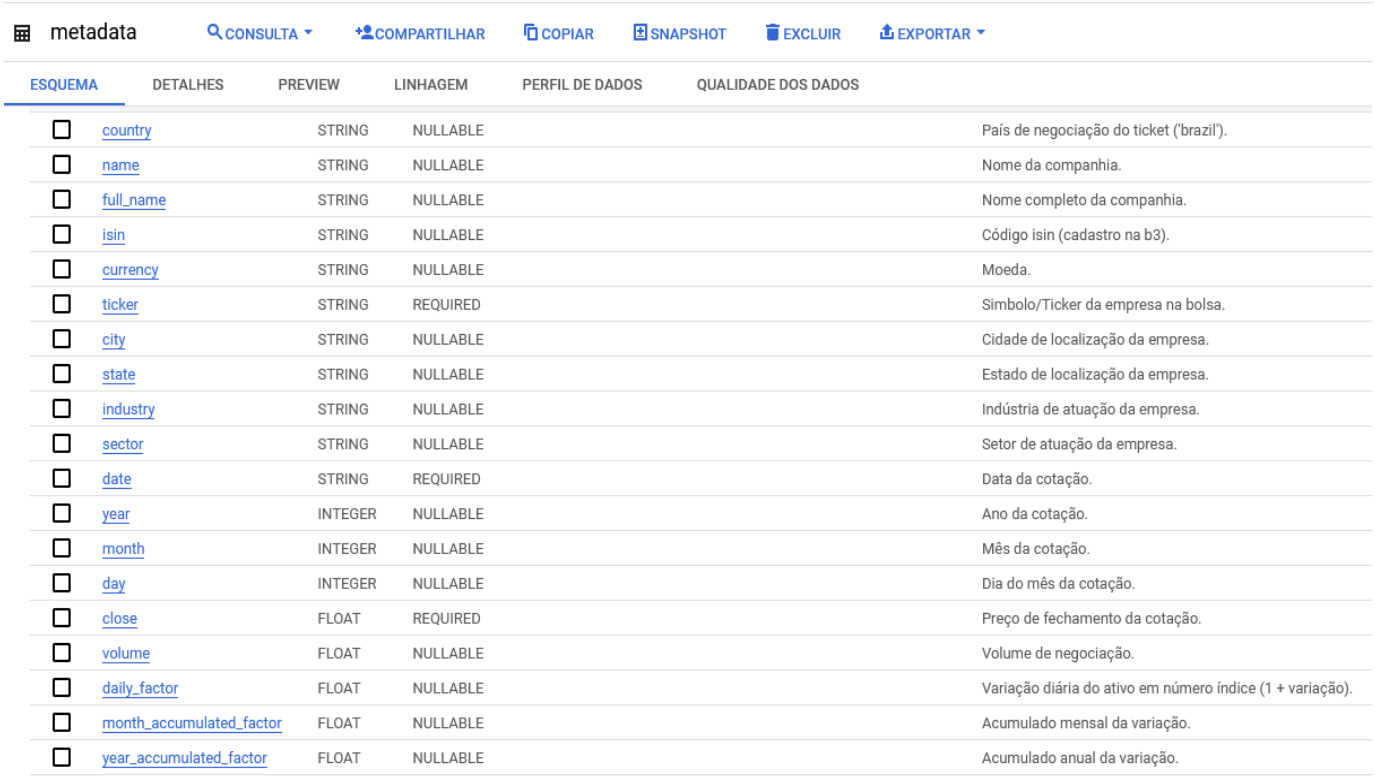
day	INTEGER	Dia do mês da cotação
ticker	STRING	Simbolo/Ticker da empresa na bolsa
close	FLOAT	Preço de fechamento da cotação
volume	FLOAT	Volume de negociação
daily_factor	FLOAT	Variação diária do ativo em número índice (1 + variação)
month_accumulated_factor	FLOAT	Acumulado mensal da variação
year_accumulated_factor	FLOAT	Acumulado anual da variação

