Projeto-Teu-Controle-Financeiro — V2 (com pandas)

Descrição

Esta V2 troca a implementação "na unha" por uma base com **pandas** para facilitar leitura/escrita, agregações e preparação para dashboards (CSV/JSON/Excel). Inclui código modular (storage, finance, cli) e exemplos de uso.

Estrutura proposta (em arquivos)

1. `models.py` — definições de tipos e validações leves
2. `storage.py` — funções de leitura/gravação (CSV/JSON/Excel) usando pandas
3. `finance.py` — lógica financeira: normalização, cálculos de montante por dias, atualizar rendimentos, agregações e relatórios prontos para BI
4. `app_cli.py` — CLI simples para operar localmente
5. `requirements.txt` — dependências
6. `examples/` — script de demonstração `demo_v2.py`

Como usar

1. Instale dependências: `pip install -r requirements.txt`
2. Rode `python demo_v2.py` para popular um CSV de exemplo e executar operações básicas.
3. Use `app_cli.py` para gerenciar transações localmente.

Arquivos (cole cada bloco em um arquivo com o mesmo nome)

FILE: requirements.txt

```
pandas>=2.0
numpy
python-dateutil
openpyxl
```

FILE: models.py

```
from __future__ import annotations
from dataclasses import dataclass
from datetime import date
from typing import Optional
```

```
TIPOS = {"receita", "despesa", "investimento"}

@dataclass
class TransacaoRaw:
    # formato bruto usado ao criar entradas (strings permitidas para data)
    data: str
    tipo: str
    valor: float
    juros_dia: Optional[float] = None  # ex.: 0.001 = 0.1% ao dia
    descricao: Optional[str] = None

    def validar(self) -> None:
        if self.tipo not in TIPOS:
            raise ValueError(f"Tipo inválido: {self.tipo}. Deve ser um de
{TIPOS}")
        if not isinstance(self.valor, (int, float)):
            raise ValueError("Valor deve ser número")
```

FILE: storage.py

```
from pathlib import Path
import pandas as pd
from typing import Iterable
from datetime import datetime

CSV_COLUMNS = ["id", "data", "tipo", "valor", "juros_dia", "montante",
"descricao"]


def _ensure_dir(path: Path):
    path.parent.mkdir(parents=True, exist_ok=True)


def load_csv(path: str) -> pd.DataFrame:
    p = Path(path)
    if not p.exists():
        return pd.DataFrame(columns=CSV_COLUMNS)
    df = pd.read_csv(p, parse_dates=["data"])  # data -> datetime64
    # normalize columns
    for c in CSV_COLUMNS:
        if c not in df.columns:
            df[c] = pd.NA
    df = df[CSV_COLUMNS]
    return df


def save_csv(df: pd.DataFrame, path: str) -> None:
    _ensure_dir(Path(path))
```

```
    df_copy = df.copy()
    # ensure date -> ISO
    if pd.api.types.is_datetime64_any_dtype(df_copy["data"]):
        df_copy["data"] = df_copy["data"].dt.strftime("%Y-%m-%d")
    df_copy.to_csv(path, index=False)


def save_excel(df: pd.DataFrame, path: str) -> None:
    _ensure_dir(Path(path))
    df.to_excel(path, index=False)


def save_json(df: pd.DataFrame, path: str) -> None:
    _ensure_dir(Path(path))
    df.to_json(path, orient="records", date_format="iso", force_ascii=False)


def next_id(df: pd.DataFrame) -> int:
    if df.empty:
        return 1
    return int(df["id"].max()) + 1
```

