import pandas as pd

import numpy as np

import io

import os

INPUT\_PATH = "Preprocessed Data.txt" # adjust if needed

def robust\_read\_table(path):

# Try explicit tab first

try:

df = pd.read\_csv(path, sep="\t", engine="python")

print("Read with sep='\\t'")

except Exception:

try:

df = pd.read\_csv(path, sep=",", engine="python")

print("Read with sep=','")

except Exception:

# last resort: let pandas sniff

df = pd.read\_csv(path, sep=None, engine="python")

print("Read with sep=None (sniffed)")

# Strip whitespace from column names

df.columns = [c.strip() for c in df.columns]

return df

def detect\_and\_clean\_numeric(df, min\_numeric\_ratio=0.5):

df\_clean = df.copy()

numeric\_cols = []

non\_numeric\_cols = []

for col in df\_clean.columns:

# Try converting column to numeric

converted = pd.to\_numeric(df\_clean[col].astype(str).str.replace(",", "").str.strip(), errors="coerce")

numeric\_ratio = converted.notna().mean()

if numeric\_ratio >= min\_numeric\_ratio:

# keep as numeric (coerced)

df\_clean[col] = converted

numeric\_cols.append(col)

else:

non\_numeric\_cols.append(col)

print(f"Detected numeric-like columns ({len(numeric\_cols)}): {numeric\_cols}")

print(f"Detected non-numeric columns ({len(non\_numeric\_cols)}): {non\_numeric\_cols}")

return df\_clean, numeric\_cols, non\_numeric\_cols

# Load

if not os.path.exists(INPUT\_PATH):

raise FileNotFoundError(f"{INPUT\_PATH} not found in {os.getcwd()}")

df = robust\_read\_table(INPUT\_PATH)

# If the first few rows look shifted, display them to inspect

print("First 6 rows (raw):")

print(df.head(6).to\_string(index=False))

# Normalize common compound column name

for cand in ["Compound Name", "Compound", "compound", "CompoundName"]:

if cand in df.columns:

df = df.rename(columns={cand: "Compound"})

break

# Clean numeric columns: convert where majority of values are numeric

df\_clean, numeric\_cols, non\_numeric\_cols = detect\_and\_clean\_numeric(df, min\_numeric\_ratio=0.6)

# If important descriptor columns got classified as non-numeric, you can force them:

# forced = ["Water Solubility", "Caco2 Permeability", "Intestinal Absorption (%)", "BBB Permeability", "Total Clearance"]

# for c in forced:

# if c in df.columns:

# df\_clean[c] = pd.to\_numeric(df[c].astype(str).str.replace(",", "").str.strip(), errors="coerce")

# if c not in numeric\_cols:

# numeric\_cols.append(c)

# if c in non\_numeric\_cols: non\_numeric\_cols.remove(c)

# Inspect rows with many NaNs in numeric columns (parsing issues)

na\_counts = df\_clean[numeric\_cols].isna().sum(axis=1)

bad\_rows = df\_clean[na\_counts > 0].head(10) # show some problematic rows

if not bad\_rows.empty:

print("Example rows with NaNs in numeric columns (inspect for parsing issues):")

print(bad\_rows[["Compound"] + numeric\_cols].head(10).to\_string(index=False))

# Final numeric matrix for modeling

X\_numeric = df\_clean[numeric\_cols].copy()

# Drop columns that are identifiers or have very low variance / mostly NaN

drop\_cols = [c for c in X\_numeric.columns if X\_numeric[c].nunique() <= 1 or X\_numeric[c].isna().mean() > 0.5]

if drop\_cols:

print(f"Dropping low-information or mostly-NaN columns: {drop\_cols}")

X\_numeric = X\_numeric.drop(columns=drop\_cols)

# Fill remaining NaNs with median

X\_numeric = X\_numeric.fillna(X\_numeric.median())

print("Numeric matrix ready. Shape:", X\_numeric.shape)