Por fim, criou-se a tabela de metadados como um dicionário das colunas do esquema de dados.

```
from google.cloud import bigquery

from google.oauth2 import service_account



class MetadataTableCreator:

    def __init__(self, project_id, credentials_path):

        self.project_id = project_id # Nome do projeto

                                                self.credentials      =
service_account.Credentials.from_service_account_file(credentials_path)   #
Credenciais do Google Cloud

                    self.client   =   bigquery.Client(project=project_id,
credentials=self.credentials) # Cliente do BigQuery



    def create_metadata_table(self, dataset_name, table_name):

        schema = [

            # Metadados da empresa
```

```
                bigquery.SchemaField("country", "STRING", mode="NULLABLE",
description="País de negociação do ticket ('brazil')."),

                    bigquery.SchemaField("name", "STRING", mode="NULLABLE",
description="Nome da companhia."),

                bigquery.SchemaField("full_name", "STRING", mode="NULLABLE",
description="Nome completo da companhia."),

                    bigquery.SchemaField("isin", "STRING", mode="NULLABLE",
description="Código isin (cadastro na b3)."),

                bigquery.SchemaField("currency", "STRING", mode="NULLABLE",
description="Moeda."),

                    bigquery.SchemaField("ticker", "STRING", mode="REQUIRED",
description="Simbolo/Ticker da empresa na bolsa."),

                    bigquery.SchemaField("city", "STRING", mode="NULLABLE",
description="Cidade de localização da empresa."),

                    bigquery.SchemaField("state", "STRING", mode="NULLABLE",
description="Estado de localização da empresa."),

                bigquery.SchemaField("industry", "STRING", mode="NULLABLE",
description="Indústria de atuação da empresa."),

                    bigquery.SchemaField("sector", "STRING", mode="NULLABLE",
description="Setor de atuação da empresa."),



            # Dados da cotação

                    bigquery.SchemaField("date", "STRING", mode="REQUIRED",
description="Data da cotação."),

                    bigquery.SchemaField("year", "INTEGER", mode="NULLABLE",
description="Ano da cotação."),

                    bigquery.SchemaField("month", "INTEGER", mode="NULLABLE",
description="Mês da cotação."),
```

```
                bigquery.SchemaField("day", "INTEGER", mode="NULLABLE",
description="Dia do mês da cotação."),

                bigquery.SchemaField("close", "FLOAT", mode="REQUIRED",
description="Preço de fechamento da cotação."),

                bigquery.SchemaField("volume", "FLOAT", mode="NULLABLE",
description="Volume de negociação."),

                        bigquery.SchemaField("daily_factor", "FLOAT64",
mode="NULLABLE", description="Variação diária do ativo em número índice (1
+ variação)."),

              bigquery.SchemaField("month_accumulated_factor", "FLOAT64",
mode="NULLABLE", description="Acumulado mensal da variação."),

               bigquery.SchemaField("year_accumulated_factor", "FLOAT64",
mode="NULLABLE", description="Acumulado anual da variação.")

        ]




                                                        table       =
bigquery.Table(f"{self.project_id}.{dataset_name}.{table_name}",
schema=schema) # Criacao da tabela

          table = self.client.create_table(table) # Criacao da tabela no
BigQuery

         print(f"Table {table_name} created in dataset {dataset_name}.") #
Mensagem de sucesso



if __name__ == "__main__":

            creator   =   MetadataTableCreator('bigquery-sandbox-385813',
"./sprint3-storage.json") # Nome do projeto e credenciais

    creator.create_metadata_table('sprint3', "metadata") # Nome do dataset
e nome da tabela
```



metadata

CONSULTA | COMPARTILHAR | COPIAR | SNAPSHOT | EXCLUIR | EXPORTAR

ESQUEMA | DETALHES | PREVIEW | LINHAGEM | PERFIL DE DADOS | QUALIDADE DOS DADOS

country	STRING	NULLABLE		País de negociação do ticket ('brazil').
name	STRING	NULLABLE		Nome da companhia.
full_name	STRING	NULLABLE		Nome completo da companhia.
isin	STRING	NULLABLE		Código isin (cadastro na b3).
currency	STRING	NULLABLE		Moeda.
ticker	STRING	REQUIRED		Simbolo/Ticker da empresa na bolsa.
city	STRING	NULLABLE		Cidade de localização da empresa.
state	STRING	NULLABLE		Estado de localização da empresa.
industry	STRING	NULLABLE		Indústria de atuação da empresa.
sector	STRING	NULLABLE		Setor de atuação da empresa.
date	STRING	REQUIRED		Data da cotação.
year	INTEGER	NULLABLE		Ano da cotação.
month	INTEGER	NULLABLE		Mês da cotação.
day	INTEGER	NULLABLE		Dia do mês da cotação.
close	FLOAT	REQUIRED		Preço de fechamento da cotação.
volume	FLOAT	NULLABLE		Volume de negociação.
daily_factor	FLOAT	NULLABLE		Variação diária do ativo em número índice (1 + variação).
month_accumulated_factor	FLOAT	NULLABLE		Acumulado mensal da variação.
year_accumulated_factor	FLOAT	NULLABLE		Acumulado anual da variação.

4. Carga

A carga será feita após a transformação dos dados no ambiente Cloud do GoogleBigQuery. A carga foi realizada no fim do script Python, utilizando a biblioteca **pandas-gbq** que faz a carga de dataframes Pandas para o Google BigQuery.