FILE: finance.py

```
from datetime import datetime, date
from math import pow
import pandas as pd
from typing import Optional, Tuple

# espera-se que df contenha colunas: id, data (datetime), tipo, valor
(float), juros_dia (float|NA), montante (float|NA), descricao


def normalize_values(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    # despesa fica negativa
    df["valor"] = df.apply(lambda r: -abs(r["valor"]) if r["tipo"] ==
"despesa" else abs(r["valor"]), axis=1)
    return df


def dias_entre(d1: pd.Timestamp, d2: pd.Timestamp) -> int:
    return max(int((d2.normalize() - d1.normalize()).days), 0)


def calcular_montante(capital: float, juros_dia: float, dias: int) -> float:
    # juros_dia em decimal (ex.: 0.001)
    if juros_dia is None or pd.isna(juros_dia):
        return capital
```

```
    return float(capital * pow(1 + float(juros_dia), dias))


def atualizar_montantes(df: pd.DataFrame, referencia: Optional[date] = None)
-> pd.DataFrame:
    """Atualiza a coluna 'montante' para linhas de tipo 'investimento'
calculadas até a data 'referencia'.
    Se referencia for None, usa hoje().
    """
    df = df.copy()
    if referencia is None:
        referencia = pd.Timestamp(datetime.utcnow().date())
    else:
        referencia = pd.Timestamp(referencia)

    def _calc(row):
        if row["tipo"] != "investimento":
            return row.get("montante", pd.NA)
        dias = dias_entre(pd.Timestamp(row["data"]), referencia)
        return calcular_montante(row["valor"], row.get("juros_dia", None),
dias)

    df["montante"] = df.apply(_calc, axis=1)
    return df


def agregar_por_mes(df: pd.DataFrame) -> pd.DataFrame:
    df2 = normalize_values(df)
    df2 = df2.copy()
    df2["ano_mes"] = df2["data"].dt.to_period("M").astype(str)
    res = df2.groupby(["ano_mes", "tipo"]) ["valor"].sum().reset_index()
    return res


def saldo_acumulado(df: pd.DataFrame) -> pd.DataFrame:
    df2 = normalize_values(df).sort_values(by="data")
    df2["saldo_acumulado"] = df2["valor"].cumsum()
    return df2[["data", "tipo", "valor", "saldo_acumulado"]]


def criar_relatorio_para_bi(df: pd.DataFrame) -> pd.DataFrame:

"""Prepara um dataframe wide por mês com colunas: ano_mes, total_receita,
total_despesa, total_investimento, saldo"""
    agg = agregar_por_mes(df)
    pivot = agg.pivot(index="ano_mes", columns="tipo",
values="valor").fillna(0)
    pivot["saldo"] = pivot.get("receita", 0) + pivot.get("investimento", 0) +
pivot.get("despesa", 0)
    pivot = pivot.reset_index()
    return pivot
```

```
def inserir_transacao(df: pd.DataFrame, data: date | str, tipo: str, valor:
float, juros_dia: Optional[float] = None, descricao: Optional[str] = None) ->
pd.DataFrame:
    df = df.copy()
    new_id = int(df["id"].max()) + 1 if not df.empty else 1
    data_ts = pd.to_datetime(data).normalize()
    df = pd.concat([df, pd.DataFrame([{"id": new_id, "data": data_ts, "tipo":
tipo, "valor": float(valor), "juros_dia": juros_dia, "montante": pd.NA,
"descricao": descricao}])], ignore_index=True)
    return df


def atualizar_transacao(df: pd.DataFrame, id_: int, **fields) ->
pd.DataFrame:
    df = df.copy()
    mask = df["id"] == id_
    if not mask.any():
        raise KeyError(f"ID {id_} não encontrado")
    for k, v in fields.items():
        if k == "data":
            df.loc[mask, "data"] = pd.to_datetime(v).normalize()
        else:
            df.loc[mask, k] = v
    return df


def deletar_transacao(df: pd.DataFrame, id_: int) -> pd.DataFrame:
    if id_ not in df["id"].values:
        raise KeyError(f"ID {id_} não encontrado")
    return df[df["id"] != id_].reset_index(drop=True)
```