# Now X\_numeric is safe to pass to StandardScaler, PCA, t-SNE, or model training

import pandas as pd

import numpy as np

import io

import os

INPUT\_PATH = "Preprocessed Data.txt" # adjust if needed

def robust\_read\_table(path):

# Try explicit tab first

try:

df = pd.read\_csv(path, sep="\t", engine="python")

print("Read with sep='\\t'")

except Exception:

try:

df = pd.read\_csv(path, sep=",", engine="python")

print("Read with sep=','")

except Exception:

# last resort: let pandas sniff

df = pd.read\_csv(path, sep=None, engine="python")

print("Read with sep=None (sniffed)")

# Strip whitespace from column names

df.columns = [c.strip() for c in df.columns]

return df

def detect\_and\_clean\_numeric(df, min\_numeric\_ratio=0.5):

df\_clean = df.copy()

numeric\_cols = []

non\_numeric\_cols = []

for col in df\_clean.columns:

# Try converting column to numeric

converted = pd.to\_numeric(df\_clean[col].astype(str).str.replace(",", "").str.strip(), errors="coerce")

numeric\_ratio = converted.notna().mean()

if numeric\_ratio >= min\_numeric\_ratio:

# keep as numeric (coerced)

df\_clean[col] = converted

numeric\_cols.append(col)

else:

non\_numeric\_cols.append(col)

print(f"Detected numeric-like columns ({len(numeric\_cols)}): {numeric\_cols}")

print(f"Detected non-numeric columns ({len(non\_numeric\_cols)}): {non\_numeric\_cols}")

return df\_clean, numeric\_cols, non\_numeric\_cols

# Load

if not os.path.exists(INPUT\_PATH):

raise FileNotFoundError(f"{INPUT\_PATH} not found in {os.getcwd()}")

df = robust\_read\_table(INPUT\_PATH)

# If the first few rows look shifted, display them to inspect

print("First 6 rows (raw):")

print(df.head(6).to\_string(index=False))

# Normalize common compound column name

for cand in ["Compound Name", "Compound", "compound", "CompoundName"]:

if cand in df.columns:

df = df.rename(columns={cand: "Compound"})

break

# Clean numeric columns: convert where majority of values are numeric

df\_clean, numeric\_cols, non\_numeric\_cols = detect\_and\_clean\_numeric(df, min\_numeric\_ratio=0.6)

# If important descriptor columns got classified as non-numeric, you can force them:

# forced = ["Water Solubility", "Caco2 Permeability", "Intestinal Absorption (%)", "BBB Permeability", "Total Clearance"]

# for c in forced:

# if c in df.columns:

# df\_clean[c] = pd.to\_numeric(df[c].astype(str).str.replace(",", "").str.strip(), errors="coerce")

# if c not in numeric\_cols:

# numeric\_cols.append(c)

# if c in non\_numeric\_cols: non\_numeric\_cols.remove(c)

# Inspect rows with many NaNs in numeric columns (parsing issues)

na\_counts = df\_clean[numeric\_cols].isna().sum(axis=1)

bad\_rows = df\_clean[na\_counts > 0].head(10) # show some problematic rows

if not bad\_rows.empty:

print("Example rows with NaNs in numeric columns (inspect for parsing issues):")

print(bad\_rows[["Compound"] + numeric\_cols].head(10).to\_string(index=False))

# Final numeric matrix for modeling

X\_numeric = df\_clean[numeric\_cols].copy()

# Drop columns that are identifiers or have very low variance / mostly NaN

drop\_cols = [c for c in X\_numeric.columns if X\_numeric[c].nunique() <= 1 or X\_numeric[c].isna().mean() > 0.5]

if drop\_cols:

print(f"Dropping low-information or mostly-NaN columns: {drop\_cols}")

X\_numeric = X\_numeric.drop(columns=drop\_cols)

# Fill remaining NaNs with median

X\_numeric = X\_numeric.fillna(X\_numeric.median())

print("Numeric matrix ready. Shape:", X\_numeric.shape)

# Now X\_numeric is safe to pass to StandardScaler, PCA, t-SNE, or model training

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_numeric)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_numeric)

# find\_moschamine.py

import pandas as pd

import os

ORIG\_PATH = "Preprocessed Data.txt"

RANKED\_PATH = "ranked\_predictions\_export.csv"

OUT\_SNIPPET = "moschamine\_and\_top10.csv"

# load ranked predictions (try common variants)

if os.path.exists(RANKED\_PATH):

ranked = pd.read\_csv(RANKED\_PATH, index\_col=0, engine="python")

else:

raise FileNotFoundError(f"{RANKED\_PATH} not found in {os.getcwd()}")

# ensure column names trimmed

ranked.columns = [c.strip() for c in ranked.columns]

# locate Moschamine in ranked list (case-insensitive)

mask = ranked["Compound"].str.lower() == "moschamine"

if mask.any():

mos\_row = ranked[mask].iloc[0]

mos\_rank = int(ranked[mask].index[0])

print("Moschamine found in ranked predictions")

print(f"Rank: {mos\_rank}")

print(f"Predicted\_Score: {mos\_row['Predicted\_Score']}")

else:

print("Moschamine not found in ranked\_predictions\_export.csv. Showing top 20 instead:")

print(ranked.head(20).to\_string())

# load original preprocessed data (auto-detect separator)

orig = pd.read\_csv(ORIG\_PATH, sep=None, engine="python")

orig.columns = [c.strip() for c in orig.columns]

# normalize compound column name

for cand in ["Compound Name", "Compound", "compound"]:

if cand in orig.columns:

orig = orig.rename(columns={cand: "Compound"})

break

# show Moschamine descriptors in original data

mos\_orig = orig[orig["Compound"].str.lower() == "moschamine".lower()]

if not mos\_orig.empty:

print("\nMoschamine descriptors from original dataset:")

print(mos\_orig.T.to\_string())

else:

print("\nMoschamine not found in original dataset file. Check spelling or column names.")

# Save Moschamine + top10 to CSV for quick review

top10 = ranked.reset\_index(drop=False).rename(columns={"index":"Rank"}).head(10)

if mask.any():

mos\_df = ranked[mask].reset\_index(drop=False).rename(columns={"index":"Rank"})

out\_df = pd.concat([mos\_df, top10]).drop\_duplicates()

else:

out\_df = top10

out\_df.to\_csv(OUT\_SNIPPET, index=False)

print(f"\nSaved {OUT\_SNIPPET} in {os.getcwd()}")

# compute\_errors.py

import pandas as pd

import numpy as np

import os

from sklearn.metrics import mean\_squared\_error

# ---------- CONFIG ----------

ORIG\_FILE = "Preprocessed Data.txt" # original file with true Interaction\_Score

PRED\_FILE = "ranked\_predictions\_export.csv" # file with Predicted\_Score and Compound

OUTPUT\_CSV = "per\_compound\_errors.csv"

# ----------------------------

def read\_original(path):

if not os.path.exists(path):

raise FileNotFoundError(f"{path} not found in {os.getcwd()}")

# try tab or autodetect

try:

df = pd.read\_csv(path, sep="\t", engine="python")

except Exception:

df = pd.read\_csv(path, sep=None, engine="python")

df.columns = [c.strip() for c in df.columns]

# normalize compound column name

for cand in ["Compound Name", "Compound", "compound", "CompoundName"]:

if cand in df.columns:

df = df.rename(columns={cand: "Compound"})

break

return df

def read\_predictions(path):

if not os.path.exists(path):

raise FileNotFoundError(f"{path} not found in {os.getcwd()}")

# ranked CSV may have an index column named "Rank" or similar; load normally

df = pd.read\_csv(path, engine="python")

df.columns = [c.strip() for c in df.columns]