5. Análise

Análises que gostaria de fazer:

- Qual estado possui maior número de empresas listadas?
- Quais setores apresentam maior número de empresas listadas?
- Quais indústrias apresentam maior número de empresas listadas?
- Qual ação apresentou maior variação no ano de 2022?
- Qual ação apresentou maior rentabilidade no ano de 2022?
- Qual ação apresentou menor rentabilidade no ano de 2022?
- Quais ações apresentaram maior volume de negociações?
- Quais ações apresentaram menor volume de negociações?

Entrega

A entrega do trabalho está no meu github, no seguinte repositório:
https://github.com/IgorEisenlohr/sprint3-engenharia-dados

Autoavaliação

Acredito ter cumprido os requisitos do trabalho, realizando todo o processo de ETL de dados para um ambiente cloud. Consegui extrair dados de diversas fontes de dados, conseguindo a parte de cadastros e de historico de preços de ações e CDI. Consegui fazer o upload de todos os arquivos necessários para o Google Cloud Storage, que permitiria a utilização do Data Fusion para um ETL *no-code.* Porém, para criação de colunas com cálculos um pouco mais complexos e algumas outras transformações que seriam mais fáceis utilizando linguagem de programação, permaneci na utilização do Python para transformação dos dados e para realizar a parte de carregamento no Google BigQuery.

Acredito ter apresentado boas análises utilizando SQL e gráficos com python. Porém, é possível realizar algumas melhoras na base de dados, para trazer eventos corporativos de split, inplit e dividendos, que alteram os valores das cotações das ações.

Código

Importando bibliotecas importantes

```
# Description: Script para coleta, transformacao e carregamento de dados
# Author: Igor Miranda Eisenlohr

# Importacao das bibliotecas
import pandas as pd
import pandas_gbq
import numpy as np
import investpy
import yfinance as yf
import os
from google.cloud import storage
from google.oauth2 import service_account
```

INICIANDO PROCESSO DE EXTRAÇÃO (E)

Extração do CDI por *json* disponibilizado pelo BCB

```
    def get_cdi(self):
```

```
        url_cdi  =
"https://api.bcb.gov.br/dados/serie/bcdata.sgs.{}/dados?formato=json".form
at(12) # Codigo do CDI
        cdi = pd.read_json(url_cdi) # Leitura do JSON
        cdi['data'] = pd.to_datetime(cdi['data'], format='%d/%m/%Y')
        cdi['ano'] = cdi['data'].dt.year # Criacao da coluna ano
        cdi.rename(columns={'data': 'date'}, inplace=True) # Renomeacao da
coluna data para Date
        cdi['ticker'] = 'CDI' # Criacao da coluna Ticker
        cdi['daily_variation'] = cdi['valor'] # Criacao da coluna Close
        cdi = cdi.query('ano >= 2022') # Filtragem dos dados a partir de
2021
        cdi_df = cdi[['date', 'ticker','daily_variation']] # Selecao das
colunas

        return cdi_df # Retorno do dataframe
```

Extração dos ativos cadastrados na bolsa:

```
    def get_stocks(self):
        acoes_df = investpy.stocks.get_stocks(country='brazil') # Coleta
de dados de acoes brasileiras
        acoes_df['symbol'] = acoes_df['symbol'].apply(lambda x: x+'.SA') #
Adicao do sufixo .SA para acoes brasileiras
        acoes_df.rename(columns={'symbol': 'ticker'}, inplace=True) #
Renomeacao da coluna symbol para Ticker
        return acoes_df # Retorno do dataframe
```

Extração de informações adicionais dos ativos da bolsa

```
    def get_stocks_info(self, acoes_df):
        tickers = acoes_df['ticker'] # Lista de tickers
        tickers_info_list = [] # Lista vazia para armazenar os dados
        for ticker in tickers: # Loop para coletar os dados de cada ticker
            try:
                tickers_info_list.append(yf.Ticker(ticker).info ) # Coleta
dos dados
                tickers_info_df = pd.DataFrame(tickers_info_list) #
Transformacao da lista em dataframe
            except:
                print(f"Ticker não possui informações disponiveis:
{ticker}") # Caso o ticker nao exista, printa o ticker
```

