Documentação Profissional do Projeto de Análise de Dados

Projeto: **Movimentações Financeiras**

Autor: José Augusto **Data:** 31/07/2025

1. Documentação de Requisitos

Objetivo: Entender profundamente as necessidades do negócio e definir os objetivos da análise.

1.1 Levantamento de Requisitos

O case tem como foco a análise de movimentações financeiras de clientes. As principais perguntas de negócio são:

- Qual o volume total de movimentações?
- Quais clientes são mais ativos?
- Como os valores se comportam ao longo do tempo?

1.2 Especificação de Métricas

- Total de Entradas Financeiras
- Total de Saídas Financeiras
- Saldo por Cliente
- Participação do PIX nas transações
- Distribuição mensal
- % de Contas Ativas e Inativas

1.3 Regras de Negócio

- CPF deve ser padronizado e anonimizado
- Entradas e saídas são registradas separadamente
- Datas devem estar formatadas como YYYY-MM-DD

2. Documentação Técnica

Objetivo: Descrever como os dados são coletados, tratados e armazenados.

2.1 Tabelas e Relacionamentos:

Table d_pais

country_id uuid [pk]

country varchar(128)

Table d_estado

state_id uuid [pk]

state varchar(128)

country_id uuid

Table d_cidade

city_id int [pk]

city varchar(256)

state_id uuid

Table d_clientes

customer_id uuid [pk]

first_name varchar(128)

last_name varchar(128)

customer_city int

country_name varchar(128)

cpf varchar(128)

Table d_contas

account_id uuid [pk]

customer_id uuid

created_at timestamp

status varchar(128)

account_branch varchar(128)

account_check_digit varchar(128)

account_number varchar(128)

Table f_entrada_fluxo

id uuid [pk]

account_id uuid

amount float

transaction_requested_at int

transaction_completed_at int

status varchar(128)

data_entrada (data)

Table f_saida_fluxo

id uuid [pk]

account_id uuid

amount float

transaction_requested_at int

transaction_completed_at int

status varchar(128)

data_saida data

Table f_mov_pix

id uuid [pk]

account_id uuid

in_or_out varchar(128)

pix_amount float

pix_requested_at int

pix_completed_at int

status varchar(128)

data_mov_pix data

Table d_calendario

Data data(criada com script no Power BI)

// Foreign key relationships(Relacionamentos)

Ref: d_estado.country_id > d_pais.country_id

Ref: d_cidade.state_id > d_estado.state_id

Ref: d_clientes.customer_city > d_cidade.city_id

Ref: d_contas.customer_id > d_clientes.customer_id

Ref: f_entrada_fluxo.account_id > d_contas.account_id

Ref: f_saida_fluxo.account_id > d_contas.account_id

Ref: f_mov_pix.account_id > d_contas.account_id

Ref: f_entrada_fluxo.data_entrada > d_calendario.data

Ref: f_saida_fluxo.data_saida > d_calendario.data

Ref: f_mov_pix.data_mov_pix > d_calendario.data

2.2 Modelo de Dados (ERD)

- Tabelas dimensão: d_clientes, d_contas, d_cidade, d_estado, d_pais, s_calendario
- Tabelas fato: f_entrada_fluxo, f_saida_fluxo, f_mov_pix

2.3 Pipeline de Dados

- Ingestão: leitura de arquivos .csv
- Transformação: tratamento de CPF, datas, timestamps, duplicatas e nulos, nova coluna criada de nome e sobrenone
- Carga: inserção no PostgreSQL com log e auditoria

2.4 Scripts

- Linguagem: Python
- Bibliotecas: pandas, sqlalchemy, logging, psycopg2
- Estrutura modular com funções para validação, limpeza e log

2.4 a (Script de pipeline detalhado)

a1. Importações de bibliotecas

import numpy as np → Manipulação de dados e tabelas.

import pandas as pd → Manipulação de dados e tabelas.

import psycopg2 → Conexão com PostgreSQL (usado internamente pelo SQLAlchemy).

import json → Leitura do arquivo de configuração db_config.json.

