client = "ARX"

a = pl.read\_ipc(r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_2023-06-01.feather")

a = a.filter(pl.col("cliente") != client)

a.write\_ipc(r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_2023-06-01.feather")

import polars as pl

import pandas as pd

import requests

import polars as pl

from xbbg import blp

class Moodys\_rating():

LINKS = [

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Corporates",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Financial+Institutions",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Fixed+Income+Funds",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Infrastructure+Projects",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Insurance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Structured+Finance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Sub+Sovereign"

]

def \_\_init\_\_(self):

self.create\_dictionary()

self.requests\_links()

def create\_dictionary(self):

self.dictionary\_ratings = {}

self.dictionary\_ratings["company"] = []

self.dictionary\_ratings["date"] = []

self.dictionary\_ratings["rating"] = []

def requests\_links(self):

for link in self.LINKS:

self.r = requests.get(link)

self.build\_dictionary()

def build\_dictionary(self):

for dictionary in r.json():

for key in self.dictionary\_ratings.keys():

self.dictionary\_ratings[key].append(None)

self.dictionary\_ratings["company"][-1] = dictionary["es"]["entity"]

self.dictionary\_ratings["date"][-1] = dictionary["es"]["date"]

if len(dictionary["es"]["instruments"]) == 1:

self.dictionary\_ratings["rating"][-1] = dictionary["es"]["instruments"][0]["ratingValue"]

else:

for rating in dictionary["es"]["instruments"]:

rating\_type = rating["ratingType"]

if rating\_type in self.dictionary\_ratings:

self.dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

else:

self.dictionary\_ratings[rating\_type] = [None]\*len(self.dictionary\_ratings["company"])

self.dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

dictionary\_ratings = {}

dictionary\_ratings["company"] = []

dictionary\_ratings["date"] = []

for dictionary in r.json():

for key in dictionary\_ratings.keys():

dictionary\_ratings[key].append(None)

dictionary\_ratings["company"][-1] = dictionary["es"]["entity"]

dictionary\_ratings["date"][-1] = dictionary["es"]["date"]

for rating in dictionary["es"]["instruments"]:

rating\_type = rating["ratingType"]

if rating\_type in dictionary\_ratings:

dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

else:

dictionary\_ratings[rating\_type] = [None]\*len(dictionary\_ratings["company"])

dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

def str\_to\_list(texto):

return texto[1:-1].replace('},{"es"','},\*\*trocar\*\*{"es"').split("\*\*trocar\*\*")

def procurar(palavra,texto):

tam=len(palavra)

ini=texto.find(palavra)

fim=texto[ini+tam+1:].find('"')

return texto[ini+tam+1:ini+tam+fim+1], texto[ini+tam+fim+1:]

def str\_to\_dict(texto):

dicio={}

dicio['Empresa'],texto=procurar('"entity":',texto)

dicio['Data'],texto=procurar(',"date":',texto)

##Instrumentos - pode ter mais de 1, então separar a lista e transformar em dict talvez?

ini=texto.find('{"ratingValue":')

while ini>-1:

r\_nota,texto=procurar('{"ratingValue":',texto)

if r\_nota!="WR":

r\_nome,texto=procurar('"ratingType":',texto)

dicio[r\_nome]=r\_nota

ini=texto.find('{"ratingValue":')

return dicio

def obter\_ratings\_m(l):

r=requests.get(l)

lista=str\_to\_list(r.text)

df=pd.Series(str\_to\_dict(lista[0]))

for t in lista[1:]:

df=pd.concat([df,pd.Series(str\_to\_dict(t))],axis=1)

df=df.T

return df

links=["https://www.moodyslocal.com/api/countries/BR/ratings?sector=Corporates","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Financial+Institutions",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Fixed+Income+Funds","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Infrastructure+Projects",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Insurance","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Structured+Finance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Sub+Sovereign"]

ratings\_m=obter\_ratings\_m(links[0])

for link in links[1:]:

ratings\_m=pd.concat([ratings\_m,obter\_ratings\_m(link)])

ratings\_m=ratings\_m.dropna(thresh=3) #Retirar os que não possuem rating algum (possivelmente era tudo WR e eu não coloquei)

import polars as pl

from datetime import datetime

from numpy import busday\_offset

import os

class Cra\_db():

'As of 15/may/23, up2data stopped sending security list for CRA and CRI'

CRA\_FIRST\_PATH = r"Y:\20230428\Fixed\_Income\CRA\SecurityList\Fixed\_Income\_FixedIncomeCRAInstrumentFile\_20230427\_2.json"

CRI\_FIRST\_PATH = r"Y:\20230428\Fixed\_Income\CRI\SecurityList\Fixed\_Income\_FixedIncomeCRIInstrumentFile\_20230427\_2.json"

DATABASE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa"

CRA\_DATABASE\_NAME = "db\_cra.parquet"

CRA\_COLUMNS = {

"TckrSymb": "ticker", 'XprtnDt': 'vencimento', 'CrpnNm': 'securitizadora', 'IsseUnitPric': 'pu\_emitido',

'IssdQty': 'quantidade\_emitido', 'IsseTtlVal': 'total\_mercado', "IntrstParamsPctg": "taxa\_emitido",

'InstrmRmnrtnTp': 'indexador', 'ISIN': 'isin', "OTCSgmtNm": "tipo", "InstrmEmssnDt": "data\_emitido",

}

def \_\_init\_\_(self):

pass

#self.update\_info()

def get\_first\_info(self):

files = os.listdir(self.DATABASE\_PATH)

if self.CRA\_DATABASE\_NAME in files:

raise Exception("Já existe db de cra e cri. Deletar na mão em vez de reescrever.")

cra\_df = pl.read\_json(self.CRA\_FIRST\_PATH)

cri\_df = pl.read\_json(self.CRI\_FIRST\_PATH)

self.df = pl.concat([cra\_df, cri\_df])

self.format\_df()

self.df.write\_parquet(self.DATABASE\_PATH + "\\" + self.CRA\_DATABASE\_NAME)

def format\_df(self):

self.df = self.df.rename(self.CRA\_COLUMNS).select(pl.col(self.CRA\_COLUMNS.values()))

for column in ["taxa\_emitido", "pu\_emitido", "quantidade\_emitido", "total\_mercado"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.cast(pl.Float32)

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col("taxa\_emitido")/100,

pl.col(["vencimento", "data\_emitido"]).str.strptime(pl.Date, "%Y-%m-%d"),

(pl.col("quantidade\_emitido")\*pl.col("pu\_emitido")).alias("reais\_emitido"),

pl.lit(None).alias("empresa")

)

self.df = self.df.with\_columns(

pl.when((pl.col("indexador") == "DI") & (pl.col("taxa\_emitido") != 0))

.then(

pl.when(pl.col("taxa\_emitido") < 0.2)

.then("DI +")

.otherwise(

pl.when(pl.col("taxa\_emitido") > 0.9)

.then("% DI")

.otherwise(pl.col("indexador"))

)

)

.otherwise(pl.col("indexador"))

.keep\_name(),

pl.when(pl.col(pl.Date).dt.year() > 2150)

.then(None)

.otherwise(pl.col(pl.Date))

.keep\_name()

)

def write\_cra(self):

pass

#Só escrever se eu conseguir criar algum update para os cris e cras

class Debenture\_db():

DEBENTURE\_FIRST\_PATH = (r"Y:\20230428\Debentures\_MTM\SecurityList\Debentures"

+ "\_MTM\_SecurityListDebentureFileV2\_20230428\_13.json")

DATABASE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa"

DEB\_DATABASE\_NAME = "db\_deb.parquet"

DEBENTURE\_COLUMNS = {

"TckrSymb": "ticker", 'XprtnDt': 'vencimento', 'FxdIntrstRate': 'taxa\_emitido', 'IndxShrtNm': 'indexador',

'CrpnNm': 'empresa', "IsseUnitPric": 'pu\_emitido', 'IssdQty': 'quantidade\_emitido',

'MktQty': 'quantidade\_mercado\_emitido', 'Law12431SpprtInd': 'incentivada', "Clss": "conversivel",

'ISIN': 'isin', "IntrstAcctgIntrvlUnit": "unidade", "InstrmDdlnClclsRef": "critério",

"IntrstSprdDesc":"metodo\_taxa", "IntrstBizDaysClclsRef": "252/360", "IsseDt": "data\_emitido",

"Law12431Artl": "artigo"

}

def \_\_init\_\_(self, date):

self.date = date

self.largest\_file()

self.format\_df()

self.update\_database()

def get\_first\_info(self):

self.df = pl.read\_json(self.DEBENTURE\_FIRST\_PATH)

files = os.listdir(self.DATABASE\_PATH)

if self.DEB\_DATABASE\_NAME in files:

raise Exception("Já existe db de debentures. Deletar na mão em vez de reescrever.")

self.format\_df()

self.df.write\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

def largest\_file(self):

files = os.scandir("Y:\\" + self.date.strftime("%Y%m%d") + r"\Debentures\_MTM\SecurityList")

paths = []

sizes = []

for file in files:

paths.append(file.path)

sizes.append(file.stat().st\_size)

df = pl.DataFrame({"paths": paths, "sizes": sizes})

json\_file = df.filter(pl.col("paths").str.contains(".json")).sort(by="sizes", descending = True)[0, 0]

self.df = pl.read\_json(json\_file)

def format\_df(self):

self.df = self.df.rename(self.DEBENTURE\_COLUMNS).select(self.DEBENTURE\_COLUMNS.values())

for column in ["taxa\_emitido", "pu\_emitido", "quantidade\_emitido", "quantidade\_mercado\_emitido"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.cast(pl.Float32)

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col("taxa\_emitido")/100,

pl.col(["vencimento", "data\_emitido"]).str.strptime(pl.Date, "%Y-%m-%d"),

pl.when(pl.col("artigo") == "")

.then(None)

.otherwise(pl.col("artigo"))

.cast(pl.Int16)

.keep\_name(),

pl.when(pl.col("incentivada") == "true")

.then(True)

.otherwise(False)

.keep\_name(),

(pl.col("quantidade\_emitido")\*pl.col("pu\_emitido")).alias("reais\_emitido"),

(pl.col("quantidade\_mercado\_emitido")\*pl.col("pu\_emitido")).alias("reais\_mercado\_emitido"),

pl.lit("DEB").alias("tipo")

)

self.df = self.df.with\_columns(

pl.when((pl.col("indexador") == "DI") & (pl.col("taxa\_emitido") != 0))

.then(

pl.when(pl.col("taxa\_emitido") < 0.2)

.then("DI +")

.otherwise(

pl.when(pl.col("taxa\_emitido") > 0.9)

.then("% DI")

.otherwise(pl.col("indexador"))

)

)

.otherwise(pl.col("indexador"))

.keep\_name(),

pl.when(pl.col(pl.Date).dt.year() > 2150)

.then(None)

.otherwise(pl.col(pl.Date))

.keep\_name()

)

def update\_database(self):

self.db = pl.read\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

self.db = pl.concat([self.db, self.df]).unique()

self.db.write\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

class Debenture\_trade():

TRADE\_COLUMNS = {"TckrSymb": "ticker", "MinPric": "price\_min", "MaxPric": "price\_max", "TradAvrgPric": "price\_avg",

"TradQty": "trades\_number", "FinInstrmQty": "trades\_quantity", "NtlFinVol": "financial\_volume",

"OprnClssfctnTpNm": "passagem", "EvtAndSttlmGap": "liquidation"}

WRITE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Trades debentures"

def \_\_init\_\_(self, date):

self.date = date

self.largest\_file()

self.format\_df()

self.write\_df()

def largest\_file(self):

files = os.scandir("Y:\\" + self.date.strftime("%Y%m%d") + r"\Debentures\_MTM\TradeInformation")

paths = []

sizes = []

for file in files:

paths.append(file.path)

sizes.append(file.stat().st\_size)

df = pl.DataFrame({"paths": paths, "sizes": sizes})

json\_file = df.filter(pl.col("paths").str.contains(".json")).sort(by="sizes", descending = True)[0, 0]

self.df = pl.read\_json(json\_file)

def format\_df(self):

self.df = self.df.rename(self.TRADE\_COLUMNS).select(self.TRADE\_COLUMNS.values())

for column in ["price\_min", "price\_max", "price\_avg", "financial\_volume", "trades\_number", "trades\_quantity"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col(["price\_min", "price\_max", "price\_avg", "financial\_volume"]).cast(pl.Float32),

pl.col(["trades\_number", "trades\_quantity"]).cast(pl.Int32)

)

def write\_df(self):

self.df.write\_ipc(self.WRITE\_PATH + "\\" + "trades\_deb\_" + self.date.strftime("%Y-%m-%d") + ".feather")

if \_\_name\_\_ == "\_\_main\_\_":

DAYS\_CHECK = 7

today = datetime.today()

for days\_subtract in range(1, DAYS\_CHECK):

date = busday\_offset(datetime.today().strftime("%Y-%m-%d"), (-1)\*days\_subtract)

date = date.item()

date\_string = date.strftime("%Y%m%d")

print(date\_string)

if exist\_up2data\_day(date\_string):

print("1")

if not day\_already\_copied(date):

print("2")

Debenture\_trade(date)

Debenture\_db(date)

class Df\_operations():

WB\_NAME = "RUN Crédito Privado.xlsm"

SHEET\_NAME = "Add RUN"

INFO\_LAST\_LINE = "B4"

RANGE\_DRAFT = "I5:V"