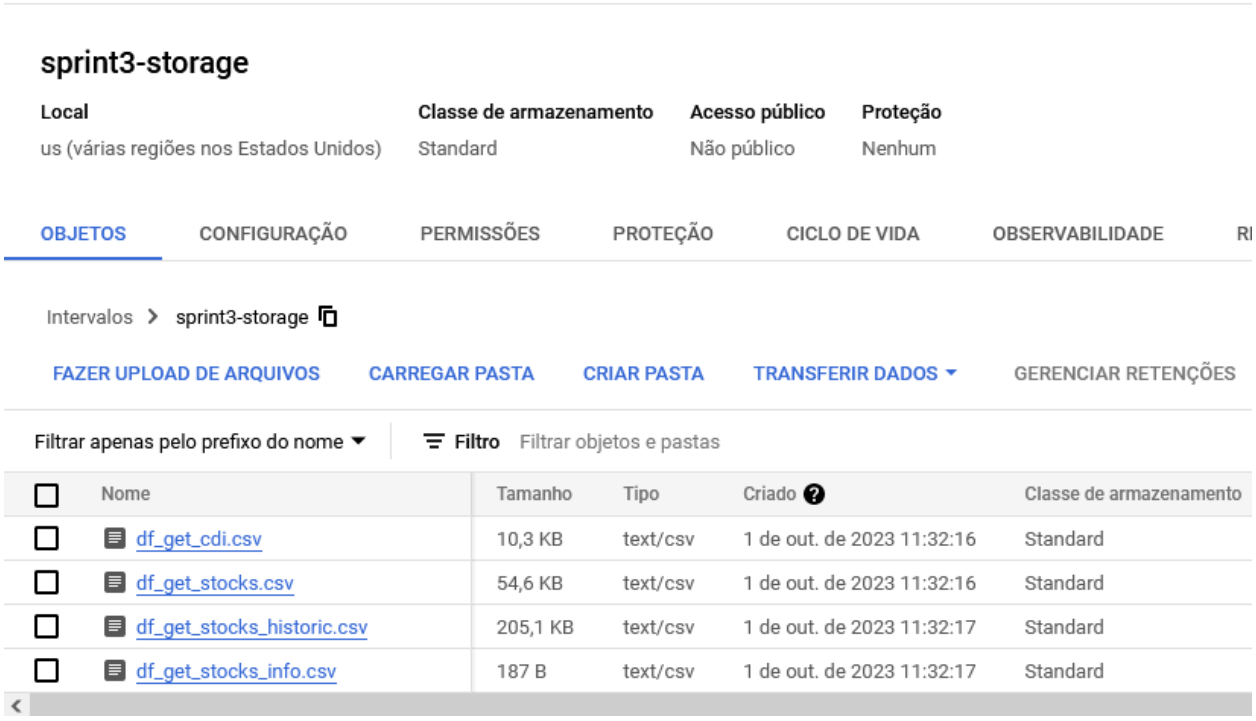
```
        tickers_info_df.rename(columns={'symbol': 'ticker'}, inplace=True)
# Renomeacao da coluna symbol para Ticker
        return tickers_info_df[['city', 'state', 'country', 'industry',
'sector', 'ticker']] # Retorno do dataframe
```

Extração dos preços históricos das ações encontradas de 2022 pra frente

```
    def get_stocks_historic(self, tickers):
        historicos_list = []  # Lista vazia para armazenar os dados
        for ticker in tickers:  # Loop para coletar os dados de cada
ticker
            historico = yf.download(ticker, start="2022-01-03")  # Coleta
dos dados
            historico['Ticker'] = ticker  # Adicao da coluna Ticker
            historicos_list.append(historico)  # Adicao do dataframe na
lista
        historicos_df = pd.concat(historicos_list) # Transformacao da
lista em dataframe
        historicos_df.reset_index(inplace=True) # Reset do index
        historicos_df.rename(columns=str.lower, inplace=True) # Renomeacao
das colunas
        return historicos_df[['date', 'ticker','close', 'volume']]  #
Retorno do dataframe
```

Carregando os dataframes gerados no Google Storage

```
    def to_google_storage(self, *dataframes, func_names):
        for df, func_name in zip(dataframes, func_names):  # Loop para
salvar os dataframes
            file_name = f'df_{func_name}.csv'  # Nome do arquivo
            df.to_csv(file_name, index=False)  # Salvando o dataframe
            blob = self.bucket.blob(file_name)  # Blob
            blob.upload_from_filename(file_name)  # Upload do arquivo no
bucket
```



INICIANDO PARTE DE TRANSFORMAÇÃO (T)

