1. **Engenharia de Dados**

O nosso modelo Ingestão de dados é baseado em scrapings (raspagens), e não fontes diretas de API’s ou plataformas do Mercado Financeiro, então todas as nossas fontes são gratuitas e não sistêmicas. A princípio todos os dados raspados serão ingeridos em um banco de dados para ser consumidos pela IA e pelo usuário final. Nos próximos pontos vamos mostrar como estamos idealizando o nosso modelo de banco de dados, e como será a raspagem desses dados.

* 1. “É ilegal e imoral consumir essas fontes!”

Essa alegação aconteceu, e não vimos a necessidade de analisar se infringe a LGPD (Lei Geral de Proteção de Dados), visto que todos os dados que estamos consumindo são disponibilizados pelas próprias empresas listadas na bolsa (dados de Finanças) e pela B3 (dados de cotação). E como é direito do Investidor ter acesso a essas informações, e todos os dados disponíveis são tratados e regulamentados por órgãos federais (CVM por exemplo) para que não exponha informações sigilosas de empresas e clientes, essa alegação é irrelevante.

* 1. Como consumimos os dados na primeira fase da APS?

Nessa primeira fase da APS focamos no desenvolvimento de um modelo funcional de Machine Learning (ML), então consumimos os dados de forma manual. No time de Finanças baixamos os dados manualmente da CVM (Comissão de Valores Mobiliários) para tratar e testar os modelos de ML, e no time de Price acessamos os dados diretamente do Yahoo Finance nós estudos de preço.

* 1. Scraping de Dados de Finanças

Bom conforme mencionado anteriormente os dados de finanças são oriundos da CVM que é um órgão que regulamenta o mercado financeiro brasileiro. É uma fonte extremamente confiável que dispõe os dados no formato de “Dados Abertos” ou CKAN. Basicamente as empresas na bolsa envia os dados do período do resultado contábil trimestral, através de um documento eletrônico chamado ITR (Formulário de Informações Trimestrais) que é encaminhado para a CVM através de um sistema que somente as empresas na bolsa tem acesso o “Empresas.NET”. O acesso a esses arquivos pode ser feito de 3 formas, acessando o site e baixando manualmente, via API CKAN (tem somente os dados dos últimos 5 anos) e via FTP (tem os dados desde 2011). O Código para extrair os dados de finanças é extremamente simples, segue:

import requests

from bs4 import BeautifulSoup

import json

import re

#Via FTP

source = requests.get(f"https://dados.cvm.gov.br/dados/CIA\_ABERTA/DOC/ITR/DADOS/").content

soup = BeautifulSoup(source, 'html.parser')

contents=soup.find\_all('pre')

for x in contents:

    for a in x.find\_all('a', href=True):

        if "../"!=f'{a["href"]}':

            source2 =f"https://dados.cvm.gov.br/dados/CIA\_ABERTA/DOC/ITR/DADOS/{a['href']}"

            r = requests.get(source2, allow\_redirects=True)

            open(rf'C:\Users\marqu\Desktop\Marcus\Estudos Finanças\IA\_FINAN-AS\_APS\_2023\CVM\{a["href"]}', 'wb').write(r.content)

Esse código baixa as informações via FTP no formato ZIP. Olha o histórico que ele tem:

Tabela

Descrição gerada automaticamenteDentro de cada arquivo zipado contém os seguintes CSV’s:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

O código para extrair via API CKAN, é bem parecido:

source = requests.get(f"https://dados.cvm.gov.br/api/3/action/package\_show?id=cia\_aberta-doc-itr").content

y = json.loads(source)

y=y["result"]["resources"]

for x in y:

    name=x["name"]

    name=re.findall('\(.\*?\)',name)

    #name=name.replace("(","")

    #name=name.replace(")","")

    if not name:

        print(name)

    else:

        name=name[0].replace("(","")

        name=name.replace(")","")

        b=x["url"]

        print(x)

        r = requests.get(b, allow\_redirects=True)

        open(f'/kaggle/working/itr\_cia\_aberta\_{name}.zip', 'wb').write(r.content)

E conforme mencionado anteriormente o histórico é de cinco anos, segue exemplo:

Texto

Descrição gerada automaticamente

* 1. Scraping de Dados de Price

Como nosso modelo de análises é baseada no valor das empresas, raspar as cotações de forma menos tempestiva, ou seja, ter um fluxo de raspagem que não seja RTD (Real Time Data) com uma periodicidade de ingestão no banco de dados mais longa (Semanal, mensal, trimestral....). Ter o Yahoo Finance (plataforma do Yahoo para análises do mercado financeiro) como fonte para dados de cotações atende os nossos requisitos. O acesso aos dados do Yahoo Finance é feito com uma biblioteca do Python chamada “yfinance”. Segundo o criador dessa biblioteca, a yfinance não é filiada ao Yahoo, e utiliza as API’s públicas do Yahoo para raspar os dados do mercado financeiro (AROUSSI et al., 2023). Então basicamente é uma ferramenta que simplifica uma API pública de uma empresa privada. O código que vamos utilizar para raspar os dados é simples, segue:

import pandas as pd

import yfinance as yf

txt=pd.read\_csv("ticks-bolsa/TICKS.csv",sep=";")

df\_data\_complete=pd.DataFrame()

try:

    for x in txt.loc:

        try:

            df\_data = yf.Ticker(f"{x['Código']}.sa")

            df\_data = df\_data.history(start="1940-01-01", end=None, interval = "1d")

            df\_data['Date'] = df\_data.index

            df\_data=df\_data.reset\_index(drop=True)

            df\_data['Date'] = pd.to\_datetime(df\_data["Date"]).dt.date

            df\_data["Empresa"]=x['Nome']

            df\_data.to\_csv(f"{x['Nome']}\_price\_daily\_historic.csv", sep=";", index=False, encoding='latin-1', mode="w",decimal=',')

            df\_data = df\_data[['Open', 'High', 'Low', 'Close', 'Volume','Date']]

            df\_data\_complete=pd.concat([df\_data,df\_data\_complete]).reset\_index(drop=True)

            df\_data = yf.Ticker(f"{x['Código']}.sa")

            df\_data = df\_data.history(start="1940-01-01", end=None, interval = "1mo")

            df\_data['Date'] = df\_data.index

            df\_data=df\_data.reset\_index(drop=True)

            df\_data['Date'] = pd.to\_datetime(df\_data["Date"]).dt.date

            df\_data["Empresa"]=x['Nome']

            df\_data.to\_csv(f"{x['Nome']}\_price\_monthly\_historic.csv", sep=";", index=False, encoding='latin-1', mode="w",decimal=',')

            df\_data = yf.Ticker(f"{x['Código']}.sa")

            df\_data = df\_data.history(start=n\_payday, end=None, interval = "1wk")

            df\_data['Date'] = df\_data.index

            df\_data=df\_data.reset\_index(drop=True)

            df\_data['Date'] = pd.to\_datetime(df\_data["Date"]).dt.date

            df\_data["Empresa"]=x['Nome']

            df\_data.to\_csv(f"{x['Nome']}\_price\_minute\_historic.csv", sep=";", index=False, encoding='latin-1', mode="w",decimal=',')

        except:

            print(x['Nome'])

except:

    print("fim")

df\_data\_complete.to\_csv(f"Completão.csv", sep=";", index=False, encoding='latin-1', mode="w",decimal=',')

Esse código basicamente ler um arquivo “De Para”, que tem todos os Tickers (código de identificação da empresa na bolsa), e faz uma busca na API do Yahoo Finance, nos retornando informações diárias, semanais e mensais de cada empresa da bolsa desde seu IPO (Oferta Pública Inicial).

* 1. Pipeline de Ingestão de dados

Conforme mencionado antes, nosso foco na primeira fase da APS foi desenvolver um modelo funcional de Machine Learning com dados de Price e Finanças. Mas já fizemos alguns rascunhos de como pretendemos ingerir os dados necessários e os dados gerados pelas IA’s em possíveis bancos de dados e servidores. Estamos levando em consideração a viabilidade do Pipeline, desde custo à desenvolvimento.

* 1. Banco de dados com o SQL Server no Windows Server (On Premisse)

A ideia desse pipeline é rodar em um servidor local (On Premisse), e o script de raspagem de dados vai rodar no próprio servidor. Segue o fluxo que fizemos:

Interface gráfica do usuário

Descrição gerada automaticamente

Após raspar os dados, eles seriam disponibilizados em uma pasta do servidor, onde cada arquivo representaria uma partição. Essas partições seriam ingeridas no banco de dados através de Jobs schedulados, que seriam basicamente ingestões programadas da última partição da pasta.

O nosso modelo de banco de dados é um Data Warehouse, visto que os dados já foram tratados anteriormente para ser disponibilizados nos dados abertos (Finanças) e pelo Yahoo Finance (Price). Teríamos que ter um código fora do ambiente banco de dados que pudesse analisar e aplicar os modelos de Machine Learning, mas que consumisse e injetaria os dados via biblioteca ODBC (Open Database Connectivity).

O script de raspagem pode ser Schedulado através de um script .bat ou pode ficar rodando a partir do momento que o servidor é ligado, mas teríamos que colocar um temporizador no código para que ele não ficasse fazendo milhares de requisições nas fontes, podendo gerar um tráfego de rede desnecessário ou ser derrubado pelo servidor-cliente.