RANGE\_FORMATTED = "C6:G2000"

CELL\_PRINT = RANGE\_FORMATTED[:2]

COLUMNS\_NAME = ["ticker", "vol bid", "bid", "ask", "vol ask"]

def \_\_init\_\_(self):

pass

def obtain\_df(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet = self.wb.sheets[self.SHEET\_NAME]

last\_line = self.sheet.range(self.INFO\_LAST\_LINE).value

my\_date\_handler = lambda year, month, day, \*\*kwargs: f"{month:02d}/{year-2000:02d}"

run\_values = self.sheet.range(self.RANGE\_DRAFT + str(int(last\_line))).options(dates=my\_date\_handler).value

self.df = pl.DataFrame(run\_values[1:], orient = "row")

self.columns\_rename = run\_values[0]

def rename\_columns(self):

dict\_rename\_columns = {}

old\_columns\_names = self.df.columns

new\_columns = []

for index in range(0, len(self.columns\_rename)):

if self.columns\_rename[index] is not None:

new\_columns.append(self.columns\_rename[index])

dict\_rename\_columns[old\_columns\_names[index]] = self.columns\_rename[index]

if ("ticker" not in new\_columns):

raise Personal\_exception("Não há coluna 'ticker'")

if ("ask" not in new\_columns) and ("bid" not in new\_columns):

raise Personal\_exception("Não há coluna 'bid' nem 'ask'")

self.df = self.df.rename(dict\_rename\_columns)

for name in self.COLUMNS\_NAME:

if name not in new\_columns:

self.df = self.df.with\_columns(pl.lit(None).alias(name))

if "vol bid/ask" in new\_columns:

self.df = self.df.with\_columns(

pl.col("vol bid/ask").alias("vol bid"),

pl.col("vol bid/ask").alias("vol ask")

)

exist\_vencimento = "venc" in self.columns\_rename

if exist\_vencimento:

self.df = self.df.select(self.COLUMNS\_NAME + ["venc"])

self.include\_maturity\_lf()

else:

self.df = self.df.select(self.COLUMNS\_NAME)

self.str\_columns = self.df.select(pl.col(pl.Utf8)).columns

def str\_columns\_adjust(self):

self.df = self.df.with\_columns(

pl.col(self.str\_columns)

.str.strip()

.str.to\_uppercase()

.str.replace(",", ".")

).with\_columns(

pl.when(pl.col("ticker") == "")

.then(None)

.otherwise(pl.col("ticker"))

.keep\_name()

)

def include\_maturity\_lf(self):

'''As of may/23, there was no debenture starting with "LF"'''

self.df = self.df.with\_columns(

pl.when(pl.col("ticker").str.contains("^LF"))

.then(pl.col("ticker") + " - " + pl.col("venc"))

.otherwise(pl.col("ticker"))

.alias("ticker")

).select(pl.exclude("venc"))

def rows\_to\_exclude(self):

self.test = self.df

self.df = (self.df

.filter(

pl.any(pl.col(["bid", "ask"]).is\_null().is\_not())

&

pl.col("ticker").is\_null().is\_not()

)

.with\_columns(

pl.col(["vol bid", "bid", "ask", "vol ask"]).fill\_null(0)

)

)

self.df = self.df.filter(

(pl.col("ticker")

.str.contains("ATIVO|TICKER|DEB[ÊE]NTURE|---|INFRA|SPREAD OVER|COMPRO|CPN|CRA$|CRI$").is\_not()

)

&

(pl.all(

~pl.col(["bid", "ask"])

.str.contains("[bB][iI][dD]|[aA][sS][kK]|VALUE|VALOR|##|IPCA|CDI|COMPRA|VENDA|INDIC 1MM"))

)

)

exist\_duplicates = not (self.df.filter(pl.col("ticker").is\_duplicated()).is\_empty())

if exist\_duplicates:

tickers\_duplicateds = self.df.select(pl.col("ticker")).filter(pl.col("ticker").is\_duplicated()).unique()

raise Personal\_exception(f"Há tickers duplicados:\n{tickers\_duplicateds}")

def bid\_ask\_adjust(self):

for column\_name in ["bid", "ask"]:

if column\_name in self.str\_columns:

self.df = self.df.with\_columns(

pl.col(column\_name)

.str.replace("%$|v$", "")

.str.strip()

).with\_columns(

pl.when(pl.col(column\_name).is\_in(["-", ""]))

.then(None)

.otherwise(pl.col(column\_name))

.cast(pl.Float64)

.keep\_name()

)

else:

self.df = self.df.with\_columns(pl.col(column\_name).cast(pl.Float64))

self.df = self.df.filter(

(pl.col("bid") > 0)

| (pl.col("ask") > 0)

)

#Minimum value is considered at the moment 180% CDI. Lower values might confuse with cdi + 1.5%

minimum\_value\_to\_divide = 1.8

divide\_by\_100 = not self.df.filter(pl.any(pl.col(["bid", "ask"]) > minimum\_value\_to\_divide)).is\_empty()

if divide\_by\_100:

#Maximum value is considered at the moment as IPCA + 10%, higher values might confuse with cdi + 0.5%

maximum\_value\_to\_not\_divide = 0.1

not\_divide\_by\_100 = not self.df.filter(

((pl.col("bid") > 0) & (pl.col("bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("ask") > 0) & (pl.col("ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_100:

message = "Alguns valores em bid/ask estão em % e outros não"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["bid", "ask"])/100)

ask\_higher\_bid\_df = (self.df

.filter(

(pl.col("ask") >= pl.col("bid"))

&

(pl.col("bid") != 0)

)

)

ask\_higher\_bid = not ask\_higher\_bid\_df.is\_empty()

if ask\_higher\_bid:

raise Personal\_exception(f"Existe ask maior ou igual a bid\n\n{ask\_higher\_bid\_df}")

def volume\_adjust(self):

self.df\_old = self.df

for column\_name in ["vol bid", "vol ask"]:

if column\_name in self.str\_columns:

self.df = self.df.with\_columns(

pl.when(pl.col(column\_name) == "")

.then(None)

.otherwise(pl.col(column\_name))

.alias(column\_name)

)

self.df = self.df.with\_columns(

pl.col(column\_name)

.str.replace("[mM][lL][nN]|[mM][mM]$|[mM]|[kK]$", "")

.str.replace("^~|^>", "")

.str.strip()

.cast(pl.Float32)

/

pl.when(pl.col("vol ask").str.contains("[kK]$"))

.then(1000)

.otherwise(1)

)

#Minimum value is considered at the moment 100mm. Lower values might confuse with high volume

minimum\_value\_to\_divide = 100

divide\_by\_1000 = not self.df.filter(pl.any(pl.col(["vol bid", "vol ask"]) > minimum\_value\_to\_divide)).is\_empty()

minimum\_value\_to\_divide = 1e5

divide\_by\_million = not self.df.filter(pl.any(pl.col(["vol bid", "vol ask"]) > minimum\_value\_to\_divide)).is\_empty()

if divide\_by\_million:

#Maximum value is considered at the moment as 0.1mm. Higher values might confuse with 100k

maximum\_value\_to\_not\_divide = 1e2

not\_divide\_by\_million = not self.df.filter(

((pl.col("vol bid") > 0) & (pl.col("vol bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("vol ask") > 0) & (pl.col("vol ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_million:

message = "Alguns valores em vol bid/vol ask estão em K e outros em MM"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["vol bid", "vol ask"])/1e6)

elif divide\_by\_1000:

#Maximum value is considered at the moment as 0.1mm. Higher values might confuse with 100k

maximum\_value\_to\_not\_divide = 0.1

not\_divide\_by\_1000 = not self.df.filter(

((pl.col("vol bid") > 0) & (pl.col("vol bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("vol ask") > 0) & (pl.col("vol ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_1000:

message = "Alguns valores em vol bid/vol ask estão em K e outros em MM"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["vol bid", "vol ask"])/1000)

def print\_to\_excel(self):

self.sheet.range(self.RANGE\_FORMATTED).value = ""

self.sheet.range(self.CELL\_PRINT).options(index=False, header=False).value = self.df.to\_pandas()

try:

run = Df\_operations()

run.obtain\_df()

run.rename\_columns()

run.str\_columns\_adjust()

run.rows\_to\_exclude()

run.bid\_ask\_adjust()

run.volume\_adjust()

run.print\_to\_excel()

except Personal\_exception as error:

error\_global = error

print(error)

class Database\_fixed\_income():

DEB\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa\db\_deb.parquet"

CRA\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa\db\_cra.parquet"

COLUMNS = ["ticker", "vencimento", "taxa\_emitido", "indexador", "empresa", "pu\_emitido",

"isin", "data\_emitido", "reais\_emitido", "tipo", "incentivada", "artigo"]

def \_\_init\_\_(self):

self.get\_deb\_database()

self.get\_cra\_database()

self.concat\_databases()

def get\_deb\_database(self):

self.df\_deb = pl.read\_parquet(self.DEB\_PATH).select(pl.col(self.COLUMNS))

def get\_cra\_database(self):

self.df\_cra = pl.read\_parquet(self.CRA\_PATH)

self.df\_cra = self.df\_cra.with\_columns(

pl.lit(True).alias("incentivada"),

pl.lit(None).cast(pl.Int16).alias("artigo")

)

self.df\_cra = self.df\_cra.select(pl.col(self.COLUMNS))

def concat\_databases(self):

self.df = pl.concat([self.df\_deb, self.df\_cra])

class Anbima():

PATH\_ANBIMA\_DEB = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Debêntures\anbima\_deb\_"

PATH\_ANBIMA\_CRA = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Cra\anbima\_cra\_"

PATH\_ANBIMA\_TIT = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Títulos públicos\anbima\_tit\_"

COLUMNS\_CRA = ["ticker", "anbima", "referencia\_b", "duration\_anbima"]

def \_\_init\_\_(self, yesterday, df):

self.yesterday = yesterday

self.yesterday\_string = yesterday.strftime("%Y-%m-%d")

self.df = df

self.get\_anbima\_deb()

self.get\_anbima\_cra()

self.get\_anbima\_tit()

def get\_anbima\_deb(self):

while True:

try:

deb = pl.read\_ipc(self.PATH\_ANBIMA\_DEB + self.yesterday\_string + ".feather")

break

except FileNotFoundError:

self.check\_last\_anbima()

self.deb = deb

self.df = self.df.join(deb, how="left", on="ticker")

def check\_last\_anbima(self):

import requests

URL = 'https://www.anbima.com.br/informacoes/merc-sec-debentures/arqs/db'

r = requests.get(URL + self.yesterday.strftime("%y%m%d") + ".txt")

file\_found = (r.status\_code == 200)

if file\_found:

raise Personal\_exception(f"Último arquivo da anbima {self.yesterday\_string} não baixado.")

else:

self.yesterday = busday\_offset(self.yesterday\_string, -1)

self.yesterday = self.yesterday.item()

self.yesterday\_string = self.yesterday.strftime("%Y-%m-%d")

def get\_anbima\_cra(self):

cra = pl.read\_ipc(self.PATH\_ANBIMA\_CRA + self.yesterday\_string + ".feather")

cra = cra.select(pl.col(self.COLUMNS\_CRA))

self.cra = cra

self.df = self.df.join(cra, how="left", on="ticker", suffix="\_right")

for column in ["anbima", "referencia\_b", "duration\_anbima"]:

self.df = self.df.with\_columns(

pl.when((pl.col(column).is\_null()) & (~pl.col(column + "\_right").is\_null()))

.then(pl.col(column + "\_right"))

.otherwise(pl.col(column))

.alias(column)

).drop(column + "\_right")

def get\_anbima\_tit(self):

tit = pl.read\_ipc(self.PATH\_ANBIMA\_TIT + self.yesterday\_string + ".feather")

tit = tit.with\_columns(

("TIT: " + pl.col("ticker") + " " + pl.col("vencimento").dt.strftime("%y")).alias("ticker")

)

self.tit = tit

columns = []

for column in self.df.columns:

if column not in self.tit.columns:

columns.append(column)

self.tit = self.tit.with\_columns(pl.lit(None).alias(column))

self.tit = self.tit.select(pl.col(self.df.columns))

self.tit = self.tit.with\_columns(

pl.col(["pu\_emitido", "taxa\_emitido", "reais\_emitido", "duration\_anbima"])

.cast(pl.Float32),

pl.col("data\_emitido").cast(pl.Date),

pl.lit("Título público").alias("tipo"),

pl.col(["indexador", "isin", "empresa", "referencia\_b"]).cast(pl.Utf8),

pl.col("incentivada").cast(pl.Boolean),

pl.col("artigo").cast(pl.Int16)

)

self.df = pl.concat([self.tit, self.df])

class Workbook\_info():

WB\_NAME = "RUN Crédito Privado.xlsm"

SHEET\_NEW\_RUN = "Add RUN"

SHEET\_EXTRA = "RUN"

RANGE\_MAX\_LINE = "B2:B3"

RANGE\_NEW\_RUN = "C6:G"

RANGE\_EXTRA = "A18:F"

LAST\_LINE\_DELETE = "2000"

CLIENT\_CELL = "C4"