Por possuir maior dominio com a linguagem python, além de fornecer uma infinidade de funções de transforamação de dados do que o Data Fusion, optou-se por manter o ETL com o próprio script. Então, apesar de ter os arquivos .csv necessários no Google Storage, vamos continuar o processo de transformação com python e posteriormente realizar o Load para o bigquery pelo script. Realizando assim, todo o processo de ETL necessário para extrair, transformar e carregar dados do mercado financeiro.

```
    def transform(self, cdi_file, stocks_file, stocks_info_file,
stocks_historic_file):
        print("TRANSFORMING DATA...")
        cdi_df = pd.read_csv(cdi_file)
        cadastro_df = pd.read_csv(stocks_file)
        info_df = pd.read_csv(stocks_info_file)
        historic_df = pd.read_csv(stocks_historic_file)

        duplicate_symbols =
cadastro_df[cadastro_df.duplicated(subset='ticker', keep=False)] #
Verificacao de duplicatas
        if not duplicate_symbols.empty: # Caso existam duplicatas
```

```
            print(f'Duplicatas encontradas no cadastro:
{duplicate_symbols["ticker"].tolist()}') # Printa os tickers duplicados
        cadastro_cleaned_df = cadastro_df.drop_duplicates(subset='ticker',
keep='first') # Remocao das duplicatas

        cadastro_df = pd.merge(cadastro_cleaned_df, info_df, on='ticker',
how='left') # Merge dos dataframes de cadastro e informações
        cadastro_df_cleaned =
cadastro_df.drop(columns='country_y').rename(columns={'country_x':
'country'}) # Renomeacao da coluna country

        cdi_df['date'] =
pd.to_datetime(cdi_df['date']).dt.strftime('%Y-%m-%d')

        prices_df = pd.concat([historic_df, cdi_df], ignore_index=True) #
Uniao dos dataframes de historico de stocks e CDI
        duplicate_rows = prices_df[prices_df.duplicated(subset=['date',
'ticker'], keep=False)]
        if not duplicate_rows.empty:
            print(f'Duplicatas encontradas na tabela de preços:
{duplicate_rows[["date", "ticker"]].to_dict(orient="records")}') # Printa
as duplicatas
        prices_df_cleaned = prices_df.drop_duplicates(subset=['date',
'ticker'], keep='first') # Remocao das duplicatas

        prices_df_cleaned = prices_df_cleaned.sort_values(by=['ticker',
'date'], ascending=True)  # Ordenacao do dataframe
        prices_df_cleaned['prev_close'] =
prices_df_cleaned.groupby('ticker')['close'].shift(1) # Criacao da coluna
Prev_Close
        prices_df_cleaned['daily_factor'] = (prices_df_cleaned['close'] /
prices_df_cleaned['prev_close']).fillna(1) # Criacao da coluna
Daily_Factor

        prices_df_cleaned['date'] =
pd.to_datetime(prices_df_cleaned['date']) # Conversao da coluna Date para
datetime
        prices_df_cleaned['year'] = prices_df_cleaned['date'].dt.year #
Criacao da coluna Year
```

```
        prices_df_cleaned['month'] = prices_df_cleaned['date'].dt.month #
Criacao da coluna Month
        prices_df_cleaned['day'] = prices_df_cleaned['date'].dt.day #
Criacao da coluna Day

        is_not_cdi = prices_df_cleaned['ticker'] != 'CDI' # Filtro para
selecionar apenas os tickers que nao sao CDI
        prices_df_cleaned.loc[is_not_cdi, 'daily_variation'] =
prices_df_cleaned.loc[is_not_cdi].groupby('ticker')['close'].transform(lam
bda x: x / x.shift(1) - 1) # Calculando a variacao diaria

        prices_df_cleaned['month_accumulated_variation'] =
prices_df_cleaned.groupby(['ticker', 'year',
'month'])['daily_variation'].transform(lambda x: (x + 1).cumprod() - 1) *
100 # Calculando a variação acumulada mensal
        prices_df_cleaned['year_accumulated_variation'] =
prices_df_cleaned.groupby(['ticker',
'year'])['daily_variation'].transform(lambda x: (x + 1).cumprod() - 1) *
100 # Calculando a variação acumulada anual

        prices_df_cleaned['date'] = prices_df_cleaned['date'].astype(str)
# Conversao da coluna Date para string

        print("DATA TRANSFORMED!")

        return cadastro_df_cleaned, prices_df_cleaned
```

Alguns pontos realizados na transformação de dados:

- Remoção de cadastros duplicados
- Junção do dataframe com o cadastro de ativos e informações extras (JOIN)
- União entre preços do CDI e preços de stocks (UNION ALL)
- Extração de Ano, mês e dia do atributo de data
- Cálculo da variação dos ativos de maneira diária
- Cálculo da rentabilidade mensal dos ativos
- Cálculo da rentabilidade anual dos ativos