import os → Operações de sistema (caminhos de arquivos, encerrar execução).

import sys → Operações de sistema (caminhos de arquivos, encerrar execução).

import logging → Operações de sistema (caminhos de arquivos, encerrar execução).

from sqlalchemy import create_engine, text, inspect→ Conexão e execução de queries no PostgreSQL.

from datetime import datetime→ Manipulação de datas.

from typing import Dict→ Tipo de anotação para funções que retornam dicionários.

a2. Configuração de diretórios e logs

```
current_dir = os.path.dirname(os.path.abspath(__file__))

CSV_FOLDER = current_dir

log_path = os.path.join(current_dir, 'pipeline2.log')

logging.basicConfig(

   filename=log_path,

   level=logging.INFO,

   format='%(asctime)s - %(levelname)s - %(message)s'
```

)

current_dir → Obtém o caminho da pasta onde está o script.

CSV_FOLDER → Define que os arquivos CSV estão nessa mesma pasta.

log_path → Caminho completo do arquivo de log pipeline2.log.

logging.basicConfig → Configura o log para gravar data/hora, nível e mensagem no arquivo

a3. Lista de arquivos obrigatórios

```
REQUIRED_CSV_FILES = [
    'cidade.csv', 'estado.csv', 'pais.csv', 'clientes.csv', 'contas.csv',
    'entrada_fluxo.csv', 'saida_fluxo.csv', 'mov_pix.csv'
]
```

Define quais arquivos CSV precisam existir para a execução

a4. Função: load_db_config

```
def load_db_config(path: str) -> Dict[str, str]:
    if not os.path.exists(path):
        print(f"❌ Arquivo de configuração '{path}' não encontrado.")
        sys.exit(1)
    with open(path, 'r') as f:
        return json.load(f)
```

Lê um arquivo JSON com as credenciais do banco.

Se não existir, mostra erro e encerra.

a5. Função: create_pg_engine

```
def create_pg_engine(config: Dict[str, str]):
    user = config['user']
    password = config['password']
    host = config['host']
    port = config['port']
    database = config['database']
    return create_engine(f"postgresql+psycopg2://{user}:{password}@{host}:{port}/{database}")
```

Cria uma conexão SQLAlchemy com PostgreSQL usando psycopg2.

a6. Função: load_csv

```
def load_csv(filename: str) -> pd.DataFrame:
    path = os.path.join(CSV_FOLDER, filename)
    if not os.path.exists(path):
        print(f"❌ Arquivo não encontrado: {path}")
        sys.exit(1)
    try:
        return pd.read_csv(path, encoding='utf-8')
    except UnicodeDecodeError:
        print(f"⚠️ Encoding UTF-8 falhou para {filename}, tentando latin1...")
        return pd.read_csv(path, encoding='latin1')
```

Lê um CSV em UTF-8.

Se der erro de codificação, tenta latin1.

Se o arquivo não existir, encerra a execução.

a7. Função: clean_dim

```
def clean_dim(df: pd.DataFrame) -> pd.DataFrame:
    return df.drop_duplicates().dropna()
```

Remove linhas duplicadas e linhas com valores nulos.

a8. Função: clean_clientes

```
def clean_clientes(df: pd.DataFrame) -> pd.DataFrame:
    df['cpf'] = df['cpf'].astype(str).str.replace('.0', '', regex=False).str.zfill(11)
    df['nome_completo'] = df['first_name'].fillna('') + ' ' + df['last_name'].fillna('')
    return df
```

Converte CPF para string, remove sufixo .0 e preenche com zeros à esquerda até 11 dígitos.

Cria coluna nome_completo juntando first_name e last_name.

a9. Função: process_timestamps

```
def process_timestamps(df: pd.DataFrame, col1: str, col2: str, new_col: str) -> pd.DataFrame:
```

```
    df[col1 + '_fmt'] = pd.to_datetime(df[col1], unit='ms', errors='coerce')
    df[col2 + '_fmt'] = pd.to_datetime(df[col2], unit='ms', errors='coerce')
    df[col1 + '_fmt'] = df[col1 + '_fmt'].dt.strftime('%Y-%m-%d %H:%M:%S').fillna('ERRO')
    df[col2 + '_fmt'] = df[col2 + '_fmt'].dt.strftime('%Y-%m-%d %H:%M:%S').fillna('ERRO')
    df[new_col] = pd.to_datetime(df[col1 + '_fmt'], errors='coerce').dt.date
    return df.drop(columns=[col1, col2])
```

Converte colunas de timestamp em milissegundos para datetime.