RANGE\_NEW\_RUN\_DELETE = RANGE\_NEW\_RUN + LAST\_LINE\_DELETE

RANGE\_FORMATTED = "H4:L" + LAST\_LINE\_DELETE

RANGE\_DRAFT = "I5:V" + LAST\_LINE\_DELETE

def \_\_init\_\_(self):

self.obtain\_sheets()

self.obtain\_new\_run()

self.obtain\_extras()

def obtain\_sheets(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet\_new\_run = self.wb.sheets[self.SHEET\_NEW\_RUN]

self.sheet\_extra = self.wb.sheets[self.SHEET\_EXTRA]

self.lines = self.sheet\_new\_run.range(self.RANGE\_MAX\_LINE).value

def obtain\_new\_run(self):

max\_line\_new\_run = str(int(self.lines[1]))

self.exist\_new\_run = self.lines[1] > 5

if self.exist\_new\_run:

self.new\_run = self.sheet\_new\_run.range(self.RANGE\_NEW\_RUN + max\_line\_new\_run).value

self.client = self.sheet\_new\_run.range(self.CLIENT\_CELL).value

if self.client is None:

raise Personal\_exception("Cliente está vazio")

else:

self.client = self.client.upper()

def obtain\_extras(self):

line\_extra = str(int(self.lines[0]))

self.exist\_extra = self.lines[0] > 17

if self.exist\_extra:

self.extra\_values = self.sheet\_extra.range(self.RANGE\_EXTRA + line\_extra).value

def clean\_lines(self):

cells\_to\_delete = [self.CLIENT\_CELL, self.RANGE\_NEW\_RUN\_DELETE, self.RANGE\_DRAFT]

cells\_to\_delete = ", ".join(cells\_to\_delete)

self.sheet\_new\_run.range(cells\_to\_delete).value = ""

class Data\_new\_run():

NEW\_COLUMNS\_NAMES = ["ticker", "vol bid", "bid", "ask", "vol ask", "cliente"]

FLOAT\_COLUMNS\_DEFAULT = NEW\_COLUMNS\_NAMES[1:-1]

STR\_COLUMNS\_DEFAULT = [NEW\_COLUMNS\_NAMES[0], NEW\_COLUMNS\_NAMES[-1]]

def \_\_init\_\_(self, wb):

self.wb = wb

self.obtain\_df()

self.rename\_columns()

self.check\_duplicates()

self.check\_columns\_types()

def obtain\_df(self):

empty\_client = (self.wb.new\_run[0][0] == None)

df = pl.DataFrame(self.wb.new\_run, orient = "row")

self.df = df.with\_columns(pl.lit(self.wb.client).alias("cliente"))

def rename\_columns(self):

dict\_rename\_columns = {}

old\_columns\_names = self.df.columns

for index in range(0, 6):

dict\_rename\_columns[old\_columns\_names[index]] = self.NEW\_COLUMNS\_NAMES[index]

self.df = self.df.rename(dict\_rename\_columns).with\_columns(pl.col(pl.Boolean).cast(pl.Float64))

def check\_duplicates(self):

exist\_duplicates = not (self.df.filter(pl.col("ticker").is\_duplicated()).is\_empty())

if exist\_duplicates:

tickers\_duplicateds = self.df.select(pl.col("ticker")).filter(pl.col("ticker").is\_duplicated()).unique()

raise Personal\_exception(f"Há tickers duplicados:\n{tickers\_duplicateds}")

def check\_columns\_types(self):

str\_columns = self.df.select(pl.col(pl.Utf8)).columns

wrong\_columns = []

for column in str\_columns:

if column not in self.STR\_COLUMNS\_DEFAULT:

wrong\_columns.append(column)

exist\_wrong\_column = len(wrong\_columns) > 0

if exist\_wrong\_column:

error\_msg = "A(s) col(s): " + ", ".join(wrong\_columns)\

+ " está(ão) como número, talvez haja uma celula errada nela(s)"

raise Personal\_exception(error\_msg)

class Runs\_consolidated():

TODAY\_STRING = datetime.today().strftime("%Y-%m-%d")

PATH\_FEATHER = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_" + TODAY\_STRING + ".feather"

def \_\_init\_\_(self, df\_run):

self.df\_run = df\_run

self.read\_file()

self.concatenate\_files()

if not self.df\_runs.is\_empty():

self.duplicated\_rows()

self.write\_concat\_file()

self.consolidate\_files()

def read\_file(self):

self.exist\_previous\_run = True

try:

self.df\_runs = pl.read\_ipc(self.PATH\_FEATHER)

except FileNotFoundError:

self.exist\_previous\_run = False

def concatenate\_files(self):

if self.df\_run.is\_empty():

if not self.exist\_previous\_run:

self.df\_runs = pl.DataFrame()

else:

if self.exist\_previous\_run:

self.df\_runs = pl.concat([self.df\_runs, self.df\_run])

else:

self.df\_runs = self.df\_run

def duplicated\_rows(self):

duplicated = self.df\_runs.filter(pl.lit(self.df\_runs.is\_duplicated())).unique(subset="ticker")

if not duplicated.is\_empty():

error\_msg = (f"Linhas abaixo já existentes no consolidado atual:\n\n{duplicated}")

raise Personal\_exception(error\_msg)

def write\_concat\_file(self):

self.df\_runs = self.df\_runs.sort(by=["ticker", "cliente"])

self.df\_runs.write\_ipc(self.PATH\_FEATHER)

def consolidate\_files(self):

self.df\_runs = (self.df\_runs.with\_columns(

pl.col("bid").filter(pl.col("bid") > 0).min().over("ticker").alias("bid\_min").fill\_null(0),

pl.col("ask").max().over("ticker").alias("ask\_max")

)

.lazy()

.groupby("ticker")

.agg(

pl.col("vol bid").filter((pl.col("bid\_min") == pl.col("bid")) & (pl.col("bid") > 0)).sum().alias("vol\_bid"),

pl.col("bid\_min").first(),

pl.col("ask\_max").first(),

pl.col("vol ask").filter((pl.col("ask\_max") == pl.col("ask")) & (pl.col("ask") > 0)).sum().alias("vol\_ask"),

pl.col("cliente").filter((pl.col("bid\_min") == pl.col("bid")) & (pl.col("bid") > 0)).alias("cliente\_bid"),

pl.col("cliente").filter((pl.col("ask\_max") == pl.col("ask")) & (pl.col("ask") > 0)).alias("cliente\_ask")

)

).collect().with\_columns(

pl.col("cliente\_bid").arr.join(", "),

pl.col("cliente\_ask").arr.join(", "),

).sort("ticker")

class Data\_extra():

def \_\_init\_\_(self, wb):

self.wb = wb

self.obtain\_extras()

pass

def obtain\_extras(self):

self.extras = pl.DataFrame(self.wb.extra\_values, orient = "row")

def kill\_orders(self):

self.extras.filter(pl.col("bid").str.contains("[mM]"))

pass

def convert\_interested\_tickers(self):

pass

class Joined\_info():

SHEET\_CONSOLIDADO = "Consolidado"

RANGE\_PRINT\_CONSOLIDADO = "A3"

COLUMNS\_ORDER = ["ticker", "vol\_bid", "bid\_min", "ask\_max", "vol\_ask", "anbima", "cliente\_bid", "cliente\_ask", "empresa",

"tipo", "indexador", "vencimento", "duration\_anbima", "referencia\_b", "incentivada"]

def \_\_init\_\_(self, database, runs, workbook):

self.df\_db = database

self.df\_runs = runs

self.wb = workbook

self.join\_info()

self.write\_sheet()

self.wb.clean\_lines()

def join\_info(self):

self.df = self.df\_db.join(self.df\_runs, how='outer', on="ticker")

self.df = self.df.select(pl.col(self.COLUMNS\_ORDER))

def write\_sheet(self):

self.wb.wb.sheets[self.SHEET\_CONSOLIDADO].range(self.RANGE\_PRINT\_CONSOLIDADO)\

.options(index=False, header=False).value = self.df.to\_pandas()

class Run\_final():

def \_\_init\_\_(self, df\_extra):

self.extra = df\_extra

self.consolidate\_dfs()

def consolidate\_dfs(self):

pass

def create\_closest\_anbima(self):

pass

try:

today = datetime.today()

yesterday = busday\_offset(today.strftime("%Y-%m-%d"), -1)

yesterday = yesterday.item()

database = Database\_fixed\_income()

df\_db = database.df

anbima = Anbima(yesterday, df\_db)

df\_db = anbima.df

df\_new\_run = pl.DataFrame()

wb = Workbook\_info()

if wb.exist\_new\_run:

new\_run = Data\_new\_run(wb)

df\_new\_run = new\_run.df

wb.exist\_extra = False #por hora, cagar para os extras

if wb.exist\_extra:

extra = Data\_extra(wb)

runs = Runs\_consolidated(df\_run=df\_new\_run)

final\_info = Joined\_info(database=df\_db, runs=runs.df\_runs, workbook=wb)

except Personal\_exception as error:

client = "ARX"

a = pl.read\_ipc(r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_2023-06-01.feather")

a = a.filter(pl.col("cliente") != client)

a.write\_ipc(r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_2023-06-01.feather")

import polars as pl

import pandas as pd

import requests

import polars as pl

from xbbg import blp

class Moodys\_rating():

LINKS = [

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Corporates",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Financial+Institutions",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Fixed+Income+Funds",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Infrastructure+Projects",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Insurance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Structured+Finance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Sub+Sovereign"

]

def \_\_init\_\_(self):

self.create\_dictionary()

self.requests\_links()

def create\_dictionary(self):

self.dictionary\_ratings = {}

self.dictionary\_ratings["company"] = []

self.dictionary\_ratings["date"] = []

self.dictionary\_ratings["rating"] = []

def requests\_links(self):

for link in self.LINKS:

self.r = requests.get(link)

self.build\_dictionary()

def build\_dictionary(self):

for dictionary in r.json():

for key in self.dictionary\_ratings.keys():

self.dictionary\_ratings[key].append(None)

self.dictionary\_ratings["company"][-1] = dictionary["es"]["entity"]

self.dictionary\_ratings["date"][-1] = dictionary["es"]["date"]

if len(dictionary["es"]["instruments"]) == 1:

self.dictionary\_ratings["rating"][-1] = dictionary["es"]["instruments"][0]["ratingValue"]

else:

for rating in dictionary["es"]["instruments"]:

rating\_type = rating["ratingType"]

if rating\_type in self.dictionary\_ratings:

self.dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

else:

self.dictionary\_ratings[rating\_type] = [None]\*len(self.dictionary\_ratings["company"])

self.dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

dictionary\_ratings = {}

dictionary\_ratings["company"] = []

dictionary\_ratings["date"] = []

for dictionary in r.json():

for key in dictionary\_ratings.keys():

dictionary\_ratings[key].append(None)

dictionary\_ratings["company"][-1] = dictionary["es"]["entity"]

dictionary\_ratings["date"][-1] = dictionary["es"]["date"]

for rating in dictionary["es"]["instruments"]:

rating\_type = rating["ratingType"]

if rating\_type in dictionary\_ratings:

dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

else:

dictionary\_ratings[rating\_type] = [None]\*len(dictionary\_ratings["company"])

dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

def str\_to\_list(texto):

return texto[1:-1].replace('},{"es"','},\*\*trocar\*\*{"es"').split("\*\*trocar\*\*")

def procurar(palavra,texto):

tam=len(palavra)

ini=texto.find(palavra)

fim=texto[ini+tam+1:].find('"')

return texto[ini+tam+1:ini+tam+fim+1], texto[ini+tam+fim+1:]

def str\_to\_dict(texto):

dicio={}

dicio['Empresa'],texto=procurar('"entity":',texto)

dicio['Data'],texto=procurar(',"date":',texto)

##Instrumentos - pode ter mais de 1, então separar a lista e transformar em dict talvez?

ini=texto.find('{"ratingValue":')

while ini>-1:

r\_nota,texto=procurar('{"ratingValue":',texto)

if r\_nota!="WR":

r\_nome,texto=procurar('"ratingType":',texto)

dicio[r\_nome]=r\_nota

ini=texto.find('{"ratingValue":')

return dicio

def obter\_ratings\_m(l):

r=requests.get(l)

lista=str\_to\_list(r.text)

df=pd.Series(str\_to\_dict(lista[0]))

for t in lista[1:]:

df=pd.concat([df,pd.Series(str\_to\_dict(t))],axis=1)

df=df.T

return df

links=["https://www.moodyslocal.com/api/countries/BR/ratings?sector=Corporates","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Financial+Institutions",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Fixed+Income+Funds","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Infrastructure+Projects",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Insurance","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Structured+Finance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Sub+Sovereign"]

ratings\_m=obter\_ratings\_m(links[0])

for link in links[1:]:

ratings\_m=pd.concat([ratings\_m,obter\_ratings\_m(link)])

ratings\_m=ratings\_m.dropna(thresh=3) #Retirar os que não possuem rating algum (possivelmente era tudo WR e eu não coloquei)

import polars as pl

from datetime import datetime

from numpy import busday\_offset

import os

class Cra\_db():

'As of 15/may/23, up2data stopped sending security list for CRA and CRI'

CRA\_FIRST\_PATH = r"Y:\20230428\Fixed\_Income\CRA\SecurityList\Fixed\_Income\_FixedIncomeCRAInstrumentFile\_20230427\_2.json"

CRI\_FIRST\_PATH = r"Y:\20230428\Fixed\_Income\CRI\SecurityList\Fixed\_Income\_FixedIncomeCRIInstrumentFile\_20230427\_2.json"

DATABASE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa"

CRA\_DATABASE\_NAME = "db\_cra.parquet"