INICIANDO PARTE DE CARREGAMENTO (L)

```
    def load(self, cadastro_info_df, prices_df, dataset_name,
cadastro_table_name, prices_table_name):
```

```
        print("LOADING DATA...")

        credentials =
service_account.Credentials.from_service_account_file(
            'sprint3-storage.json'
        ) # Credenciais do Google Cloud

        cadastro_schema = [
            bigquery.SchemaField("country", "STRING", mode="NULLABLE"  ),
            bigquery.SchemaField("name", "STRING", mode="NULLABLE"     ),
            bigquery.SchemaField("full_name", "STRING", mode="NULLABLE"),
            bigquery.SchemaField("isin", "STRING", mode="NULLABLE"     ),
            bigquery.SchemaField("currency", "STRING", mode="NULLABLE" ),
            bigquery.SchemaField("ticker", "STRING", mode="REQUIRED"   ),
            bigquery.SchemaField("city", "STRING", mode="NULLABLE"     ),
            bigquery.SchemaField("state", "STRING", mode="NULLABLE"    ),
            bigquery.SchemaField("industry", "STRING", mode="NULLABLE" ),
            bigquery.SchemaField("sector", "STRING", mode="NULLABLE"   )
        ]

        # Conversão para lista de dicionários
        cadastro_schema_dicts = [{'name': field.name, 'type':
field.field_type, 'mode': field.mode} for field in cadastro_schema]

        # Carregar o dataframe cadastro_info_df no BigQuery
        pandas_gbq.to_gbq(
            cadastro_info_df,
            f"{dataset_name}.{cadastro_table_name}",
            project_id='bigquery-sandbox-385813',
            if_exists='replace',
            table_schema=cadastro_schema_dicts,
            credentials=credentials
        )

        prices_schema = [
            bigquery.SchemaField("date", "STRING", mode="REQUIRED" ),
            bigquery.SchemaField("year", "INTEGER", mode="NULLABLE"),
            bigquery.SchemaField("month", "INTEGER", mode="NULLABLE"),
            bigquery.SchemaField("day", "INTEGER", mode="NULLABLE" ),
            bigquery.SchemaField("ticker", "STRING", mode="REQUIRED"),
```

```
            bigquery.SchemaField("close", "FLOAT", mode="REQUIRED"),
            bigquery.SchemaField("volume", "FLOAT", mode="NULLABLE"),
            bigquery.SchemaField("daily_factor", "FLOAT64",
mode="NULLABLE"),
            bigquery.SchemaField("month_accumulated_factor", "FLOAT64",
mode="NULLABLE"),
            bigquery.SchemaField("year_accumulated_factor", "FLOAT64",
mode="NULLABLE")
        ]

        # Conversão para lista de dicionários
        prices_schema_dicts = [{'name': field.name, 'type':
field.field_type, 'mode': field.mode} for field in prices_schema]

        # Carregar o dataframe prices_df no BigQuery
        pandas_gbq.to_gbq(
            prices_df,
            f"{dataset_name}.{prices_table_name}",
            project_id='bigquery-sandbox-385813',
            if_exists='replace',
            table_schema=prices_schema_dicts,
            credentials=credentials
        )
```

Gerando 2 tabelas (cadastro e prices) para o dataset (sprint3)

Linha	table	count
1	cadastro	749

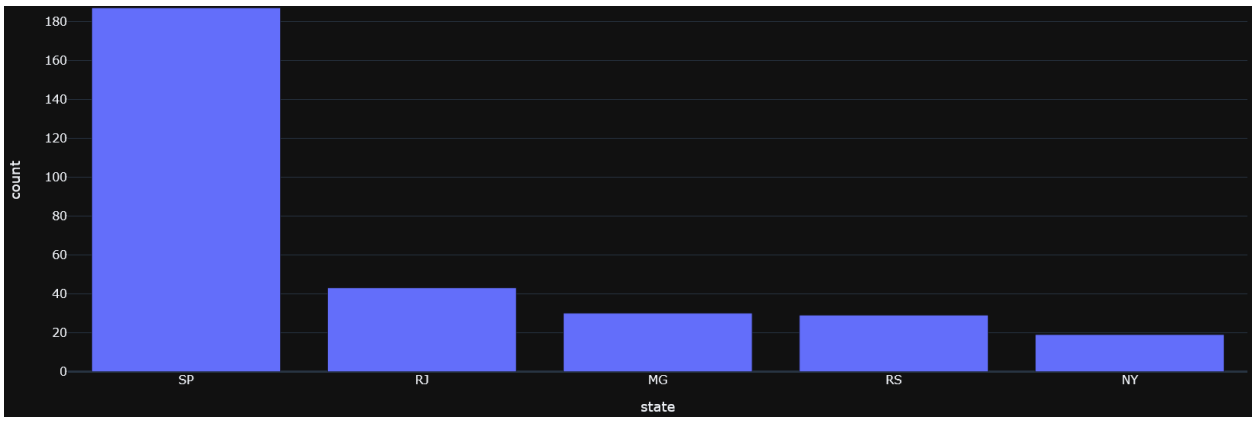
Linha	table	count
1	prices	253787

Resultado da análise de dados

Estados com mais empresas listadas no mercado financeiro

```
SELECT
  state,
  COUNT(ticker) as count
FROM sprint3.cadastro
  WHERE state IS NOT NULL
  GROUP BY state
  ORDER BY count DESC
  LIMIT 5
```