Formata como YYYY-MM-DD HH:MM:SS.

Cria uma nova coluna só com a data (new_col).

Remove as colunas originais de timestamp.

a10. Função: validate_csv_files

```
def validate_csv_files():
    missing = [f for f in REQUIRED_CSV_FILES if not os.path.exists(os.path.join(CSV_FOLDER, f))]
    if missing:
        for f in missing:
            print(f"❌ Arquivo CSV ausente: {f}")
        sys.exit(1)
```

Verifica se todos os arquivos obrigatórios existem.

Se faltar algum, encerra o scrip

a11. Função: log_insert_with_metrics

```
def log_insert_with_metrics(df: pd.DataFrame, table_name: str, engine):
    metrics = {...}
    try:
        inspector = inspect(engine)
        ...
        df.to_sql(table_name, engine, if_exists='replace', index=False)
        ...
    except Exception as e:
```

```
    ...
  ...
  logging.info(msg)
  print(msg)
  return metrics
```

Gera métricas de inserção:

Quantas linhas foram inseridas.

Quantas linhas foram removidas.

Quais novas colunas surgiram.

Se houve erro.

Insere os dados no PostgreSQL usando replace (apaga e insere de novo).

Registra e imprime as métricas no log.

a12. Função principal: main

```
def main():
  config_path = ...
  validate_csv_files()
  config = load_db_config(config_path)
  engine = create_pg_engine(config)

  totais = {...}

  try:
    # Carrega e limpa dimensões
    d_cidade = clean_dim(load_csv('cidade.csv'))
    ...
    d_clientes = clean_clientes(load_csv('clientes.csv'))
    ...

    # Ajusta datas
    d_contas['data_conta'] = pd.to_datetime(d_contas['created_at'], errors='coerce').dt.date
```

```
        # Processa fatos com timestamps
        f_entrada_fluxo = process_timestamps(...)
        ...

        # Insere todas as tabelas no banco com log
        for df, tbl in [...]:
            met = log_insert_with_metrics(df, tbl, engine)
            ...

        # Mensagem final
        if totais["erros"] == 0:
            final_msg = f"🎉 Pipeline concluído com sucesso! ..."
        else:
            final_msg = f"⚠️ Pipeline concluído com erros..."
        logging.info(final_msg)
        print(final_msg)
    except Exception as e:
        msg = f"❌ Erro geral no pipeline: {e}"
        logging.error(msg)
        print(msg)
```

Fluxo geral da execução:

1. **Valida se todos os CSV existem.**
2. **Lê config do banco e cria conexão.**
3. **Carrega e limpa as tabelas dimensão.**
4. **Trata as datas de criação de contas.**
5. **Processa tabelas fato, ajustando timestamps.**
6. **Insere todas as tabelas no PostgreSQL registrando métricas.**
7. **Mostra mensagem final de sucesso ou erro**

a13. Execução do script

```
if __name__ == '__main__':
    main()
```

Garante que o main() só será executado quando o script for rodado diretamente

2.5 Config. de Conexões Arquivo db_config.json contém as credenciais para o PostgreSQL.Formato esperado:

{

"user": "usuario",

"password": "senha",

"host": "localhost",

"port": "5432",

"database": "nome_banco"

} **-Esse arquivo Json Garante a integridade dos dados pessoais ou das empresas.**

3. Documentação de Processos

Objetivo: Garantir a replicabilidade e manutenção da pipeline.