CRA\_COLUMNS = {

"TckrSymb": "ticker", 'XprtnDt': 'vencimento', 'CrpnNm': 'securitizadora', 'IsseUnitPric': 'pu\_emitido',

'IssdQty': 'quantidade\_emitido', 'IsseTtlVal': 'total\_mercado', "IntrstParamsPctg": "taxa\_emitido",

'InstrmRmnrtnTp': 'indexador', 'ISIN': 'isin', "OTCSgmtNm": "tipo", "InstrmEmssnDt": "data\_emitido",

}

def \_\_init\_\_(self):

pass

#self.update\_info()

def get\_first\_info(self):

files = os.listdir(self.DATABASE\_PATH)

if self.CRA\_DATABASE\_NAME in files:

raise Exception("Já existe db de cra e cri. Deletar na mão em vez de reescrever.")

cra\_df = pl.read\_json(self.CRA\_FIRST\_PATH)

cri\_df = pl.read\_json(self.CRI\_FIRST\_PATH)

self.df = pl.concat([cra\_df, cri\_df])

self.format\_df()

self.df.write\_parquet(self.DATABASE\_PATH + "\\" + self.CRA\_DATABASE\_NAME)

def format\_df(self):

self.df = self.df.rename(self.CRA\_COLUMNS).select(pl.col(self.CRA\_COLUMNS.values()))

for column in ["taxa\_emitido", "pu\_emitido", "quantidade\_emitido", "total\_mercado"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.cast(pl.Float32)

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col("taxa\_emitido")/100,

pl.col(["vencimento", "data\_emitido"]).str.strptime(pl.Date, "%Y-%m-%d"),

(pl.col("quantidade\_emitido")\*pl.col("pu\_emitido")).alias("reais\_emitido"),

pl.lit(None).alias("empresa")

)

self.df = self.df.with\_columns(

pl.when((pl.col("indexador") == "DI") & (pl.col("taxa\_emitido") != 0))

.then(

pl.when(pl.col("taxa\_emitido") < 0.2)

.then("DI +")

.otherwise(

pl.when(pl.col("taxa\_emitido") > 0.9)

.then("% DI")

.otherwise(pl.col("indexador"))

)

)

.otherwise(pl.col("indexador"))

.keep\_name(),

pl.when(pl.col(pl.Date).dt.year() > 2150)

.then(None)

.otherwise(pl.col(pl.Date))

.keep\_name()

)

def write\_cra(self):

pass

#Só escrever se eu conseguir criar algum update para os cris e cras

class Debenture\_db():

DEBENTURE\_FIRST\_PATH = (r"Y:\20230428\Debentures\_MTM\SecurityList\Debentures"

+ "\_MTM\_SecurityListDebentureFileV2\_20230428\_13.json")

DATABASE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa"

DEB\_DATABASE\_NAME = "db\_deb.parquet"

DEBENTURE\_COLUMNS = {

"TckrSymb": "ticker", 'XprtnDt': 'vencimento', 'FxdIntrstRate': 'taxa\_emitido', 'IndxShrtNm': 'indexador',

'CrpnNm': 'empresa', "IsseUnitPric": 'pu\_emitido', 'IssdQty': 'quantidade\_emitido',

'MktQty': 'quantidade\_mercado\_emitido', 'Law12431SpprtInd': 'incentivada', "Clss": "conversivel",

'ISIN': 'isin', "IntrstAcctgIntrvlUnit": "unidade", "InstrmDdlnClclsRef": "critério",

"IntrstSprdDesc":"metodo\_taxa", "IntrstBizDaysClclsRef": "252/360", "IsseDt": "data\_emitido",

"Law12431Artl": "artigo"

}

def \_\_init\_\_(self, date):

self.date = date

self.largest\_file()

self.format\_df()

self.update\_database()

def get\_first\_info(self):

self.df = pl.read\_json(self.DEBENTURE\_FIRST\_PATH)

files = os.listdir(self.DATABASE\_PATH)

if self.DEB\_DATABASE\_NAME in files:

raise Exception("Já existe db de debentures. Deletar na mão em vez de reescrever.")

self.format\_df()

self.df.write\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

def largest\_file(self):

files = os.scandir("Y:\\" + self.date.strftime("%Y%m%d") + r"\Debentures\_MTM\SecurityList")

paths = []

sizes = []

for file in files:

paths.append(file.path)

sizes.append(file.stat().st\_size)

df = pl.DataFrame({"paths": paths, "sizes": sizes})

json\_file = df.filter(pl.col("paths").str.contains(".json")).sort(by="sizes", descending = True)[0, 0]

self.df = pl.read\_json(json\_file)

def format\_df(self):

self.df = self.df.rename(self.DEBENTURE\_COLUMNS).select(self.DEBENTURE\_COLUMNS.values())

for column in ["taxa\_emitido", "pu\_emitido", "quantidade\_emitido", "quantidade\_mercado\_emitido"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.cast(pl.Float32)

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col("taxa\_emitido")/100,

pl.col(["vencimento", "data\_emitido"]).str.strptime(pl.Date, "%Y-%m-%d"),

pl.when(pl.col("artigo") == "")

.then(None)

.otherwise(pl.col("artigo"))

.cast(pl.Int16)

.keep\_name(),

pl.when(pl.col("incentivada") == "true")

.then(True)

.otherwise(False)

.keep\_name(),

(pl.col("quantidade\_emitido")\*pl.col("pu\_emitido")).alias("reais\_emitido"),

(pl.col("quantidade\_mercado\_emitido")\*pl.col("pu\_emitido")).alias("reais\_mercado\_emitido"),

pl.lit("DEB").alias("tipo")

)

self.df = self.df.with\_columns(

pl.when((pl.col("indexador") == "DI") & (pl.col("taxa\_emitido") != 0))

.then(

pl.when(pl.col("taxa\_emitido") < 0.2)

.then("DI +")

.otherwise(

pl.when(pl.col("taxa\_emitido") > 0.9)

.then("% DI")

.otherwise(pl.col("indexador"))

)

)

.otherwise(pl.col("indexador"))

.keep\_name(),

pl.when(pl.col(pl.Date).dt.year() > 2150)

.then(None)

.otherwise(pl.col(pl.Date))

.keep\_name()

)

def update\_database(self):

self.db = pl.read\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

self.db = pl.concat([self.db, self.df]).unique()

self.db.write\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

class Debenture\_trade():

TRADE\_COLUMNS = {"TckrSymb": "ticker", "MinPric": "price\_min", "MaxPric": "price\_max", "TradAvrgPric": "price\_avg",

"TradQty": "trades\_number", "FinInstrmQty": "trades\_quantity", "NtlFinVol": "financial\_volume",

"OprnClssfctnTpNm": "passagem", "EvtAndSttlmGap": "liquidation"}

WRITE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Trades debentures"

def \_\_init\_\_(self, date):

self.date = date

self.largest\_file()

self.format\_df()

self.write\_df()

def largest\_file(self):

files = os.scandir("Y:\\" + self.date.strftime("%Y%m%d") + r"\Debentures\_MTM\TradeInformation")

paths = []

sizes = []

for file in files:

paths.append(file.path)

sizes.append(file.stat().st\_size)

df = pl.DataFrame({"paths": paths, "sizes": sizes})

json\_file = df.filter(pl.col("paths").str.contains(".json")).sort(by="sizes", descending = True)[0, 0]

self.df = pl.read\_json(json\_file)

def format\_df(self):

self.df = self.df.rename(self.TRADE\_COLUMNS).select(self.TRADE\_COLUMNS.values())

for column in ["price\_min", "price\_max", "price\_avg", "financial\_volume", "trades\_number", "trades\_quantity"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col(["price\_min", "price\_max", "price\_avg", "financial\_volume"]).cast(pl.Float32),

pl.col(["trades\_number", "trades\_quantity"]).cast(pl.Int32)

)

def write\_df(self):

self.df.write\_ipc(self.WRITE\_PATH + "\\" + "trades\_deb\_" + self.date.strftime("%Y-%m-%d") + ".feather")

if \_\_name\_\_ == "\_\_main\_\_":

DAYS\_CHECK = 7

today = datetime.today()

for days\_subtract in range(1, DAYS\_CHECK):

date = busday\_offset(datetime.today().strftime("%Y-%m-%d"), (-1)\*days\_subtract)

date = date.item()

date\_string = date.strftime("%Y%m%d")

print(date\_string)

if exist\_up2data\_day(date\_string):

print("1")

if not day\_already\_copied(date):

print("2")

Debenture\_trade(date)

Debenture\_db(date)

class Df\_operations():

WB\_NAME = "RUN Crédito Privado.xlsm"

SHEET\_NAME = "Add RUN"

INFO\_LAST\_LINE = "B4"

RANGE\_DRAFT = "I5:V"

RANGE\_FORMATTED = "C6:G2000"

CELL\_PRINT = RANGE\_FORMATTED[:2]

COLUMNS\_NAME = ["ticker", "vol bid", "bid", "ask", "vol ask"]

def \_\_init\_\_(self):

pass

def obtain\_df(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet = self.wb.sheets[self.SHEET\_NAME]

last\_line = self.sheet.range(self.INFO\_LAST\_LINE).value

my\_date\_handler = lambda year, month, day, \*\*kwargs: f"{month:02d}/{year-2000:02d}"

run\_values = self.sheet.range(self.RANGE\_DRAFT + str(int(last\_line))).options(dates=my\_date\_handler).value

self.df = pl.DataFrame(run\_values[1:], orient = "row")

self.columns\_rename = run\_values[0]

def rename\_columns(self):

dict\_rename\_columns = {}

old\_columns\_names = self.df.columns

new\_columns = []

for index in range(0, len(self.columns\_rename)):

if self.columns\_rename[index] is not None:

new\_columns.append(self.columns\_rename[index])

dict\_rename\_columns[old\_columns\_names[index]] = self.columns\_rename[index]

if ("ticker" not in new\_columns):

raise Personal\_exception("Não há coluna 'ticker'")

if ("ask" not in new\_columns) and ("bid" not in new\_columns):

raise Personal\_exception("Não há coluna 'bid' nem 'ask'")

self.df = self.df.rename(dict\_rename\_columns)

for name in self.COLUMNS\_NAME:

if name not in new\_columns:

self.df = self.df.with\_columns(pl.lit(None).alias(name))

if "vol bid/ask" in new\_columns:

self.df = self.df.with\_columns(

pl.col("vol bid/ask").alias("vol bid"),

pl.col("vol bid/ask").alias("vol ask")

)

exist\_vencimento = "venc" in self.columns\_rename

if exist\_vencimento:

self.df = self.df.select(self.COLUMNS\_NAME + ["venc"])

self.include\_maturity\_lf()

else:

self.df = self.df.select(self.COLUMNS\_NAME)

self.str\_columns = self.df.select(pl.col(pl.Utf8)).columns

def str\_columns\_adjust(self):

self.df = self.df.with\_columns(

pl.col(self.str\_columns)

.str.strip()

.str.to\_uppercase()

.str.replace(",", ".")

).with\_columns(

pl.when(pl.col("ticker") == "")

.then(None)

.otherwise(pl.col("ticker"))

.keep\_name()

)

def include\_maturity\_lf(self):

'''As of may/23, there was no debenture starting with "LF"'''

self.df = self.df.with\_columns(

pl.when(pl.col("ticker").str.contains("^LF"))

.then(pl.col("ticker") + " - " + pl.col("venc"))

.otherwise(pl.col("ticker"))

.alias("ticker")

).select(pl.exclude("venc"))

def rows\_to\_exclude(self):

self.test = self.df

self.df = (self.df

.filter(

pl.any(pl.col(["bid", "ask"]).is\_null().is\_not())

&

pl.col("ticker").is\_null().is\_not()

)

.with\_columns(

pl.col(["vol bid", "bid", "ask", "vol ask"]).fill\_null(0)

)

)

self.df = self.df.filter(

(pl.col("ticker")

.str.contains("ATIVO|TICKER|DEB[ÊE]NTURE|---|INFRA|SPREAD OVER|COMPRO|CPN|CRA$|CRI$").is\_not()

)

&

(pl.all(

~pl.col(["bid", "ask"])

.str.contains("[bB][iI][dD]|[aA][sS][kK]|VALUE|VALOR|##|IPCA|CDI|COMPRA|VENDA|INDIC 1MM"))

)

)

exist\_duplicates = not (self.df.filter(pl.col("ticker").is\_duplicated()).is\_empty())

if exist\_duplicates:

tickers\_duplicateds = self.df.select(pl.col("ticker")).filter(pl.col("ticker").is\_duplicated()).unique()

raise Personal\_exception(f"Há tickers duplicados:\n{tickers\_duplicateds}")

def bid\_ask\_adjust(self):

for column\_name in ["bid", "ask"]:

if column\_name in self.str\_columns:

self.df = self.df.with\_columns(

pl.col(column\_name)

.str.replace("%$|v$", "")

.str.strip()

).with\_columns(

pl.when(pl.col(column\_name).is\_in(["-", ""]))

.then(None)

.otherwise(pl.col(column\_name))

.cast(pl.Float64)

.keep\_name()

)

else:

self.df = self.df.with\_columns(pl.col(column\_name).cast(pl.Float64))

self.df = self.df.filter(

(pl.col("bid") > 0)

| (pl.col("ask") > 0)

)