* 1. AWS Back e Dados (Cloud Computing)

O AWS é uma das opções mais interessantes que temos hoje, visto que temos a nossa disposição todo um ecossistema para atender nossas necessidades quanto engenheiros da computação. O único problema é que os custos são consideráveis, e mesmo com a AWS disponibilizando o free tier de alguns produtos, existe o risco de extrapolar o uso disponibilizado e ser cobrado na fatura do cartão. Mas seria o melhor ambiente hoje para fazermos o deploy do aplicativo e conversar com o back end e banco de dados. Segue como ficou o modelo:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Parece ser mais difícil, mas o meu intuito inicial era só fazer um fluxo de ingestão no banco de dados e acabei desenvolvendo um fluxo inteiro de back end e acabei otimizando/economizando um banco de dados, visto que nesse exemplo usamos o Athena para consumir tabelas. Parece irrelevante, mas simplificamos a estruturação de banco de dados e fizemos desnecessário ter um Data WH (Warehouse) visto que no AWS Glue a gente salva os dados raspados já tratados no AWS S3 e depois consome eles no Athena.

Salvo as raspagens, esses dados são carregados em outro script do AWS Glue usando o BOTO3 que é um framework da AWS para interagir no ambiente da nuvem com linguagens de programação. Esse Job que vai aplicar os modelos de IA e depois criar uma base no AWS S3. Para conseguirmos acessar esses arquivos tratados, a gente vai consumir uma tabela com os arquivos salvos no AWS S3.

Toda a parte que controla o que sai e o que entra de arquivos confidenciais é o AWS Lambda, que é basicamente um trigger. O AWS AppSync é basicamente uma ponte entre a fonte e o Back End. O AWS Amplify é um framework de back end na nuvem, a ideia dele é facilitar o deploy (disponibilização) de aplicativos Web e Mobile, ele pode ser plugado em um ambiente que tenha o código do aplicativo como o github ou pode utilizar o próprio AWS S3 para armazena o código do aplicativo. Por fim temos o AWS Cognito que é um framework de autenticação de login de usuário com contas de outras plataformas como Google e Facebook, a ideia dele é economizar tempo desenvolvendo uma lógica de login e tentando protegê-la.

* 1. Windows Server + Kaggle (Hybrid)

A ideia desse modelo de ingestão é aproveitar a infraestrutura do Kaggle que é gratuita para rodar tanto a parte do web scraping quanto a parte de aplicação da IA. Segue como ficou o modelo:

Interface gráfica do usuário

Descrição gerada automaticamente

A gente conseguiria schedular os Jobs no Kaggle, então a ingestão iria acontecer de forma automática. Para conectar o ambiente do Kaggle no SQL Server, iriamos utilizar a biblioteca ODBC do Python. E como a gente faria todo o tratamento de dados no Kaggle para ser utilizada pela IA, não seria necessário ingerir dados brutos e ter um Data Warehouse no nosso banco de dados. Os pontos mais importantes dessa ideia, é que iriamos unir uma ideia com otimização. Visto que uma das nossas ideias era aplicar o Data Mesh (Democratização de dados) e disponibilizando os dados do Job em um repositório no Kaggle é a melhor forma de fazer isso. E otimização por que deixariamos o local server com menos tarefas para executar. Esse modelo ele é híbrido porque utiliza um banco de dados local server, e um ambiente de ETL na nuvem.

* 1. Data Mesh

O Data Mesh é em linhas gerais democratização de dados, onde o nosso usuário poderia baixar os nossos dados de forma simples e sem burocracias. Poderíamos cobrar um valor para baixar esses dados e isso não faria com que o data mesh não fosse uma forma democratizada de consumir dados, porque o objetivo principal é que o usuário não precise consumir uma API ou uma ferramenta de gerenciamento de banco de dados para baixar os dados tratados, ele teria um botão no front end onde ele conseguiria baixar os dados que ele está visualizando. Ou seja quando desenvolvemos um modelo data mesh estamos pensando em uma visão cliente.

Apesar da simplicidade dessa ideia o data mesh se faz necessário em muitas empresas que tem “owners” de dados, isso significa que os usuários e clientes tem dificuldades em acessar e consumir informações e esse é um problema escalar que pode ser resolvido na raiz, então quando arquitetamos um back end voltado ao data mesh já eliminamos esse problema.

Pelos fluxos que desenvolvemos o modelo da AWS e o Hybrid são os que possuem características que poderiam facilitar no desenvolvimento do conceito Data Mesh, principalmente o modelo Hybrid visto que disponibilizaríamos um repositório sem custos em uma das principais plataformas de dados da atualidade.