3.1 Manual de Execução

- Executar via terminal: python pipelinetestefinal.py
- Automatizar via .bat + Agendador de Tarefas

3.2 Arquivo .BAT

@echo off

cd C:\caminho\do\projeto

python pipelinetestefinal.py

pause

3.3 Orquestração

- Utilização do Agendador do Windows
- Alternativas viáveis: Pentaho ou Apache Airflow para pipelines mais robustos

4. Documentação Analítica

Objetivo: Justificar decisões com base na análise dos dados transformados.

4.1 Metodologia (EDA)

- Análise inicial com pandas e seaborn
- Visualização final em Power BI
- Verificação de outliers, sazonalidade e agrupamentos

4.2 Hipóteses Testadas

- Transações se concentram no início de cada mês
- Contas mais antigas movimentam valores maiores
- O uso do PIX vem crescendo mês a mês

4.3 Visualizações e Dashboards (Power BI)

Tipo de Layout

- Dashboard financeiro com blocos bem definidos
- KPIs no topo, filtros laterais e gráficos no corpo central

Paleta de Cores Página Local

- Verde primário (#49553B)
- Branco Para filtros e Menus (#FFFFFF)
- Azul(#118DFF) para Card 1, Vermelho(#118DFF) Card 2, Laranja(#E66C37) Card 3
- Para Gráfico de Pizza Roxo(#8B3897) fatia Saída Pix, Laranja(#E66C37) fatia Entrada Pix, Azul(#118DFF) fatia Entradas Din/Cart, Vermelho(#118DFF) fatia Saídas Din/Cart

Carga Incremental

- Implementada com base na data da última atualização
- Colunas como data_entrada, data_saida, data_mov_pix usadas como referência
- Evita recarga completa a cada atualização

Boas Práticas

- Tabela de medidas centralizada
- Nomes claros e padronizados
- Uso de filtros visuais e hierárquicos
- Responsividade ajustada para múltiplos dispositivos

4.4 Medidas DAX

4.4.1. Entrada Pix

Entrada Pix =

CALCULATE([Total PIX], f_mov_pix[in_or_out] = "pix_in")

Descrição: Calcula o valor total das transações de PIX recebidas.**Lógica:**

- Aplica a medida [Total PIX] considerando apenas registros onde f_mov_pix[in_or_out] seja "pix_in".
- **Objetivo:** Quantificar o montante que entrou via PIX no período analisado.

4.4.2. Entradas Din/Cart

Entradas Din/Cart =

SUM(f_entrada_fluxo[amount])

Descrição: Soma o valor total de entradas em dinheiro ou cartão.**Lógica:**

- Usa a tabela f_entrada_fluxo e soma a coluna amount.
- **Objetivo:** Medir todas as entradas financeiras que não vieram por PIX.

4.4.3. Saída Pix

Saida Pix =

CALCULATE(SUM(f_mov_pix[pix_amount]), f_mov_pix[in_or_out] = "pix_out")

Descrição: Calcula o valor total das transações de PIX enviadas.**Lógica:**

- Soma pix_amount filtrando apenas as linhas onde in_or_out é "pix_out".
- **Objetivo:** Mensurar todo o valor enviado via PIX.

4.4.4. Saídas Din/Cart

Saídas Din/cart =

SUM(f_saida_fluxo[amount])

Descrição: Soma o valor total de saídas financeiras em dinheiro ou cartão.**Lógica:**

- Usa a tabela f_saida_fluxo para somar amount.
- **Objetivo:** Quantificar retiradas e pagamentos que não foram PIX.

4.4.5. Volume Financeiro

Volume Financeiro =

[Entradas Din/Cart] + [Entrada Pix] + [Saídas Din/cart] + [Saida Pix]

Descrição: Soma todas as entradas e saídas financeiras, independentemente do meio.**Lógica:**

- Combina entradas e saídas de dinheiro/cartão e PIX.
- **Objetivo:** Mostrar o volume bruto de movimentações no período.

4.4.6. % Clientes Negativos

% Clientes Negativos =

DIVIDE([Qtd Clientes Negativos], [Qtd Clientes C/ Saldo])

Descrição: Percentual de clientes com saldo negativo.**Lógica:**

- Divide a quantidade de clientes negativos pelo total de clientes com saldo.
- **Objetivo:** Indicar o índice de clientes com saldo abaixo de zero.