state	count
SP	187
RJ	43
MG	30
RS	29
NY	19



Setores e indústrias com mais empresas listadas

```
SELECT
  sector,
  industry,
  COUNT(ticker) as count
```

```
FROM sprint3.cadastro
  GROUP BY sector, industry
  ORDER BY count DESC
  LIMIT 5
```

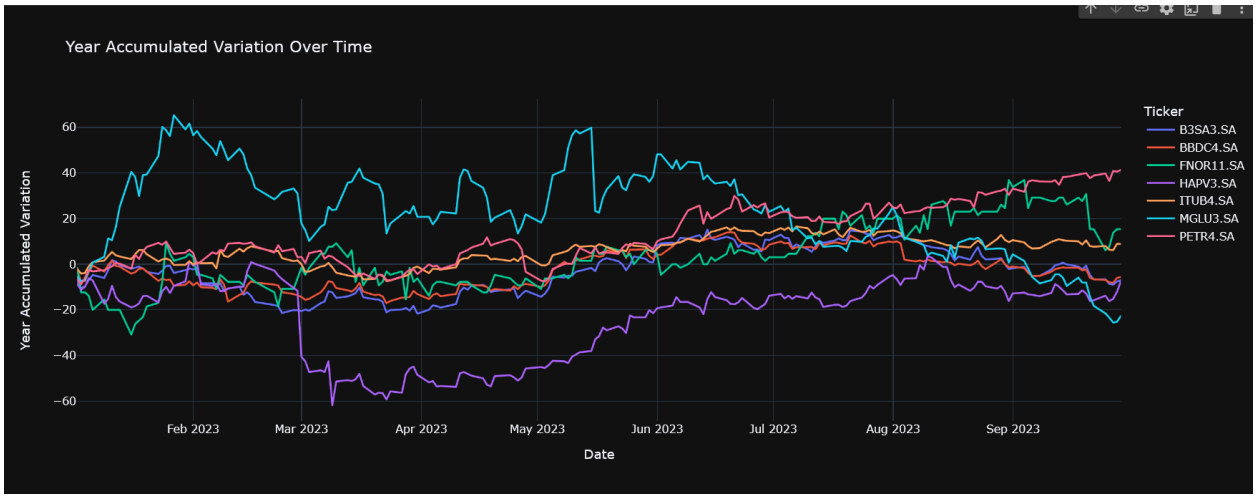
Linha	sector	industry	count
1	Financial Services	Banks—Regional	31
2	Utilities	Utilities—Regulated Electric	22
3	Real Estate	REIT—Diversified	15
4	Basic Materials	Steel	14
5	Utilities	Utilities—Renewable	13

Analise de rentabilidade das ações com maior média de negociações

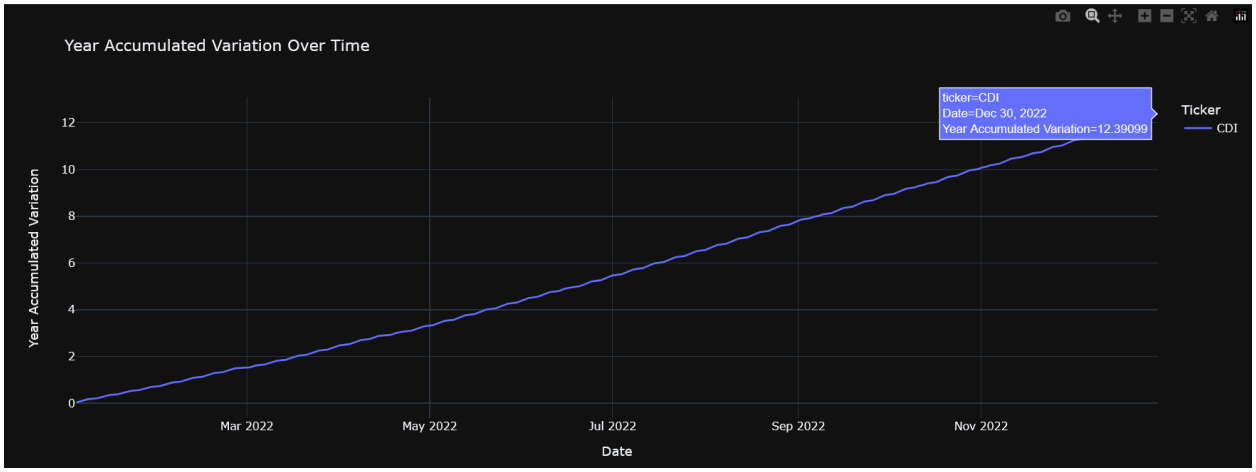
```
query = """
  SELECT *
  FROM sprint3.prices p
  LEFT JOIN sprint3.cadastro c ON p.ticker = c.ticker
  WHERE year = 2022 and p.ticker IN (SELECT ticker FROM sprint3.prices
                                     GROUP BY ticker
                                     order by AVG(volume) DESC
                                     limit 7)
  ORDER BY date ASC
"""
query_job = client.query(query)
results = query_job.result()
fig = px.line(df, x='date', y='year_accumulated_variation',
color='ticker',
              title='Year Accumulated Variation Over Time',
              labels={'year_accumulated_variation': 'Year Accumulated
Variation', 'date': 'Date'},
              template='plotly_dark')

fig.update_traces(mode='lines')
fig.update_layout(legend_title_text='Ticker')
fig.show()
```

2022

2023



Variação do CDI em 2022



Quantos ativos superaram o cdi ao final do ano de 2022

```
SELECT 'superior ao CDI' condition, count(ticker) as count FROM sprint3.prices
where date = '2022-12-29' and year_accumulated_variation > (SELECT
year_accumulated_variation
  FROM sprint3.prices p
  LEFT JOIN sprint3.cadastro c ON p.ticker = c.ticker
  WHERE year = 2022 and p.ticker = 'CDI'
  ORDER BY date DESC
  limit 1 )
```

condition	count
superior ao CDI	87

Maiores variações diárias

```
  SELECT date, p.ticker, close, prev_close, daily_variation
  FROM sprint3.prices p
  LEFT JOIN sprint3.cadastro c ON p.ticker = c.ticker
  ORDER BY daily_variation DESC
  LIMIT 5
```

date	ticker	close	prev_close	daily_variation
2023-01-02	PDGR3.SA	12.0	0.119999997317...	99.00000223517...
2022-06-14	HFOF11.SA	7129.0	71.38999938964...	98.85992521291...
2022-06-14	TMOS34.SA	5427.0	54.45000076293...	98.66942009106...
2022-10-05	PRSN11B.SA	3.0	0.860000014305...	2.488372034998...
2023-04-24	CTNM3.SA	12.01000022888...	4.010000228881...	1.995012354957...

Nitidamente, é possível perceber que a diferença entre o close e o prev_close apresenta uma discrepância muito grande. Teria que verificar se são casos de split ou inplit de ações (situações nas quais uma ação que custa 100 reais vira 10 ações de 10 reais ou a situação inversa). Seria necessário buscar essas informações que alteram bruscamente as cotações das ações mas que não alteram o valor final.

Menores variações diárias

```
SELECT date, p.ticker, close, prev_close, daily_variation
FROM sprint3.prices p
LEFT JOIN sprint3.cadastro c ON p.ticker = c.ticker
ORDER BY daily_variation ASC
```

date	ticker	close	prev_close	daily_variation
2022-06-15	HFOF11.SA	71.20999908447...	7129.0	-0.99001122189...
2022-06-15	TMOS34.SA	54.27000045776...	5427.0	-0.98999999991...
2022-11-17	TRVC34.SA	26.25	420.0	-0.9375
2023-09-20	CPTS11B.SA	10.28081226348...	102.8081207275...	-0.89999999814...
2023-08-31	BRPR3.SA	360.0	1911.0	-0.81161695447...

Me parece o mesmo caso do tópico acima, porém a situação inversa.

Média de volume de negociações por setor e indústria

```
SELECT
  sector, industry, AVG(volume) as avg_volume
 FROM sprint3.prices p
LEFT JOIN sprint3.cadastro c ON p.ticker = c.ticker
GROUP BY sector, industry
ORDER BY avg_volume DESC
LIMIT 5
```

sector	industry	avg_volume
Financial Services	Financial Data & Stock Exchan...	42713615.29680...
Financial Services	Insurance—Life	33972247.43085...
Consumer Cyclical	Specialty Retail	29810734.25947...
Energy	Oil & Gas Integrated	22479433.16152...
Basic Materials	Other Industrial Metals & Mining	14659295.77142...