#Minimum value is considered at the moment 180% CDI. Lower values might confuse with cdi + 1.5%

minimum\_value\_to\_divide = 1.8

divide\_by\_100 = not self.df.filter(pl.any(pl.col(["bid", "ask"]) > minimum\_value\_to\_divide)).is\_empty()

if divide\_by\_100:

#Maximum value is considered at the moment as IPCA + 10%, higher values might confuse with cdi + 0.5%

maximum\_value\_to\_not\_divide = 0.1

not\_divide\_by\_100 = not self.df.filter(

((pl.col("bid") > 0) & (pl.col("bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("ask") > 0) & (pl.col("ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_100:

message = "Alguns valores em bid/ask estão em % e outros não"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["bid", "ask"])/100)

ask\_higher\_bid\_df = (self.df

.filter(

(pl.col("ask") >= pl.col("bid"))

&

(pl.col("bid") != 0)

)

)

ask\_higher\_bid = not ask\_higher\_bid\_df.is\_empty()

if ask\_higher\_bid:

raise Personal\_exception(f"Existe ask maior ou igual a bid\n\n{ask\_higher\_bid\_df}")

def volume\_adjust(self):

self.df\_old = self.df

for column\_name in ["vol bid", "vol ask"]:

if column\_name in self.str\_columns:

self.df = self.df.with\_columns(

pl.when(pl.col(column\_name) == "")

.then(None)

.otherwise(pl.col(column\_name))

.alias(column\_name)

)

self.df = self.df.with\_columns(

pl.col(column\_name)

.str.replace("[mM][lL][nN]|[mM][mM]$|[mM]|[kK]$", "")

.str.replace("^~|^>", "")

.str.strip()

.cast(pl.Float32)

/

pl.when(pl.col("vol ask").str.contains("[kK]$"))

.then(1000)

.otherwise(1)

)

#Minimum value is considered at the moment 100mm. Lower values might confuse with high volume

minimum\_value\_to\_divide = 100

divide\_by\_1000 = not self.df.filter(pl.any(pl.col(["vol bid", "vol ask"]) > minimum\_value\_to\_divide)).is\_empty()

minimum\_value\_to\_divide = 1e5

divide\_by\_million = not self.df.filter(pl.any(pl.col(["vol bid", "vol ask"]) > minimum\_value\_to\_divide)).is\_empty()

if divide\_by\_million:

#Maximum value is considered at the moment as 0.1mm. Higher values might confuse with 100k

maximum\_value\_to\_not\_divide = 1e2

not\_divide\_by\_million = not self.df.filter(

((pl.col("vol bid") > 0) & (pl.col("vol bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("vol ask") > 0) & (pl.col("vol ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_million:

message = "Alguns valores em vol bid/vol ask estão em K e outros em MM"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["vol bid", "vol ask"])/1e6)

elif divide\_by\_1000:

#Maximum value is considered at the moment as 0.1mm. Higher values might confuse with 100k

maximum\_value\_to\_not\_divide = 0.1

not\_divide\_by\_1000 = not self.df.filter(

((pl.col("vol bid") > 0) & (pl.col("vol bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("vol ask") > 0) & (pl.col("vol ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_1000:

message = "Alguns valores em vol bid/vol ask estão em K e outros em MM"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["vol bid", "vol ask"])/1000)

def print\_to\_excel(self):

self.sheet.range(self.RANGE\_FORMATTED).value = ""

self.sheet.range(self.CELL\_PRINT).options(index=False, header=False).value = self.df.to\_pandas()

try:

run = Df\_operations()

run.obtain\_df()

run.rename\_columns()

run.str\_columns\_adjust()

run.rows\_to\_exclude()

run.bid\_ask\_adjust()

run.volume\_adjust()

run.print\_to\_excel()

except Personal\_exception as error:

error\_global = error

print(error)

class Database\_fixed\_income():

DEB\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa\db\_deb.parquet"

CRA\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa\db\_cra.parquet"

COLUMNS = ["ticker", "vencimento", "taxa\_emitido", "indexador", "empresa", "pu\_emitido",

"isin", "data\_emitido", "reais\_emitido", "tipo", "incentivada", "artigo"]

def \_\_init\_\_(self):

self.get\_deb\_database()

self.get\_cra\_database()

self.concat\_databases()

def get\_deb\_database(self):

self.df\_deb = pl.read\_parquet(self.DEB\_PATH).select(pl.col(self.COLUMNS))

def get\_cra\_database(self):

self.df\_cra = pl.read\_parquet(self.CRA\_PATH)

self.df\_cra = self.df\_cra.with\_columns(

pl.lit(True).alias("incentivada"),

pl.lit(None).cast(pl.Int16).alias("artigo")

)

self.df\_cra = self.df\_cra.select(pl.col(self.COLUMNS))

def concat\_databases(self):

self.df = pl.concat([self.df\_deb, self.df\_cra])

class Anbima():

PATH\_ANBIMA\_DEB = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Debêntures\anbima\_deb\_"

PATH\_ANBIMA\_CRA = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Cra\anbima\_cra\_"

PATH\_ANBIMA\_TIT = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Títulos públicos\anbima\_tit\_"

COLUMNS\_CRA = ["ticker", "anbima", "referencia\_b", "duration\_anbima"]

def \_\_init\_\_(self, yesterday, df):

self.yesterday = yesterday

self.yesterday\_string = yesterday.strftime("%Y-%m-%d")

self.df = df

self.get\_anbima\_deb()

self.get\_anbima\_cra()

self.get\_anbima\_tit()

def get\_anbima\_deb(self):

while True:

try:

deb = pl.read\_ipc(self.PATH\_ANBIMA\_DEB + self.yesterday\_string + ".feather")

break

except FileNotFoundError:

self.check\_last\_anbima()

self.deb = deb

self.df = self.df.join(deb, how="left", on="ticker")

def check\_last\_anbima(self):

import requests

URL = 'https://www.anbima.com.br/informacoes/merc-sec-debentures/arqs/db'

r = requests.get(URL + self.yesterday.strftime("%y%m%d") + ".txt")

file\_found = (r.status\_code == 200)

if file\_found:

raise Personal\_exception(f"Último arquivo da anbima {self.yesterday\_string} não baixado.")

else:

self.yesterday = busday\_offset(self.yesterday\_string, -1)

self.yesterday = self.yesterday.item()

self.yesterday\_string = self.yesterday.strftime("%Y-%m-%d")

def get\_anbima\_cra(self):

cra = pl.read\_ipc(self.PATH\_ANBIMA\_CRA + self.yesterday\_string + ".feather")

cra = cra.select(pl.col(self.COLUMNS\_CRA))

self.cra = cra

self.df = self.df.join(cra, how="left", on="ticker", suffix="\_right")

for column in ["anbima", "referencia\_b", "duration\_anbima"]:

self.df = self.df.with\_columns(

pl.when((pl.col(column).is\_null()) & (~pl.col(column + "\_right").is\_null()))

.then(pl.col(column + "\_right"))

.otherwise(pl.col(column))

.alias(column)

).drop(column + "\_right")

def get\_anbima\_tit(self):

tit = pl.read\_ipc(self.PATH\_ANBIMA\_TIT + self.yesterday\_string + ".feather")

tit = tit.with\_columns(

("TIT: " + pl.col("ticker") + " " + pl.col("vencimento").dt.strftime("%y")).alias("ticker")

)

self.tit = tit

columns = []

for column in self.df.columns:

if column not in self.tit.columns:

columns.append(column)

self.tit = self.tit.with\_columns(pl.lit(None).alias(column))

self.tit = self.tit.select(pl.col(self.df.columns))

self.tit = self.tit.with\_columns(

pl.col(["pu\_emitido", "taxa\_emitido", "reais\_emitido", "duration\_anbima"])

.cast(pl.Float32),

pl.col("data\_emitido").cast(pl.Date),

pl.lit("Título público").alias("tipo"),

pl.col(["indexador", "isin", "empresa", "referencia\_b"]).cast(pl.Utf8),

pl.col("incentivada").cast(pl.Boolean),

pl.col("artigo").cast(pl.Int16)

)

self.df = pl.concat([self.tit, self.df])

class Workbook\_info():

WB\_NAME = "RUN Crédito Privado.xlsm"

SHEET\_NEW\_RUN = "Add RUN"

SHEET\_EXTRA = "RUN"

RANGE\_MAX\_LINE = "B2:B3"

RANGE\_NEW\_RUN = "C6:G"

RANGE\_EXTRA = "A18:F"

LAST\_LINE\_DELETE = "2000"

CLIENT\_CELL = "C4"

RANGE\_NEW\_RUN\_DELETE = RANGE\_NEW\_RUN + LAST\_LINE\_DELETE

RANGE\_FORMATTED = "H4:L" + LAST\_LINE\_DELETE

RANGE\_DRAFT = "I5:V" + LAST\_LINE\_DELETE

def \_\_init\_\_(self):

self.obtain\_sheets()

self.obtain\_new\_run()

self.obtain\_extras()

def obtain\_sheets(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet\_new\_run = self.wb.sheets[self.SHEET\_NEW\_RUN]

self.sheet\_extra = self.wb.sheets[self.SHEET\_EXTRA]

self.lines = self.sheet\_new\_run.range(self.RANGE\_MAX\_LINE).value

def obtain\_new\_run(self):

max\_line\_new\_run = str(int(self.lines[1]))

self.exist\_new\_run = self.lines[1] > 5

if self.exist\_new\_run:

self.new\_run = self.sheet\_new\_run.range(self.RANGE\_NEW\_RUN + max\_line\_new\_run).value

self.client = self.sheet\_new\_run.range(self.CLIENT\_CELL).value

if self.client is None:

raise Personal\_exception("Cliente está vazio")

else:

self.client = self.client.upper()

def obtain\_extras(self):

line\_extra = str(int(self.lines[0]))

self.exist\_extra = self.lines[0] > 17

if self.exist\_extra:

self.extra\_values = self.sheet\_extra.range(self.RANGE\_EXTRA + line\_extra).value

def clean\_lines(self):

cells\_to\_delete = [self.CLIENT\_CELL, self.RANGE\_NEW\_RUN\_DELETE, self.RANGE\_DRAFT]

cells\_to\_delete = ", ".join(cells\_to\_delete)

self.sheet\_new\_run.range(cells\_to\_delete).value = ""

class Data\_new\_run():

NEW\_COLUMNS\_NAMES = ["ticker", "vol bid", "bid", "ask", "vol ask", "cliente"]

FLOAT\_COLUMNS\_DEFAULT = NEW\_COLUMNS\_NAMES[1:-1]

STR\_COLUMNS\_DEFAULT = [NEW\_COLUMNS\_NAMES[0], NEW\_COLUMNS\_NAMES[-1]]

def \_\_init\_\_(self, wb):

self.wb = wb

self.obtain\_df()

self.rename\_columns()

self.check\_duplicates()

self.check\_columns\_types()

def obtain\_df(self):

empty\_client = (self.wb.new\_run[0][0] == None)

df = pl.DataFrame(self.wb.new\_run, orient = "row")

self.df = df.with\_columns(pl.lit(self.wb.client).alias("cliente"))

def rename\_columns(self):

dict\_rename\_columns = {}

old\_columns\_names = self.df.columns

for index in range(0, 6):

dict\_rename\_columns[old\_columns\_names[index]] = self.NEW\_COLUMNS\_NAMES[index]

self.df = self.df.rename(dict\_rename\_columns).with\_columns(pl.col(pl.Boolean).cast(pl.Float64))

def check\_duplicates(self):

exist\_duplicates = not (self.df.filter(pl.col("ticker").is\_duplicated()).is\_empty())

if exist\_duplicates:

tickers\_duplicateds = self.df.select(pl.col("ticker")).filter(pl.col("ticker").is\_duplicated()).unique()

raise Personal\_exception(f"Há tickers duplicados:\n{tickers\_duplicateds}")

def check\_columns\_types(self):

str\_columns = self.df.select(pl.col(pl.Utf8)).columns

wrong\_columns = []

for column in str\_columns:

if column not in self.STR\_COLUMNS\_DEFAULT:

wrong\_columns.append(column)

exist\_wrong\_column = len(wrong\_columns) > 0

if exist\_wrong\_column:

error\_msg = "A(s) col(s): " + ", ".join(wrong\_columns)\

+ " está(ão) como número, talvez haja uma celula errada nela(s)"

raise Personal\_exception(error\_msg)

class Runs\_consolidated():

TODAY\_STRING = datetime.today().strftime("%Y-%m-%d")

PATH\_FEATHER = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_" + TODAY\_STRING + ".feather"

def \_\_init\_\_(self, df\_run):

self.df\_run = df\_run

self.read\_file()

self.concatenate\_files()

if not self.df\_runs.is\_empty():

self.duplicated\_rows()

self.write\_concat\_file()

self.consolidate\_files()

def read\_file(self):

self.exist\_previous\_run = True

try:

self.df\_runs = pl.read\_ipc(self.PATH\_FEATHER)

except FileNotFoundError:

self.exist\_previous\_run = False

def concatenate\_files(self):

if self.df\_run.is\_empty():

if not self.exist\_previous\_run:

self.df\_runs = pl.DataFrame()

else:

if self.exist\_previous\_run:

self.df\_runs = pl.concat([self.df\_runs, self.df\_run])

else:

self.df\_runs = self.df\_run

def duplicated\_rows(self):

duplicated = self.df\_runs.filter(pl.lit(self.df\_runs.is\_duplicated())).unique(subset="ticker")

if not duplicated.is\_empty():

error\_msg = (f"Linhas abaixo já existentes no consolidado atual:\n\n{duplicated}")

raise Personal\_exception(error\_msg)

def write\_concat\_file(self):

self.df\_runs = self.df\_runs.sort(by=["ticker", "cliente"])

self.df\_runs.write\_ipc(self.PATH\_FEATHER)

def consolidate\_files(self):

self.df\_runs = (self.df\_runs.with\_columns(

pl.col("bid").filter(pl.col("bid") > 0).min().over("ticker").alias("bid\_min").fill\_null(0),

pl.col("ask").max().over("ticker").alias("ask\_max")