4.4.7. % Clientes Positivos

% Clientes Positivos =

DIVIDE([Qtd Clientes Positivos], [Qtd Clientes C/ Saldo])

Descrição: Percentual de clientes com saldo positivo.**Lógica:**

- Divide a quantidade de clientes positivos pelo total de clientes com saldo.
- **Objetivo:** Avaliar a saúde financeira da base de clientes.

4.4.8. Saldo Total

DAX

CopiarEditar

Saldo Total =

[Entradas Din/Cart] +

CALCULATE([Total PIX], f_mov_pix[in_or_out] = "pix_in") -

([Saídas Din/cart] + CALCULATE([Total PIX], f_mov_pix[in_or_out] = "pix_out"))

Descrição: Saldo líquido considerando todas as entradas e saídas.**Lógica:**

- Soma entradas de dinheiro/cartão e PIX recebido
- Subtrai saídas de dinheiro/cartão e PIX enviado.
- **Objetivo:** Apontar o saldo final após movimentações.

4.4.9. Média de entrada PIX por Cliente

Média de entrada PIX por Cliente =

DIVIDE([Entrada Pix], [Qtd Clientes], 0)

Descrição: Valor médio recebido via PIX por cliente.

Objetivo: Avaliar o ticket médio de recebimento PIX.

4.4.10. Média de Entradas Din/Cart por Cliente

Média de Entradas din_cart por Cliente =

DIVIDE([Entradas Din/Cart], [Qtd Clientes], 0)

Descrição: Valor médio de entradas em dinheiro/cartão por cliente.

Objetivo: Medir a média de entradas por outros meios.

4.4.11. Média de saída PIX por Cliente

Média de saida PIX por Cliente =

DIVIDE([Saida Pix], [Qtd Clientes], 0)

Descrição: Valor médio enviado via PIX por cliente.

Objetivo: Indicar a média de transferências PIX.

4.4.12. Média de Saídas Din/Cart por Cliente

Média de Saídas din_cart por Cliente =

DIVIDE([Saídas Din/cart], [Qtd Clientes], 0)

Descrição: Valor médio de saídas em dinheiro/cartão por cliente.

Objetivo: Medir a média de gastos fora do PIX.

4.4.13. RELACIONAMENTO DE TABELAS NO POWER BI

4.5 Compartilhamento

- Publicação no Power BI Service
- Atualização automática com Gateway local
- Segurança via RLS (se necessário)

5. Documentação de Qualidade e Governança

Objetivo: Garantir a confiabilidade e rastreabilidade dos dados.

5.1 Controle de Versões

- Histórico dos arquivos .csv salvo localmente
- Versionamento de código com .py

5.2 Logs e Auditoria

- Log salvo em pipeline2.log
- Cada execução registra data/hora, erros, colunas novas e volume de dados

5.3 Tratamento de Dados Sensíveis

- CPF anonimizado e convertido para string de 11 dígitos
- Dados pessoais expostos nos dashboards internos

5.4 SLA de Atualização

- Atualização semanal automática
- SLA de resposta: 24 horas em caso de falha
-

6. Documentação de Projetos

Objetivo: Monitorar entregas e evolução do projeto.

6.1 Cronograma de Entregas

Etapa	Início	Fim
Requisitos e Planejamento	01/07/2025	03/07/2025
Desenvolvimento Pipeline	04/07/2025	15/07/2025
Validação e Testes	16/07/2025	25/07/2025
Publicação Power BI	26/07/2025	28/07/2025

6.2 Gestão de Tarefas

- **VsCode(Python)** para versionamento do pipeline , dos scripts e ingestão de dados, Análise exploratória, ETL;
- **Agendador do Windows para automatizar**
- **PostgreSql** para armazenar os dados.
- **Power BI para análise de dados e Visualização.**

6.3 Lições Aprendidas

- Automatizar com logs facilita suporte e auditoria
- O uso de carga incremental reduz significativamente o tempo de atualização
- Importância da padronização dos dados de entrada

FLUXO DO PROJETO