# try to find predicted score column

if "Predicted\_Score" not in df.columns:

candidates = [c for c in df.columns if "pred" in c.lower() or "score" in c.lower()]

if not candidates:

raise ValueError("Predicted score column not found in predictions file.")

df = df.rename(columns={candidates[0]: "Predicted\_Score"})

# normalize compound name column

for cand in ["Compound", "compound", "Compound Name"]:

if cand in df.columns:

df = df.rename(columns={cand: "Compound"})

break

return df[["Compound", "Predicted\_Score"]]

def ensure\_numeric(series):

return pd.to\_numeric(series.astype(str).str.replace(",", "").str.strip(), errors="coerce")

def main():

orig = read\_original(ORIG\_FILE)

preds = read\_predictions(PRED\_FILE)

# ensure Interaction\_Score exists in original (true labels)

if "Interaction\_Score" not in orig.columns:

raise ValueError(f"Original file must contain Interaction\_Score column with true labels. Column not found in {ORIG\_FILE}")

# merge on Compound (case-insensitive)

orig["Compound\_key"] = orig["Compound"].astype(str).str.strip().str.lower()

preds["Compound\_key"] = preds["Compound"].astype(str).str.strip().str.lower()

merged = pd.merge(orig, preds, on="Compound\_key", how="inner", suffixes=("\_orig", "\_pred"))

if merged.empty:

raise ValueError("No matching compounds found between original data and predictions. Check compound names and spelling.")

# prefer original Compound naming for output

# compute\_errors\_with\_proxy.py

import os

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

# CONFIG

ORIG = "Preprocessed Data.txt"

PRED = "ranked\_predictions\_export.csv"

OUT = "per\_compound\_errors.csv"

RANDOM\_STATE = 42

def robust\_read(path):

if not os.path.exists(path):

raise FileNotFoundError(f"{path} not found in {os.getcwd()}")

try:

df = pd.read\_csv(path, sep="\t", engine="python")

except Exception:

df = pd.read\_csv(path, sep=None, engine="python")

df.columns = [c.strip() for c in df.columns]

# normalize compound column

for cand in ["Compound Name", "Compound", "compound", "CompoundName"]:

if cand in df.columns:

df = df.rename(columns={cand: "Compound"})

break

return df

def detect\_numeric(df, min\_ratio=0.6):

numeric\_cols = []

for c in df.columns:

if c == "Compound":

continue

test = pd.to\_numeric(df[c].astype(str).str.replace(",","").str.strip(), errors="coerce")

if test.notna().mean() >= min\_ratio:

numeric\_cols.append(c)

return numeric\_cols

def build\_proxy(df, numeric\_cols, top\_k=6):

if not numeric\_cols:

raise ValueError("No numeric columns detected to build a proxy.")

# pick most variable numeric cols

numdf = df[numeric\_cols].apply(pd.to\_numeric, errors="coerce")

numdf = numdf.fillna(numdf.median())

vars\_sorted = numdf.var().sort\_values(ascending=False)

selected = vars\_sorted.index[:top\_k].tolist()

# standardize selected then average

scaler = StandardScaler()

S = scaler.fit\_transform(numdf[selected])

proxy\_raw = S.mean(axis=1)

# penalize known toxicity flags if present

penalty = np.zeros(len(df))

tox\_cols = [c for c in df.columns if any(tok in c.lower() for tok in ["ames","herg","hepatotox","toxic"])]

for c in tox\_cols:

s = df[c].astype(str).str.lower()

yes = s.str.contains("yes", na=False).astype(float).values

# numeric fallback: >0.5 considered positive

try:

num = pd.to\_numeric(df[c], errors="coerce").fillna(0).values

yes = np.where(num > 0.5, 1.0, yes)

except Exception:

pass

penalty += 0.8 \* yes

proxy\_raw = proxy\_raw - penalty

# scale 0-1

mn, mx = proxy\_raw.min(), proxy\_raw.max()

if mx - mn == 0:

proxy = np.full(len(proxy\_raw), 0.5)

else:

proxy = (proxy\_raw - mn) / (mx - mn)

return proxy, selected, tox\_cols

def read\_predictions(path):

if not os.path.exists(path):

raise FileNotFoundError(f"{path} not found in {os.getcwd()}")

df = pd.read\_csv(path, engine="python")

df.columns = [c.strip() for c in df.columns]

# find predicted column

if "Predicted\_Score" not in df.columns:

cands = [c for c in df.columns if "pred" in c.lower() or "score" in c.lower()]

if not cands:

raise ValueError("Could not find Predicted\_Score column in predictions file.")

df = df.rename(columns={cands[0]: "Predicted\_Score"})

for cand in ["Compound", "compound", "Compound Name"]:

if cand in df.columns:

df = df.rename(columns={cand: "Compound"})

break

return df[["Compound","Predicted\_Score"]]

def main():

orig = robust\_read(ORIG)

preds = read\_predictions(PRED)

if "Compound" not in orig.columns:

orig.insert(0, "Compound", [f"cmpd\_{i}" for i in range(len(orig))])

numeric\_cols = detect\_numeric(orig, min\_ratio=0.5)

if not numeric\_cols:

raise ValueError("No numeric columns detected in the original file to build a proxy Interaction\_Score.")

proxy, selected, tox\_cols = build\_proxy(orig, numeric\_cols, top\_k=6)

orig["Interaction\_Score\_proxy"] = np.round(proxy, 6)

print("Proxy built using numeric features:", selected)

if tox\_cols:

print("Toxicity-like columns considered for penalty:", tox\_cols)

# merge on compound (case-insensitive)

orig["ckey"] = orig["Compound"].astype(str).str.strip().str.lower()

preds["ckey"] = preds["Compound"].astype(str).str.strip().str.lower()

merged = pd.merge(orig, preds, on="ckey", how="inner", suffixes=("\_orig","\_pred"))

if merged.empty:

raise ValueError("No matching compounds between original and predictions. Check names.")

merged["Compound\_out"] = merged["Compound\_orig"]

# numeric coercion

merged["Interaction\_Score\_true"] = pd.to\_numeric(merged["Interaction\_Score\_proxy"], errors="coerce")

merged["Predicted\_Score"] = pd.to\_numeric(merged["Predicted\_Score"], errors="coerce")

merged = merged.dropna(subset=["Interaction\_Score\_true","Predicted\_Score"]).reset\_index(drop=True)

merged["Error"] = merged["Predicted\_Score"] - merged["Interaction\_Score\_true"]

merged["Squared\_Error"] = merged["Error"]\*\*2

merged["RMSE\_i"] = np.sqrt(merged["Squared\_Error"])

mse\_overall = merged["Squared\_Error"].mean()

rmse\_overall = np.sqrt(mse\_overall)

out = merged[["Compound\_out","Interaction\_Score\_true","Predicted\_Score","Error","Squared\_Error","RMSE\_i"]].rename(columns={"Compound\_out":"Compound"})

out.to\_csv(OUT, index=False)

print(f"Saved per-compound errors to {OUT}")

print(f"Compounds evaluated: {len(merged)}")

print(f"Overall MSE: {mse\_overall:.6f}")

print(f"Overall RMSE: {rmse\_overall:.6f}")

print("\nTop 10 worst-predicted compounds by RMSE\_i:")

print(out.sort\_values("RMSE\_i", ascending=False).head(10).to\_string(index=False))

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

main()

import pandas as pd

df = pd.read\_csv("per\_compound\_errors.csv")

mos = df[df["Compound"].str.lower()=="moschamine"]

print(mos.to\_string(index=False) if not mos.empty else "Moschamine not found")

df = pd.read\_csv("per\_compound\_errors.csv")

df = df.sort\_values("RMSE\_i", ascending=False).reset\_index(drop=True)

# rank by RMSE (worst = rank 1)

df["RMSE\_rank"] = df.index + 1

mos\_rank = df[df["Compound"].str.lower()=="moschamine"]

print(mos\_rank.to\_string(index=False) if not mos\_rank.empty else "Moschamine not found")

# If you prefer rank by Predicted\_Score descending:

df2 = pd.read\_csv("per\_compound\_errors.csv").sort\_values("Predicted\_Score", ascending=False).reset\_index(drop=True)

df2["Rank\_by\_score"] = df2.index + 1

print(df2[df2["Compound"].str.lower()=="moschamine"].to\_string(index=False) if "moschamine" in df2["Compound"].str.lower().values else "Moschamine not found")

import pandas as pd, matplotlib.pyplot as plt

df = pd.read\_csv("per\_compound\_errors.csv")

plt.figure(figsize=(6,4))

plt.scatter(df["Interaction\_Score\_true"], df["Error"], alpha=0.8)

plt.axhline(0, color='k', linestyle='--')

plt.xlabel("True Interaction Score"); plt.ylabel("Error (pred - true)")

plt.title("Residuals vs True Score"); plt.show()

plt.figure(figsize=(6,4))

plt.hist(df["Error"], bins=25, edgecolor='k')

plt.xlabel("Error"); plt.title("Error distribution"); plt.show()

import pandas as pd

orig = pd.read\_csv("Preprocessed Data.txt", sep=None, engine="python")

orig.columns = [c.strip() for c in orig.columns]

for cand in ["Compound Name","Compound","compound"]:

if cand in orig.columns:

orig = orig.rename(columns={cand:"Compound"}); break

worst = pd.read\_csv("per\_compound\_errors.csv").sort\_values("RMSE\_i", ascending=False).head(6)["Compound"].tolist()

print("Worst compounds:", worst)

print(orig[orig["Compound"].isin(worst)].to\_string(index=False))

import pandas as pd, numpy as np

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# load numeric features used previously (X\_numeric) and df\_clean or recreate numeric set:

X = pd.read\_csv("X\_numeric.csv", index\_col=0) if os.path.exists("X\_numeric.csv") else None

# if X not saved, create numeric matrix as before (see earlier snippets)

# Drop PCID if present

if X is not None:

X2 = X.drop(columns=[c for c in X.columns if c.lower()=="pcid"], errors='ignore')

y = pd.read\_csv("per\_compound\_errors.csv")["Interaction\_Score\_true"] # align externally if needed

model = RandomForestRegressor(n\_estimators=300, random\_state=42)

model.fit(X2, y)

preds = model.predict(X2)

mse = mean\_squared\_error(y, preds)

print("RMSE after dropping PCID:", mse\*\*0.5)

else:

print("No X\_numeric.csv found; recreate numeric matrix then re-run this block.")

# Why Moschamine is best

import pandas as pd

df = pd.read\_csv("Preprocessed Data.txt", sep=None, engine="python")

df.columns = [c.strip() for c in df.columns]

for cand in ["Compound Name","Compound","compound"]:

if cand in df.columns:

df = df.rename(columns={cand:"Compound"}); break

targets = ["Moschamine"]

# replace with top predicted compounds if you have the ranked CSV

# ranked = pd.read\_csv("ranked\_predictions\_export.csv")

# targets += ranked["Compound"].head(5).tolist()

cols = ["Water Solubility","Caco2 Permeability","Intestinal Absorption (%)","BBB Permeability","Total Clearance","AMES Toxicity","hERG I Inhibitor","hERG II Inhibitor","Hepatotoxicity","P-gp Substrate","CYP3A4 Substrate","CYP Inhibitors"]

out = df[df["Compound"].isin(targets)][["Compound"]+ [c for c in cols if c in df.columns]].reset\_index(drop=True)

print(out.to\_string(index=False))