)

.lazy()

.groupby("ticker")

.agg(

pl.col("vol bid").filter((pl.col("bid\_min") == pl.col("bid")) & (pl.col("bid") > 0)).sum().alias("vol\_bid"),

pl.col("bid\_min").first(),

pl.col("ask\_max").first(),

pl.col("vol ask").filter((pl.col("ask\_max") == pl.col("ask")) & (pl.col("ask") > 0)).sum().alias("vol\_ask"),

pl.col("cliente").filter((pl.col("bid\_min") == pl.col("bid")) & (pl.col("bid") > 0)).alias("cliente\_bid"),

pl.col("cliente").filter((pl.col("ask\_max") == pl.col("ask")) & (pl.col("ask") > 0)).alias("cliente\_ask")

)

).collect().with\_columns(

pl.col("cliente\_bid").arr.join(", "),

pl.col("cliente\_ask").arr.join(", "),

).sort("ticker")

class Data\_extra():

def \_\_init\_\_(self, wb):

self.wb = wb

self.obtain\_extras()

pass

def obtain\_extras(self):

self.extras = pl.DataFrame(self.wb.extra\_values, orient = "row")

def kill\_orders(self):

self.extras.filter(pl.col("bid").str.contains("[mM]"))

pass

def convert\_interested\_tickers(self):

pass

class Joined\_info():

SHEET\_CONSOLIDADO = "Consolidado"

RANGE\_PRINT\_CONSOLIDADO = "A3"

COLUMNS\_ORDER = ["ticker", "vol\_bid", "bid\_min", "ask\_max", "vol\_ask", "anbima", "cliente\_bid", "cliente\_ask", "empresa",

"tipo", "indexador", "vencimento", "duration\_anbima", "referencia\_b", "incentivada"]

def \_\_init\_\_(self, database, runs, workbook):

self.df\_db = database

self.df\_runs = runs

self.wb = workbook

self.join\_info()

self.write\_sheet()

self.wb.clean\_lines()

def join\_info(self):

self.df = self.df\_db.join(self.df\_runs, how='outer', on="ticker")

self.df = self.df.select(pl.col(self.COLUMNS\_ORDER))

def write\_sheet(self):

self.wb.wb.sheets[self.SHEET\_CONSOLIDADO].range(self.RANGE\_PRINT\_CONSOLIDADO)\

.options(index=False, header=False).value = self.df.to\_pandas()

class Run\_final():

def \_\_init\_\_(self, df\_extra):

self.extra = df\_extra

self.consolidate\_dfs()

def consolidate\_dfs(self):

pass

def create\_closest\_anbima(self):

pass

try:

today = datetime.today()

yesterday = busday\_offset(today.strftime("%Y-%m-%d"), -1)

yesterday = yesterday.item()

database = Database\_fixed\_income()

df\_db = database.df

anbima = Anbima(yesterday, df\_db)

df\_db = anbima.df

df\_new\_run = pl.DataFrame()

wb = Workbook\_info()

if wb.exist\_new\_run:

new\_run = Data\_new\_run(wb)

df\_new\_run = new\_run.df

wb.exist\_extra = False #por hora, cagar para os extras

if wb.exist\_extra:

extra = Data\_extra(wb)

runs = Runs\_consolidated(df\_run=df\_new\_run)

final\_info = Joined\_info(database=df\_db, runs=runs.df\_runs, workbook=wb)

except Personal\_exception as error:

error\_global = error

client = "ARX"

a = pl.read\_ipc(r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_2023-06-01.feather")

a = a.filter(pl.col("cliente") != client)

a.write\_ipc(r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_2023-06-01.feather")

import polars as pl

import pandas as pd

import requests

import polars as pl

from xbbg import blp

class Moodys\_rating():

LINKS = [

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Corporates",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Financial+Institutions",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Fixed+Income+Funds",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Infrastructure+Projects",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Insurance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Structured+Finance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Sub+Sovereign"

]

def \_\_init\_\_(self):

self.create\_dictionary()

self.requests\_links()

def create\_dictionary(self):

self.dictionary\_ratings = {}

self.dictionary\_ratings["company"] = []

self.dictionary\_ratings["date"] = []

self.dictionary\_ratings["rating"] = []

def requests\_links(self):

for link in self.LINKS:

self.r = requests.get(link)

self.build\_dictionary()

def build\_dictionary(self):

for dictionary in r.json():

for key in self.dictionary\_ratings.keys():

self.dictionary\_ratings[key].append(None)

self.dictionary\_ratings["company"][-1] = dictionary["es"]["entity"]

self.dictionary\_ratings["date"][-1] = dictionary["es"]["date"]

if len(dictionary["es"]["instruments"]) == 1:

self.dictionary\_ratings["rating"][-1] = dictionary["es"]["instruments"][0]["ratingValue"]

else:

for rating in dictionary["es"]["instruments"]:

rating\_type = rating["ratingType"]

if rating\_type in self.dictionary\_ratings:

self.dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

else:

self.dictionary\_ratings[rating\_type] = [None]\*len(self.dictionary\_ratings["company"])

self.dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

dictionary\_ratings = {}

dictionary\_ratings["company"] = []

dictionary\_ratings["date"] = []

for dictionary in r.json():

for key in dictionary\_ratings.keys():

dictionary\_ratings[key].append(None)

dictionary\_ratings["company"][-1] = dictionary["es"]["entity"]

dictionary\_ratings["date"][-1] = dictionary["es"]["date"]

for rating in dictionary["es"]["instruments"]:

rating\_type = rating["ratingType"]

if rating\_type in dictionary\_ratings:

dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

else:

dictionary\_ratings[rating\_type] = [None]\*len(dictionary\_ratings["company"])

dictionary\_ratings[rating\_type][-1] = rating["ratingValue"]

def str\_to\_list(texto):

return texto[1:-1].replace('},{"es"','},\*\*trocar\*\*{"es"').split("\*\*trocar\*\*")

def procurar(palavra,texto):

tam=len(palavra)

ini=texto.find(palavra)

fim=texto[ini+tam+1:].find('"')

return texto[ini+tam+1:ini+tam+fim+1], texto[ini+tam+fim+1:]

def str\_to\_dict(texto):

dicio={}

dicio['Empresa'],texto=procurar('"entity":',texto)

dicio['Data'],texto=procurar(',"date":',texto)

##Instrumentos - pode ter mais de 1, então separar a lista e transformar em dict talvez?

ini=texto.find('{"ratingValue":')

while ini>-1:

r\_nota,texto=procurar('{"ratingValue":',texto)

if r\_nota!="WR":

r\_nome,texto=procurar('"ratingType":',texto)

dicio[r\_nome]=r\_nota

ini=texto.find('{"ratingValue":')

return dicio

def obter\_ratings\_m(l):

r=requests.get(l)

lista=str\_to\_list(r.text)

df=pd.Series(str\_to\_dict(lista[0]))

for t in lista[1:]:

df=pd.concat([df,pd.Series(str\_to\_dict(t))],axis=1)

df=df.T

return df

links=["https://www.moodyslocal.com/api/countries/BR/ratings?sector=Corporates","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Financial+Institutions",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Fixed+Income+Funds","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Infrastructure+Projects",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Insurance","https://www.moodyslocal.com/api/countries/BR/ratings?sector=Structured+Finance",

"https://www.moodyslocal.com/api/countries/BR/ratings?sector=Sub+Sovereign"]

ratings\_m=obter\_ratings\_m(links[0])

for link in links[1:]:

ratings\_m=pd.concat([ratings\_m,obter\_ratings\_m(link)])

ratings\_m=ratings\_m.dropna(thresh=3) #Retirar os que não possuem rating algum (possivelmente era tudo WR e eu não coloquei)

import polars as pl

from datetime import datetime

from numpy import busday\_offset

import os

class Cra\_db():

'As of 15/may/23, up2data stopped sending security list for CRA and CRI'

CRA\_FIRST\_PATH = r"Y:\20230428\Fixed\_Income\CRA\SecurityList\Fixed\_Income\_FixedIncomeCRAInstrumentFile\_20230427\_2.json"

CRI\_FIRST\_PATH = r"Y:\20230428\Fixed\_Income\CRI\SecurityList\Fixed\_Income\_FixedIncomeCRIInstrumentFile\_20230427\_2.json"

DATABASE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa"

CRA\_DATABASE\_NAME = "db\_cra.parquet"

CRA\_COLUMNS = {

"TckrSymb": "ticker", 'XprtnDt': 'vencimento', 'CrpnNm': 'securitizadora', 'IsseUnitPric': 'pu\_emitido',

'IssdQty': 'quantidade\_emitido', 'IsseTtlVal': 'total\_mercado', "IntrstParamsPctg": "taxa\_emitido",

'InstrmRmnrtnTp': 'indexador', 'ISIN': 'isin', "OTCSgmtNm": "tipo", "InstrmEmssnDt": "data\_emitido",

}

def \_\_init\_\_(self):

pass

#self.update\_info()

def get\_first\_info(self):

files = os.listdir(self.DATABASE\_PATH)

if self.CRA\_DATABASE\_NAME in files:

raise Exception("Já existe db de cra e cri. Deletar na mão em vez de reescrever.")

cra\_df = pl.read\_json(self.CRA\_FIRST\_PATH)

cri\_df = pl.read\_json(self.CRI\_FIRST\_PATH)

self.df = pl.concat([cra\_df, cri\_df])

self.format\_df()

self.df.write\_parquet(self.DATABASE\_PATH + "\\" + self.CRA\_DATABASE\_NAME)

def format\_df(self):

self.df = self.df.rename(self.CRA\_COLUMNS).select(pl.col(self.CRA\_COLUMNS.values()))

for column in ["taxa\_emitido", "pu\_emitido", "quantidade\_emitido", "total\_mercado"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.cast(pl.Float32)

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col("taxa\_emitido")/100,

pl.col(["vencimento", "data\_emitido"]).str.strptime(pl.Date, "%Y-%m-%d"),

(pl.col("quantidade\_emitido")\*pl.col("pu\_emitido")).alias("reais\_emitido"),

pl.lit(None).alias("empresa")

)

self.df = self.df.with\_columns(

pl.when((pl.col("indexador") == "DI") & (pl.col("taxa\_emitido") != 0))

.then(

pl.when(pl.col("taxa\_emitido") < 0.2)

.then("DI +")

.otherwise(

pl.when(pl.col("taxa\_emitido") > 0.9)

.then("% DI")

.otherwise(pl.col("indexador"))

)

)

.otherwise(pl.col("indexador"))

.keep\_name(),

pl.when(pl.col(pl.Date).dt.year() > 2150)

.then(None)

.otherwise(pl.col(pl.Date))

.keep\_name()

)

def write\_cra(self):

pass

#Só escrever se eu conseguir criar algum update para os cris e cras

class Debenture\_db():

DEBENTURE\_FIRST\_PATH = (r"Y:\20230428\Debentures\_MTM\SecurityList\Debentures"

+ "\_MTM\_SecurityListDebentureFileV2\_20230428\_13.json")

DATABASE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa"

DEB\_DATABASE\_NAME = "db\_deb.parquet"

DEBENTURE\_COLUMNS = {

"TckrSymb": "ticker", 'XprtnDt': 'vencimento', 'FxdIntrstRate': 'taxa\_emitido', 'IndxShrtNm': 'indexador',

'CrpnNm': 'empresa', "IsseUnitPric": 'pu\_emitido', 'IssdQty': 'quantidade\_emitido',

'MktQty': 'quantidade\_mercado\_emitido', 'Law12431SpprtInd': 'incentivada', "Clss": "conversivel",

'ISIN': 'isin', "IntrstAcctgIntrvlUnit": "unidade", "InstrmDdlnClclsRef": "critério",

"IntrstSprdDesc":"metodo\_taxa", "IntrstBizDaysClclsRef": "252/360", "IsseDt": "data\_emitido",

"Law12431Artl": "artigo"

}

def \_\_init\_\_(self, date):

self.date = date

self.largest\_file()

self.format\_df()

self.update\_database()

def get\_first\_info(self):

self.df = pl.read\_json(self.DEBENTURE\_FIRST\_PATH)

files = os.listdir(self.DATABASE\_PATH)

if self.DEB\_DATABASE\_NAME in files:

raise Exception("Já existe db de debentures. Deletar na mão em vez de reescrever.")

self.format\_df()

self.df.write\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

def largest\_file(self):

files = os.scandir("Y:\\" + self.date.strftime("%Y%m%d") + r"\Debentures\_MTM\SecurityList")

paths = []

sizes = []

for file in files:

paths.append(file.path)

sizes.append(file.stat().st\_size)

df = pl.DataFrame({"paths": paths, "sizes": sizes})

json\_file = df.filter(pl.col("paths").str.contains(".json")).sort(by="sizes", descending = True)[0, 0]

self.df = pl.read\_json(json\_file)

def format\_df(self):

self.df = self.df.rename(self.DEBENTURE\_COLUMNS).select(self.DEBENTURE\_COLUMNS.values())

for column in ["taxa\_emitido", "pu\_emitido", "quantidade\_emitido", "quantidade\_mercado\_emitido"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.cast(pl.Float32)