FILE: demo_v2.py

```
"""Demo que cria um CSV de exemplo, atualiza montantes e gera relatórios
prontos para BI."""
from pathlib import Path
import pandas as pd
from datetime import date, timedelta
from storage import save_csv, load_csv, save_json, save_excel
from finance import inserir_transacao, atualizar_montantes,
criar_relatorio_para_bi

DATA_FILE = "data/transacoes_v2.csv"

# start a fresh df
try:
    df = load_csv(DATA_FILE)
```

```
except Exception:
    df =
pd.DataFrame(columns=["id","data","tipo","valor","juros_dia","montante","descricao"])

# se vazio, popula exemplo
if df.empty:
    df = df.append({"id":1,"data":pd.to_datetime(date.today()-
timedelta(days=60)),"tipo":"receita","valor":
5000.0,"juros_dia":pd.NA,"montante":pd.NA,"descricao":"Salário"},
ignore_index=True)
    df = df.append({"id":2,"data":pd.to_datetime(date.today()-
timedelta(days=45)),"tipo":"despesa","valor":
1500.0,"juros_dia":pd.NA,"montante":pd.NA,"descricao":"Aluguel"},
ignore_index=True)
    df = df.append({"id":3,"data":pd.to_datetime(date.today()-
timedelta(days=30)),"tipo":"investimento","valor":1000.0,"juros_dia":
0.001,"montante":pd.NA,"descricao":"Tesouro"}, ignore_index=True)

# atualiza montantes até hoje
from finance import atualizar_montantes

df = atualizar_montantes(df)

# salva em formatos
Path("data").mkdir(exist_ok=True)
save_csv(df, DATA_FILE)
save_json(df, "data/transacoes_v2.json")
save_excel(df, "data/transacoes_v2.xlsx")

# gera relatório pronto para BI
from finance import criar_relatorio_para_bi
rel = criar_relatorio_para_bi(df)
rel.to_csv("data/relatorio_mensal.csv", index=False)

print("Demo executada. Arquivos em ./data/")
```

FILE: app_cli.py

```
"""CLI mínimo para operar sobre o CSV local (listar, inserir, atualizar,
deletar, exportar)."""
import argparse
import pandas as pd
from storage import load_csv, save_csv
from finance import inserir_transacao, atualizar_transacao,
deletar_transacao, atualizar_montantes

DATA_FILE = "data/transacoes_v2.csv"
```

```
def cmd_list(args):
    df = load_csv(DATA_FILE)
    if df.empty:
        print("Nenhuma transação encontrada.")
        return
    print(df.sort_values(by="data").to_string(index=False))


def cmd_add(args):
    df = load_csv(DATA_FILE)
    df = inserir_transacao(df, args.data, args.tipo, args.valor,
args.juros_dia, args.descricao)
    save_csv(df, DATA_FILE)
    print("Inserido com sucesso")


def cmd_update(args):
    df = load_csv(DATA_FILE)
    df = atualizar_transacao(df, args.id, **{k:v for k,v in
vars(args).items() if k in ["data","tipo","valor","juros_dia","descricao"]
and v is not None})
    save_csv(df, DATA_FILE)
    print("Atualizado")


def cmd_delete(args):
    df = load_csv(DATA_FILE)
    df = deletar_transacao(df, args.id)
    save_csv(df, DATA_FILE)
    print("Deletado")


def cmd_update_rendimentos(args):
    df = load_csv(DATA_FILE)
    df = atualizar_montantes(df, referencia=None)
    save_csv(df, DATA_FILE)
    print("Montantes atualizados")


parser = argparse.ArgumentParser(prog="tcf-v2")
sub = parser.add_subparsers()

p = sub.add_parser("list")
p.set_defaults(func=cmd_list)

p = sub.add_parser("add")
p.add_argument("data")
p.add_argument("tipo")
p.add_argument("valor", type=float)
p.add_argument("--juros_dia", type=float, default=None)
p.add_argument("--descricao", default=None)
```

```
p.set_defaults(func=cmd_add)

p = sub.add_parser("update")
p.add_argument("id", type=int)
p.add_argument("--data")
p.add_argument("--tipo")
p.add_argument("--valor", type=float)
p.add_argument("--juros_dia", type=float)
p.add_argument("--descricao")
p.set_defaults(func=cmd_update)

p = sub.add_parser("delete")
p.add_argument("id", type=int)
p.set_defaults(func=cmd_delete)

p = sub.add_parser("update_rendimentos")
p.set_defaults(func=cmd_update_rendimentos)

if __name__ == "__main__":
    args = parser.parse_args()
    if hasattr(args, "func"):
        args.func(args)
    else:
        parser.print_help()
```

Observações e próximos passos sugeridos

- Posso converter isto para um pacote instalável (`setup.py` / `pyproject.toml`) se quiser facilitar distribuição.
- Posso também adicionar testes unitários (`pytest`) e CI (GitHub Actions).
- Se quiser, geramos Jupyter Notebook com visualizações (matplotlib/plotly) e um arquivo `.pbix` mínimo para Power BI (precisa do CSV exportado).

Se quiser que eu gere ZIP com estes arquivos ou que implemente diretamente algum recurso extra (ex.: integração com SQLite, autenticação de usuário, ou um dashboard básico em Streamlit), me diz que eu já adiciono no próximo passo.