import pandas as pd

import numpy as np

import json

import os

from datetime import datetime

from pathlib import Path

class CSVAnalyzer:

def \_\_init\_\_(self, chunk\_size=10000):

self.df = None

self.file\_path = None

self.chunk\_size = chunk\_size

self.total\_rows = 0

self.columns = []

self.analysis\_results = {}

def read\_csv\_chunked(self, file\_path):

self.file\_path = str(file\_path)

try:

sample\_df = pd.read\_csv(file\_path, nrows=1000)

self.columns = list(sample\_df.columns)

with open(file\_path, "r", encoding="utf-8") as f:

self.total\_rows = sum(1 for \_ in f) - 1

if self.total\_rows > self.chunk\_size:

self.\_process\_large\_csv()

else:

self.df = pd.read\_csv(file\_path)

return True

except Exception as e:

print(f"❌ Error reading CSV: {e}")

return False

def \_process\_large\_csv(self):

chunk\_reader = pd.read\_csv(self.file\_path, chunksize=self.chunk\_size)

chunk\_summaries = []

dtypes\_info, missing\_info, categorical\_info = {}, {}, {}

for chunk in chunk\_reader:

for col in chunk.columns:

if col not in dtypes\_info:

dtypes\_info[col] = str(chunk[col].dtype)

missing\_in\_chunk = chunk.isnull().sum()

for col, missing\_count in missing\_in\_chunk.items():

missing\_info[col] = missing\_info.get(col, 0) + missing\_count

for col in chunk.select\_dtypes(include=["object", "category"]).columns:

if col not in categorical\_info:

categorical\_info[col] = {}

value\_counts = chunk[col].value\_counts().head(10)

for value, count in value\_counts.items():

categorical\_info[col][value] = categorical\_info[col].get(value, 0) + count

numerical\_cols = chunk.select\_dtypes(include=[np.number]).columns

if len(numerical\_cols) > 0:

chunk\_summary = chunk[numerical\_cols].describe()

chunk\_summaries.append(chunk\_summary)

self.chunk\_analysis = {

"dtypes\_info": dtypes\_info,

"missing\_info": missing\_info,

"categorical\_info": categorical\_info,

"chunk\_summaries": chunk\_summaries,

}

def separate\_data\_types(self):

if self.df is not None:

num\_cols = list(self.df.select\_dtypes(include=[np.number]).columns)

cat\_cols = list(self.df.select\_dtypes(include=["object", "category"]).columns)

dt\_cols = list(self.df.select\_dtypes(include=["datetime64"]).columns)

for col in cat\_cols.copy():

try:

pd.to\_datetime(self.df[col].dropna().head(10), errors="raise")

dt\_cols.append(col)

cat\_cols.remove(col)

except Exception:

pass

else:

dtypes\_info = self.chunk\_analysis["dtypes\_info"]

num\_cols, cat\_cols, dt\_cols = [], [], []

for col, dtype in dtypes\_info.items():

if "int" in dtype or "float" in dtype:

num\_cols.append(col)

elif "datetime" in dtype:

dt\_cols.append(col)

else:

cat\_cols.append(col)

return tuple(num\_cols), tuple(cat\_cols), tuple(dt\_cols)

def analyze\_dataset(self):

analysis = {}

num\_cols, cat\_cols, dt\_cols = self.separate\_data\_types()

analysis["basic\_info"] = {

"total\_rows": self.total\_rows,

"total\_columns": len(self.columns),

"file\_name": os.path.basename(self.file\_path),

"columns": self.columns,

"processing\_method": "chunked" if self.df is None else "full\_load",

}

analysis["data\_types"] = {

"numerical\_columns": list(num\_cols),

"categorical\_columns": list(cat\_cols),

"datetime\_columns": list(dt\_cols),

}

if self.df is not None:

self.\_analyze\_complete(analysis, num\_cols, cat\_cols)

else:

self.\_analyze\_chunked(analysis, num\_cols, cat\_cols)

self.analysis\_results = analysis

return analysis

def \_analyze\_complete(self, analysis, num\_cols, cat\_cols):

missing = self.df.isnull().sum()

analysis["missing\_values"] = {

"total\_missing": int(missing.sum()),

"columns\_with\_missing": {c: int(v) for c, v in missing[missing > 0].items()},

}

if num\_cols:

desc = self.df[list(num\_cols)].describe()

analysis["numerical\_summary"] = {

c: {k: float(v) for k, v in desc[c].to\_dict().items()} for c in desc.columns

}

cat\_info = {}

for c in cat\_cols:

cat\_info[c] = {

"unique\_values": int(self.df[c].nunique()),

"most\_common": self.df[c].value\_counts().head(3).to\_dict(),

}

analysis["categorical\_insights"] = cat\_info

total\_cells = len(self.df) \* len(self.df.columns)

completeness = ((total\_cells - missing.sum()) / total\_cells) \* 100

analysis["data\_quality"] = {"duplicate\_rows": int(self.df.duplicated().sum()), "completeness\_score": completeness}

def \_analyze\_chunked(self, analysis, num\_cols, cat\_cols):

info = self.chunk\_analysis

total\_missing = sum(info["missing\_info"].values())

analysis["missing\_values"] = {"total\_missing": total\_missing}

if num\_cols and info["chunk\_summaries"]:

combined = {}

for col in num\_cols:

stats = [chunk[col] for chunk in info["chunk\_summaries"] if col in chunk.columns]

if stats:

combined[col] = {

"count": sum(s["count"] for s in stats),

"mean": np.mean([s["mean"] for s in stats]),

"std": np.mean([s["std"] for s in stats]),

"min": min(s["min"] for s in stats),

"max": max(s["max"] for s in stats),

}

analysis["numerical\_summary"] = combined

cat\_info = {}

for col in cat\_cols:

if col in info["categorical\_info"]:

sorted\_vals = sorted(info["categorical\_info"][col].items(), key=lambda x: x[1], reverse=True)

cat\_info[col] = {"unique\_values": len(info["categorical\_info"][col]), "most\_common": dict(sorted\_vals[:3])}

analysis["categorical\_insights"] = cat\_info

total\_cells = self.total\_rows \* len(self.columns)

completeness = ((total\_cells - total\_missing) / total\_cells) \* 100

analysis["data\_quality"] = {"duplicate\_rows": 0, "completeness\_score": completeness}

def generate\_json(self, job\_id=None):

analysis = self.analysis\_results

return {

"main\_title": "CSV Dataset Overview & Analysis",

"summary\_stats": {

"total\_records": analysis["basic\_info"]["total\_rows"],

"total\_variables": analysis["basic\_info"]["total\_columns"],

"completeness\_percentage": round(analysis["data\_quality"]["completeness\_score"], 2),

"processing\_method": analysis["basic\_info"]["processing\_method"],

},

"data\_type\_separation": analysis["data\_types"],

"full\_analysis": analysis,

"generation\_timestamp": datetime.now().isoformat(),

"processing\_info": {

"job\_id": job\_id,

"file\_path": str(self.file\_path),

"processing\_status": "completed",

"chunk\_size\_used": self.chunk\_size,

},

}

# 🔑 This is the function others will call

def description(file\_path, chunk\_size=10000):

analyzer = CSVAnalyzer(chunk\_size=chunk\_size)

if not analyzer.read\_csv\_chunked(file\_path):

return {"error": f"Could not read file {file\_path}"}

analyzer.analyze\_dataset()

return analyzer.generate\_json(job\_id="description\_call")

# CLI Support

def main():

import sys

if len(sys.argv) > 1 and sys.argv[1] == "test":

if len(sys.argv) > 2:

file\_path = sys.argv[2]

chunk\_size = int(sys.argv[3]) if len(sys.argv) > 3 else 10000

result = description(file\_path, chunk\_size)

out\_file = Path("outputs") / "test\_description.json"

out\_file.parent.mkdir(exist\_ok=True)

with open(out\_file, "w") as f:

json.dump(result, f, indent=2, default=str)

print(f"✅ Test complete → {out\_file}")

else:

print("Usage: python csv\_analyzer.py test <file\_path> [chunk\_size]")

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

main()