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col("taxa\_emitido")/100,

pl.col(["vencimento", "data\_emitido"]).str.strptime(pl.Date, "%Y-%m-%d"),

pl.when(pl.col("artigo") == "")

.then(None)

.otherwise(pl.col("artigo"))

.cast(pl.Int16)

.keep\_name(),

pl.when(pl.col("incentivada") == "true")

.then(True)

.otherwise(False)

.keep\_name(),

(pl.col("quantidade\_emitido")\*pl.col("pu\_emitido")).alias("reais\_emitido"),

(pl.col("quantidade\_mercado\_emitido")\*pl.col("pu\_emitido")).alias("reais\_mercado\_emitido"),

pl.lit("DEB").alias("tipo")

)

self.df = self.df.with\_columns(

pl.when((pl.col("indexador") == "DI") & (pl.col("taxa\_emitido") != 0))

.then(

pl.when(pl.col("taxa\_emitido") < 0.2)

.then("DI +")

.otherwise(

pl.when(pl.col("taxa\_emitido") > 0.9)

.then("% DI")

.otherwise(pl.col("indexador"))

)

)

.otherwise(pl.col("indexador"))

.keep\_name(),

pl.when(pl.col(pl.Date).dt.year() > 2150)

.then(None)

.otherwise(pl.col(pl.Date))

.keep\_name()

)

def update\_database(self):

self.db = pl.read\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

self.db = pl.concat([self.db, self.df]).unique()

self.db.write\_parquet(self.DATABASE\_PATH + "\\" + self.DEB\_DATABASE\_NAME)

class Debenture\_trade():

TRADE\_COLUMNS = {"TckrSymb": "ticker", "MinPric": "price\_min", "MaxPric": "price\_max", "TradAvrgPric": "price\_avg",

"TradQty": "trades\_number", "FinInstrmQty": "trades\_quantity", "NtlFinVol": "financial\_volume",

"OprnClssfctnTpNm": "passagem", "EvtAndSttlmGap": "liquidation"}

WRITE\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Trades debentures"

def \_\_init\_\_(self, date):

self.date = date

self.largest\_file()

self.format\_df()

self.write\_df()

def largest\_file(self):

files = os.scandir("Y:\\" + self.date.strftime("%Y%m%d") + r"\Debentures\_MTM\TradeInformation")

paths = []

sizes = []

for file in files:

paths.append(file.path)

sizes.append(file.stat().st\_size)

df = pl.DataFrame({"paths": paths, "sizes": sizes})

json\_file = df.filter(pl.col("paths").str.contains(".json")).sort(by="sizes", descending = True)[0, 0]

self.df = pl.read\_json(json\_file)

def format\_df(self):

self.df = self.df.rename(self.TRADE\_COLUMNS).select(self.TRADE\_COLUMNS.values())

for column in ["price\_min", "price\_max", "price\_avg", "financial\_volume", "trades\_number", "trades\_quantity"]:

self.df = self.df.with\_columns(

pl.when(pl.col(column) == "")

.then(None)

.otherwise(pl.col(column))

.keep\_name()

)

self.df = self.df.with\_columns(

pl.col(["price\_min", "price\_max", "price\_avg", "financial\_volume"]).cast(pl.Float32),

pl.col(["trades\_number", "trades\_quantity"]).cast(pl.Int32)

)

def write\_df(self):

self.df.write\_ipc(self.WRITE\_PATH + "\\" + "trades\_deb\_" + self.date.strftime("%Y-%m-%d") + ".feather")

if \_\_name\_\_ == "\_\_main\_\_":

DAYS\_CHECK = 7

today = datetime.today()

for days\_subtract in range(1, DAYS\_CHECK):

date = busday\_offset(datetime.today().strftime("%Y-%m-%d"), (-1)\*days\_subtract)

date = date.item()

date\_string = date.strftime("%Y%m%d")

print(date\_string)

if exist\_up2data\_day(date\_string):

print("1")

if not day\_already\_copied(date):

print("2")

Debenture\_trade(date)

Debenture\_db(date)

class Df\_operations():

WB\_NAME = "RUN Crédito Privado.xlsm"

SHEET\_NAME = "Add RUN"

INFO\_LAST\_LINE = "B4"

RANGE\_DRAFT = "I5:V"

RANGE\_FORMATTED = "C6:G2000"

CELL\_PRINT = RANGE\_FORMATTED[:2]

COLUMNS\_NAME = ["ticker", "vol bid", "bid", "ask", "vol ask"]

def \_\_init\_\_(self):

pass

def obtain\_df(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet = self.wb.sheets[self.SHEET\_NAME]

last\_line = self.sheet.range(self.INFO\_LAST\_LINE).value

my\_date\_handler = lambda year, month, day, \*\*kwargs: f"{month:02d}/{year-2000:02d}"

run\_values = self.sheet.range(self.RANGE\_DRAFT + str(int(last\_line))).options(dates=my\_date\_handler).value

self.df = pl.DataFrame(run\_values[1:], orient = "row")

self.columns\_rename = run\_values[0]

def rename\_columns(self):

dict\_rename\_columns = {}

old\_columns\_names = self.df.columns

new\_columns = []

for index in range(0, len(self.columns\_rename)):

if self.columns\_rename[index] is not None:

new\_columns.append(self.columns\_rename[index])

dict\_rename\_columns[old\_columns\_names[index]] = self.columns\_rename[index]

if ("ticker" not in new\_columns):

raise Personal\_exception("Não há coluna 'ticker'")

if ("ask" not in new\_columns) and ("bid" not in new\_columns):

raise Personal\_exception("Não há coluna 'bid' nem 'ask'")

self.df = self.df.rename(dict\_rename\_columns)

for name in self.COLUMNS\_NAME:

if name not in new\_columns:

self.df = self.df.with\_columns(pl.lit(None).alias(name))

if "vol bid/ask" in new\_columns:

self.df = self.df.with\_columns(

pl.col("vol bid/ask").alias("vol bid"),

pl.col("vol bid/ask").alias("vol ask")

)

exist\_vencimento = "venc" in self.columns\_rename

if exist\_vencimento:

self.df = self.df.select(self.COLUMNS\_NAME + ["venc"])

self.include\_maturity\_lf()

else:

self.df = self.df.select(self.COLUMNS\_NAME)

self.str\_columns = self.df.select(pl.col(pl.Utf8)).columns

def str\_columns\_adjust(self):

self.df = self.df.with\_columns(

pl.col(self.str\_columns)

.str.strip()

.str.to\_uppercase()

.str.replace(",", ".")

).with\_columns(

pl.when(pl.col("ticker") == "")

.then(None)

.otherwise(pl.col("ticker"))

.keep\_name()

)

def include\_maturity\_lf(self):

'''As of may/23, there was no debenture starting with "LF"'''

self.df = self.df.with\_columns(

pl.when(pl.col("ticker").str.contains("^LF"))

.then(pl.col("ticker") + " - " + pl.col("venc"))

.otherwise(pl.col("ticker"))

.alias("ticker")

).select(pl.exclude("venc"))

def rows\_to\_exclude(self):

self.test = self.df

self.df = (self.df

.filter(

pl.any(pl.col(["bid", "ask"]).is\_null().is\_not())

&

pl.col("ticker").is\_null().is\_not()

)

.with\_columns(

pl.col(["vol bid", "bid", "ask", "vol ask"]).fill\_null(0)

)

)

self.df = self.df.filter(

(pl.col("ticker")

.str.contains("ATIVO|TICKER|DEB[ÊE]NTURE|---|INFRA|SPREAD OVER|COMPRO|CPN|CRA$|CRI$").is\_not()

)

&

(pl.all(

~pl.col(["bid", "ask"])

.str.contains("[bB][iI][dD]|[aA][sS][kK]|VALUE|VALOR|##|IPCA|CDI|COMPRA|VENDA|INDIC 1MM"))

)

)

exist\_duplicates = not (self.df.filter(pl.col("ticker").is\_duplicated()).is\_empty())

if exist\_duplicates:

tickers\_duplicateds = self.df.select(pl.col("ticker")).filter(pl.col("ticker").is\_duplicated()).unique()

raise Personal\_exception(f"Há tickers duplicados:\n{tickers\_duplicateds}")

def bid\_ask\_adjust(self):

for column\_name in ["bid", "ask"]:

if column\_name in self.str\_columns:

self.df = self.df.with\_columns(

pl.col(column\_name)

.str.replace("%$|v$", "")

.str.strip()

).with\_columns(

pl.when(pl.col(column\_name).is\_in(["-", ""]))

.then(None)

.otherwise(pl.col(column\_name))

.cast(pl.Float64)

.keep\_name()

)

else:

self.df = self.df.with\_columns(pl.col(column\_name).cast(pl.Float64))

self.df = self.df.filter(

(pl.col("bid") > 0)

| (pl.col("ask") > 0)

)

#Minimum value is considered at the moment 180% CDI. Lower values might confuse with cdi + 1.5%

minimum\_value\_to\_divide = 1.8

divide\_by\_100 = not self.df.filter(pl.any(pl.col(["bid", "ask"]) > minimum\_value\_to\_divide)).is\_empty()

if divide\_by\_100:

#Maximum value is considered at the moment as IPCA + 10%, higher values might confuse with cdi + 0.5%

maximum\_value\_to\_not\_divide = 0.1

not\_divide\_by\_100 = not self.df.filter(

((pl.col("bid") > 0) & (pl.col("bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("ask") > 0) & (pl.col("ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_100:

message = "Alguns valores em bid/ask estão em % e outros não"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["bid", "ask"])/100)

ask\_higher\_bid\_df = (self.df

.filter(

(pl.col("ask") >= pl.col("bid"))

&

(pl.col("bid") != 0)

)

)

ask\_higher\_bid = not ask\_higher\_bid\_df.is\_empty()

if ask\_higher\_bid:

raise Personal\_exception(f"Existe ask maior ou igual a bid\n\n{ask\_higher\_bid\_df}")

def volume\_adjust(self):

self.df\_old = self.df

for column\_name in ["vol bid", "vol ask"]:

if column\_name in self.str\_columns:

self.df = self.df.with\_columns(

pl.when(pl.col(column\_name) == "")

.then(None)

.otherwise(pl.col(column\_name))

.alias(column\_name)

)

self.df = self.df.with\_columns(

pl.col(column\_name)

.str.replace("[mM][lL][nN]|[mM][mM]$|[mM]|[kK]$", "")

.str.replace("^~|^>", "")

.str.strip()

.cast(pl.Float32)

/

pl.when(pl.col("vol ask").str.contains("[kK]$"))

.then(1000)

.otherwise(1)

)

#Minimum value is considered at the moment 100mm. Lower values might confuse with high volume

minimum\_value\_to\_divide = 100

divide\_by\_1000 = not self.df.filter(pl.any(pl.col(["vol bid", "vol ask"]) > minimum\_value\_to\_divide)).is\_empty()

minimum\_value\_to\_divide = 1e5

divide\_by\_million = not self.df.filter(pl.any(pl.col(["vol bid", "vol ask"]) > minimum\_value\_to\_divide)).is\_empty()

if divide\_by\_million:

#Maximum value is considered at the moment as 0.1mm. Higher values might confuse with 100k

maximum\_value\_to\_not\_divide = 1e2

not\_divide\_by\_million = not self.df.filter(

((pl.col("vol bid") > 0) & (pl.col("vol bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("vol ask") > 0) & (pl.col("vol ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_million:

message = "Alguns valores em vol bid/vol ask estão em K e outros em MM"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["vol bid", "vol ask"])/1e6)

elif divide\_by\_1000:

#Maximum value is considered at the moment as 0.1mm. Higher values might confuse with 100k

maximum\_value\_to\_not\_divide = 0.1

not\_divide\_by\_1000 = not self.df.filter(

((pl.col("vol bid") > 0) & (pl.col("vol bid") < maximum\_value\_to\_not\_divide))

|

((pl.col("vol ask") > 0) & (pl.col("vol ask") < maximum\_value\_to\_not\_divide))

).is\_empty()

if not\_divide\_by\_1000:

message = "Alguns valores em vol bid/vol ask estão em K e outros em MM"

raise Personal\_exception(message)

else:

self.df = self.df.with\_columns(pl.col(["vol bid", "vol ask"])/1000)

def print\_to\_excel(self):

self.sheet.range(self.RANGE\_FORMATTED).value = ""

self.sheet.range(self.CELL\_PRINT).options(index=False, header=False).value = self.df.to\_pandas()

try:

run = Df\_operations()

run.obtain\_df()

run.rename\_columns()

run.str\_columns\_adjust()

run.rows\_to\_exclude()

run.bid\_ask\_adjust()

run.volume\_adjust()

run.print\_to\_excel()

except Personal\_exception as error:

error\_global = error

print(error)

class Database\_fixed\_income():

DEB\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa\db\_deb.parquet"

CRA\_PATH = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Database renda fixa\db\_cra.parquet"

COLUMNS = ["ticker", "vencimento", "taxa\_emitido", "indexador", "empresa", "pu\_emitido",

"isin", "data\_emitido", "reais\_emitido", "tipo", "incentivada", "artigo"]

def \_\_init\_\_(self):

self.get\_deb\_database()

self.get\_cra\_database()

self.concat\_databases()

def get\_deb\_database(self):

self.df\_deb = pl.read\_parquet(self.DEB\_PATH).select(pl.col(self.COLUMNS))

def get\_cra\_database(self):

self.df\_cra = pl.read\_parquet(self.CRA\_PATH)

self.df\_cra = self.df\_cra.with\_columns(

pl.lit(True).alias("incentivada"),

pl.lit(None).cast(pl.Int16).alias("artigo")

)

self.df\_cra = self.df\_cra.select(pl.col(self.COLUMNS))

def concat\_databases(self):

self.df = pl.concat([self.df\_deb, self.df\_cra])

class Anbima():

PATH\_ANBIMA\_DEB = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Debêntures\anbima\_deb\_"

PATH\_ANBIMA\_CRA = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Cra\anbima\_cra\_"

PATH\_ANBIMA\_TIT = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\Anbima\Títulos públicos\anbima\_tit\_"

COLUMNS\_CRA = ["ticker", "anbima", "referencia\_b", "duration\_anbima"]

def \_\_init\_\_(self, yesterday, df):

self.yesterday = yesterday

self.yesterday\_string = yesterday.strftime("%Y-%m-%d")

self.df = df

self.get\_anbima\_deb()

self.get\_anbima\_cra()

self.get\_anbima\_tit()

def get\_anbima\_deb(self):

while True:

try:

deb = pl.read\_ipc(self.PATH\_ANBIMA\_DEB + self.yesterday\_string + ".feather")

break

except FileNotFoundError:

self.check\_last\_anbima()

self.deb = deb

self.df = self.df.join(deb, how="left", on="ticker")

def check\_last\_anbima(self):

import requests

URL = 'https://www.anbima.com.br/informacoes/merc-sec-debentures/arqs/db'

r = requests.get(URL + self.yesterday.strftime("%y%m%d") + ".txt")

file\_found = (r.status\_code == 200)

if file\_found:

raise Personal\_exception(f"Último arquivo da anbima {self.yesterday\_string} não baixado.")

else:

self.yesterday = busday\_offset(self.yesterday\_string, -1)

self.yesterday = self.yesterday.item()

self.yesterday\_string = self.yesterday.strftime("%Y-%m-%d")

def get\_anbima\_cra(self):

cra = pl.read\_ipc(self.PATH\_ANBIMA\_CRA + self.yesterday\_string + ".feather")

cra = cra.select(pl.col(self.COLUMNS\_CRA))

self.cra = cra

self.df = self.df.join(cra, how="left", on="ticker", suffix="\_right")

for column in ["anbima", "referencia\_b", "duration\_anbima"]:

self.df = self.df.with\_columns(

pl.when((pl.col(column).is\_null()) & (~pl.col(column + "\_right").is\_null()))

.then(pl.col(column + "\_right"))

.otherwise(pl.col(column))

.alias(column)

).drop(column + "\_right")

def get\_anbima\_tit(self):

tit = pl.read\_ipc(self.PATH\_ANBIMA\_TIT + self.yesterday\_string + ".feather")

tit = tit.with\_columns(

("TIT: " + pl.col("ticker") + " " + pl.col("vencimento").dt.strftime("%y")).alias("ticker")

)

self.tit = tit

columns = []

for column in self.df.columns:

if column not in self.tit.columns:

columns.append(column)

self.tit = self.tit.with\_columns(pl.lit(None).alias(column))

self.tit = self.tit.select(pl.col(self.df.columns))

self.tit = self.tit.with\_columns(

pl.col(["pu\_emitido", "taxa\_emitido", "reais\_emitido", "duration\_anbima"])

.cast(pl.Float32),

pl.col("data\_emitido").cast(pl.Date),

pl.lit("Título público").alias("tipo"),

pl.col(["indexador", "isin", "empresa", "referencia\_b"]).cast(pl.Utf8),

pl.col("incentivada").cast(pl.Boolean),

pl.col("artigo").cast(pl.Int16)

)

self.df = pl.concat([self.tit, self.df])

class Workbook\_info():

WB\_NAME = "RUN Crédito Privado.xlsm"

SHEET\_NEW\_RUN = "Add RUN"

SHEET\_EXTRA = "RUN"

RANGE\_MAX\_LINE = "B2:B3"

RANGE\_NEW\_RUN = "C6:G"

RANGE\_EXTRA = "A18:F"

LAST\_LINE\_DELETE = "2000"

CLIENT\_CELL = "C4"

RANGE\_NEW\_RUN\_DELETE = RANGE\_NEW\_RUN + LAST\_LINE\_DELETE

RANGE\_FORMATTED = "H4:L" + LAST\_LINE\_DELETE

RANGE\_DRAFT = "I5:V" + LAST\_LINE\_DELETE

def \_\_init\_\_(self):

self.obtain\_sheets()

self.obtain\_new\_run()

self.obtain\_extras()

def obtain\_sheets(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet\_new\_run = self.wb.sheets[self.SHEET\_NEW\_RUN]

self.sheet\_extra = self.wb.sheets[self.SHEET\_EXTRA]

self.lines = self.sheet\_new\_run.range(self.RANGE\_MAX\_LINE).value

def obtain\_new\_run(self):

max\_line\_new\_run = str(int(self.lines[1]))

self.exist\_new\_run = self.lines[1] > 5

if self.exist\_new\_run:

self.new\_run = self.sheet\_new\_run.range(self.RANGE\_NEW\_RUN + max\_line\_new\_run).value

self.client = self.sheet\_new\_run.range(self.CLIENT\_CELL).value

if self.client is None:

raise Personal\_exception("Cliente está vazio")

else:

self.client = self.client.upper()

def obtain\_extras(self):

line\_extra = str(int(self.lines[0]))

self.exist\_extra = self.lines[0] > 17

if self.exist\_extra:

self.extra\_values = self.sheet\_extra.range(self.RANGE\_EXTRA + line\_extra).value

def clean\_lines(self):

cells\_to\_delete = [self.CLIENT\_CELL, self.RANGE\_NEW\_RUN\_DELETE, self.RANGE\_DRAFT]

cells\_to\_delete = ", ".join(cells\_to\_delete)

self.sheet\_new\_run.range(cells\_to\_delete).value = ""

class Data\_new\_run():

NEW\_COLUMNS\_NAMES = ["ticker", "vol bid", "bid", "ask", "vol ask", "cliente"]

FLOAT\_COLUMNS\_DEFAULT = NEW\_COLUMNS\_NAMES[1:-1]

STR\_COLUMNS\_DEFAULT = [NEW\_COLUMNS\_NAMES[0], NEW\_COLUMNS\_NAMES[-1]]

def \_\_init\_\_(self, wb):

self.wb = wb

self.obtain\_df()

self.rename\_columns()

self.check\_duplicates()

self.check\_columns\_types()

def obtain\_df(self):

empty\_client = (self.wb.new\_run[0][0] == None)

df = pl.DataFrame(self.wb.new\_run, orient = "row")

self.df = df.with\_columns(pl.lit(self.wb.client).alias("cliente"))

def rename\_columns(self):

dict\_rename\_columns = {}

old\_columns\_names = self.df.columns

for index in range(0, 6):

dict\_rename\_columns[old\_columns\_names[index]] = self.NEW\_COLUMNS\_NAMES[index]

self.df = self.df.rename(dict\_rename\_columns).with\_columns(pl.col(pl.Boolean).cast(pl.Float64))

def check\_duplicates(self):

exist\_duplicates = not (self.df.filter(pl.col("ticker").is\_duplicated()).is\_empty())

if exist\_duplicates:

tickers\_duplicateds = self.df.select(pl.col("ticker")).filter(pl.col("ticker").is\_duplicated()).unique()

raise Personal\_exception(f"Há tickers duplicados:\n{tickers\_duplicateds}")

def check\_columns\_types(self):

str\_columns = self.df.select(pl.col(pl.Utf8)).columns

wrong\_columns = []

for column in str\_columns:

if column not in self.STR\_COLUMNS\_DEFAULT:

wrong\_columns.append(column)

exist\_wrong\_column = len(wrong\_columns) > 0

if exist\_wrong\_column:

error\_msg = "A(s) col(s): " + ", ".join(wrong\_columns)\

+ " está(ão) como número, talvez haja uma celula errada nela(s)"

raise Personal\_exception(error\_msg)

class Runs\_consolidated():

TODAY\_STRING = datetime.today().strftime("%Y-%m-%d")

PATH\_FEATHER = r"T:\Mesa Operacoes\Credito\Narizinho\Rotinas - arquivos\RUNs\run\_" + TODAY\_STRING + ".feather"

def \_\_init\_\_(self, df\_run):

self.df\_run = df\_run

self.read\_file()

self.concatenate\_files()

if not self.df\_runs.is\_empty():

self.duplicated\_rows()

self.write\_concat\_file()

self.consolidate\_files()

def read\_file(self):

self.exist\_previous\_run = True

try:

self.df\_runs = pl.read\_ipc(self.PATH\_FEATHER)

except FileNotFoundError:

self.exist\_previous\_run = False

def concatenate\_files(self):

if self.df\_run.is\_empty():

if not self.exist\_previous\_run:

self.df\_runs = pl.DataFrame()

else:

if self.exist\_previous\_run:

self.df\_runs = pl.concat([self.df\_runs, self.df\_run])

else:

self.df\_runs = self.df\_run

def duplicated\_rows(self):

duplicated = self.df\_runs.filter(pl.lit(self.df\_runs.is\_duplicated())).unique(subset="ticker")

if not duplicated.is\_empty():

error\_msg = (f"Linhas abaixo já existentes no consolidado atual:\n\n{duplicated}")

raise Personal\_exception(error\_msg)

def write\_concat\_file(self):

self.df\_runs = self.df\_runs.sort(by=["ticker", "cliente"])

self.df\_runs.write\_ipc(self.PATH\_FEATHER)

def consolidate\_files(self):

self.df\_runs = (self.df\_runs.with\_columns(

pl.col("bid").filter(pl.col("bid") > 0).min().over("ticker").alias("bid\_min").fill\_null(0),

pl.col("ask").max().over("ticker").alias("ask\_max")

)

.lazy()

.groupby("ticker")

.agg(

pl.col("vol bid").filter((pl.col("bid\_min") == pl.col("bid")) & (pl.col("bid") > 0)).sum().alias("vol\_bid"),

pl.col("bid\_min").first(),

pl.col("ask\_max").first(),

pl.col("vol ask").filter((pl.col("ask\_max") == pl.col("ask")) & (pl.col("ask") > 0)).sum().alias("vol\_ask"),

pl.col("cliente").filter((pl.col("bid\_min") == pl.col("bid")) & (pl.col("bid") > 0)).alias("cliente\_bid"),

pl.col("cliente").filter((pl.col("ask\_max") == pl.col("ask")) & (pl.col("ask") > 0)).alias("cliente\_ask")

)

).collect().with\_columns(

pl.col("cliente\_bid").arr.join(", "),

pl.col("cliente\_ask").arr.join(", "),

).sort("ticker")

class Data\_extra():

def \_\_init\_\_(self, wb):

self.wb = wb

self.obtain\_extras()

pass

def obtain\_extras(self):

self.extras = pl.DataFrame(self.wb.extra\_values, orient = "row")

def kill\_orders(self):

self.extras.filter(pl.col("bid").str.contains("[mM]"))

pass

def convert\_interested\_tickers(self):

pass

class Joined\_info():

SHEET\_CONSOLIDADO = "Consolidado"

RANGE\_PRINT\_CONSOLIDADO = "A3"

COLUMNS\_ORDER = ["ticker", "vol\_bid", "bid\_min", "ask\_max", "vol\_ask", "anbima", "cliente\_bid", "cliente\_ask", "empresa",

"tipo", "indexador", "vencimento", "duration\_anbima", "referencia\_b", "incentivada"]

def \_\_init\_\_(self, database, runs, workbook):

self.df\_db = database

self.df\_runs = runs

self.wb = workbook

self.join\_info()

self.write\_sheet()

self.wb.clean\_lines()

def join\_info(self):

self.df = self.df\_db.join(self.df\_runs, how='outer', on="ticker")

self.df = self.df.select(pl.col(self.COLUMNS\_ORDER))

def write\_sheet(self):

self.wb.wb.sheets[self.SHEET\_CONSOLIDADO].range(self.RANGE\_PRINT\_CONSOLIDADO)\

.options(index=False, header=False).value = self.df.to\_pandas()

class Run\_final():

def \_\_init\_\_(self, df\_extra):

self.extra = df\_extra

self.consolidate\_dfs()

def consolidate\_dfs(self):

pass

def create\_closest\_anbima(self):

pass

try:

today = datetime.today()

yesterday = busday\_offset(today.strftime("%Y-%m-%d"), -1)

yesterday = yesterday.item()

database = Database\_fixed\_income()

df\_db = database.df

anbima = Anbima(yesterday, df\_db)

df\_db = anbima.df

df\_new\_run = pl.DataFrame()

wb = Workbook\_info()

if wb.exist\_new\_run:

new\_run = Data\_new\_run(wb)

df\_new\_run = new\_run.df

wb.exist\_extra = False #por hora, cagar para os extras

if wb.exist\_extra:

extra = Data\_extra(wb)

runs = Runs\_consolidated(df\_run=df\_new\_run)

final\_info = Joined\_info(database=df\_db, runs=runs.df\_runs, workbook=wb)

except Personal\_exception as error: