# step 1

#1

import pandas as pd

csv\_path = r"C:\Users\lenovo\Desktop\MDSProject\数据集大修.csv"

pickle\_path = r"C:\Users\lenovo\Desktop\MDSProject\数据集大修.pkl"

try:

df = pd.read\_csv(csv\_path, encoding='utf-8')

except UnicodeDecodeError:

print("⚠️ utf-8解码失败，改用更宽容的模式")

df = pd.read\_csv(csv\_path, encoding='utf-8', encoding\_errors='replace')

df["created\_at 推文发布时间"] = pd.to\_datetime(df["created\_at 推文发布时间"], errors='coerce')

df["account\_created\_at 账户创建时间"] = pd.to\_datetime(df["account\_created\_at 账户创建时间"], errors='coerce')

df.to\_pickle(pickle\_path)

print("CSV 已成功转为 pkl 文件。")

#2

import pandas as pd

import re

pkl\_path = r"C:\Users\lenovo\Desktop\MDSProject\数据清洗和预处理\数据集大修.pkl"

df = pd.read\_pickle(pkl\_path)

print(f"成功读取数据，共 {df.shape[0]} 条")

def clean\_text(text):

if not isinstance(text, str):

return ""

text = text.lower()

text = re.sub(r"http\S+", "", text)

text = re.sub(r"@\w+", "", text)

text = re.sub(r"#\w+", "", text)

text = re.sub(r"\s+", " ", text)

text = re.sub(r"[^\x00-\x7F]+", "", text)

return text.strip()

text\_col = "text 推文文本内容"

if text\_col not in df.columns:

raise ValueError(f"❗️ 列 '{text\_col}' 不存在，请检查数据！")

df['cleaned\_text'] = df[text\_col].apply(clean\_text)

print("\n 清洗后示例：")

print(df[[text\_col, 'cleaned\_text']].head())

save\_path = r"C:\Users\lenovo\Desktop\MDSProject\数据集大修\_cleaned.pkl"

df.to\_pickle(save\_path)

print(f"\n 已保存到：{save\_path}")

#3

import pandas as pd

df = pd.read\_pickle("数据集大修\_cleaned.pkl")

df['created\_at 推文发布时间'] = pd.to\_datetime(df['created\_at 推文发布时间'], errors='coerce')

start\_date = pd.Timestamp("2017-01-01")

end\_date = pd.Timestamp("2020-12-31")

df\_filtered = df[(df['created\_at 推文发布时间'] >= start\_date) & (df['created\_at 推文发布时间'] <= end\_date)]

print(f"过滤后数据总量：{df\_filtered.shape[0]} 条")

print(df\_filtered['created\_at 推文发布时间'].describe())

df\_filtered.to\_csv("filtereddata\_2017\_2020.csv", index=False)

#4

import pandas as pd

df\_filtered = pd.read\_csv("filtereddata\_2017\_2020.csv")

org\_counts = df\_filtered['organization 组织名称'].value\_counts().reset\_index()

org\_counts.columns = ['organization 组织名称', 'tweet\_count']

df\_merged = df\_filtered.merge(org\_counts, on='organization 组织名称', how='left')

def stratified\_sample(group):

count = group['tweet\_count'].iloc[0]

if count >= 1000:

n = min(100, len(group))

elif 200 <= count < 1000:

n = min(50, len(group))

elif 50 <= count < 200:

n = min(30, len(group))

else:

n = len(group)

return group.sample(n=n, random\_state=42)

df\_sampled = df\_merged.groupby('organization 组织名称', group\_keys=False).apply(stratified\_sample)

print(f"抽样后数据总量：{df\_sampled.shape[0]} 条")

print(f"涉及组织数量：{df\_sampled['organization 组织名称'].nunique()} 个")

df\_sampled.to\_csv("sampleddataforValuesML.csv", index=False)

#5

import pandas as pd

import torch

from transformers import AutoTokenizer, AutoConfig, AutoModelForSequenceClassification, Trainer, TrainingArguments, DataCollatorWithPadding

df = pd.read\_csv("sampleddataforValuesML.csv")

label\_cols = [

'values\_se',

'values\_co',

'values\_tr',

'values\_be',

'values\_un',

'values\_sd',

'values\_st',

'values\_he',

'values\_ac',

'values\_po'

]

text\_col = 'cleaned\_text'

print(f" 确认文本列: {text\_col}")

print(f" 确认标签列: {label\_cols}")

df = df.sample(frac=1.0, random\_state=42).reset\_index(drop=True)

train\_frac = 0.8

train\_size = int(train\_frac \* len(df))

train\_df = df.iloc[:train\_size]

val\_df = df.iloc[train\_size:]

X\_train = train\_df[text\_col].tolist()

X\_val = val\_df[text\_col].tolist()

y\_train = train\_df[label\_cols].values

y\_val = val\_df[label\_cols].values

tokenizer = AutoTokenizer.from\_pretrained("roberta-base")

train\_encodings = tokenizer(X\_train, truncation=True, max\_length=256)

val\_encodings = tokenizer(X\_val, truncation=True, max\_length=256)

class ValueDataset(torch.utils.data.Dataset):

def \_\_init\_\_(self, encodings, labels):

self.encodings = encodings

self.labels = labels

def \_\_len\_\_(self):

return len(self.labels)

def \_\_getitem\_\_(self, idx):

item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

item["labels"] = torch.tensor(self.labels[idx], dtype=torch.float)

return item

train\_dataset = ValueDataset(train\_encodings, y\_train)

val\_dataset = ValueDataset(val\_encodings, y\_val)

config = AutoConfig.from\_pretrained(

"roberta-base",

num\_labels=len(label\_cols),

problem\_type="multi\_label\_classification"

)

model = AutoModelForSequenceClassification.from\_pretrained(

"roberta-base",

config=config

)

pos\_counts = y\_train.sum(axis=0)

total = len(y\_train)

pos\_weights = []

for count in pos\_counts:

if count == 0:

pos\_weights.append(1.0)

else:

neg\_count = total - count

pos\_weights.append(neg\_count / float(count))

pos\_weights = torch.tensor(pos\_weights, dtype=torch.float)

class WeightedTrainer(Trainer):

def compute\_loss(self, model, inputs, return\_outputs=False):

labels = inputs.get("labels").to(model.device)

outputs = model(\*\*inputs)

logits = outputs.logits

loss\_fn = torch.nn.BCEWithLogitsLoss(pos\_weight=pos\_weights.to(model.device))

loss = loss\_fn(logits, labels)

return (loss, outputs) if return\_outputs else loss

training\_args = TrainingArguments(

output\_dir="value\_model\_output",

overwrite\_output\_dir=True,

learning\_rate=2e-5,

per\_device\_train\_batch\_size=4,

per\_device\_eval\_batch\_size=4,

num\_train\_epochs=3,

evaluation\_strategy="epoch",

save\_strategy="epoch",

logging\_strategy="epoch",

logging\_dir="logs",

load\_best\_model\_at\_end=False,

fp16=True,

seed=42

)

trainer = WeightedTrainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=val\_dataset,

data\_collator=DataCollatorWithPadding(tokenizer=tokenizer),

)

trainer.train()

pred\_outputs = trainer.predict(val\_dataset)

logits = pred\_outputs.predictions

probs = torch.sigmoid(torch.tensor(logits))

pred\_labels = (probs.numpy() >= 0.5).astype(int)

pred\_df = pd.DataFrame(pred\_labels, columns=label\_cols)

pred\_df.insert(0, text\_col, val\_df[text\_col].values)

pred\_df.to\_csv("predicted\_values.csv", index=False)

print(" 预测结果已保存到 predicted\_values.csv")

probs\_df = pd.DataFrame(probs.numpy(), columns=label\_cols)

probs\_df.insert(0, text\_col, val\_df[text\_col].values)

probs\_df.to\_csv("predicted\_values\_probs.csv", index=False)

print(" 概率结果已保存到 predicted\_values\_probs.csv")

# Step 2

# 1

import pandas as pd

df = pd.read\_csv("merged\_results.csv")

print(df.columns)

# 2

PVD\_columns = [col for col in df.columns if col.endswith('\_PVD')]

ValuesML\_columns = [col for col in df.columns if col.endswith('\_ValuesML')]

org\_col = 'organization 组织名称'

pvd\_org = df.groupby(org\_col)[PVD\_columns].mean().reset\_index()

valuesml\_org = df.groupby(org\_col)[ValuesML\_columns].mean().reset\_index()

pvd\_org.columns = [col.replace('\_PVD','') for col in pvd\_org.columns]

valuesml\_org.columns = [col.replace('\_ValuesML','') for col in valuesml\_org.columns]

org\_values = pd.merge(

pvd\_org,

valuesml\_org,

on='organization 组织名称',

suffixes=('\_PVD', '\_ValuesML')

)

org\_values.to\_csv('organization\_level\_values.csv', index=False)

print(org\_values.head())

# 3

like\_cols = ['favorite\_count 点赞数', 'retweet\_count 转发数', 'listed\_count 被收录于他人列表的次数']

org\_metrics = df.groupby(org\_col)[like\_cols].mean().reset\_index()

org\_metrics.to\_csv('organization\_level\_social\_metrics.csv', index=False)

print(org\_metrics.head())

aggregation\_dict = {

'followers 组织粉丝数': 'first',

'verified 是否为认证账户': 'first',

'engage 互动评分（赞转评）': 'mean',

'rating.Financial 财务评分': 'first',

'rating.Accountability & Transparency 责任与透明度评分': 'first',

'rating.Overall Score & Rating 总评分': 'first',

'financialPerformance.Fundraising Efficiency 筹款效率': 'first',

'financialPerformance.Working Capital Ratio 营运资本比': 'first',

'financialPerformance.Program Expenses Growth 项目支出增长率': 'first',

'financialPerformance.Liabilities to Assets 负债资产比': 'first',

'incomeStatement.Government Grants 政府拨款': 'first',

'incomeStatement.Total Contributions 总捐赠': 'first',

'incomeStatement.Program Service Revenue 服务型收入': 'first',

'incomeStatement.TOTAL REVENUE 总收入': 'first',

'incomeStatement.Program Expenses 项目支出': 'first',

'incomeStatement.Administrative Expenses 行政支出': 'first',

'incomeStatement.Net Assets 净资产': 'first',

'incomeStatement.Excess (or Deficit) for the year 年度盈余/赤字': 'first'

}

org\_info = df.groupby(org\_col).agg(aggregation\_dict).reset\_index()

org\_info.to\_csv('organization\_level\_info.csv', index=False)

print(org\_info.head())

# 4

org\_values = pd.read\_csv('organization\_level\_values.csv')

org\_metrics = pd.read\_csv('organization\_level\_social\_metrics.csv')

org\_info = pd.read\_csv('organization\_level\_info.csv')

final\_org\_level = org\_values.merge(org\_info, on='organization 组织名称', how='left')

final\_org\_level = final\_org\_level.merge(org\_metrics, on='organization 组织名称', how='left')

final\_org\_level.to\_csv('organization\_level\_final\_data.csv', index=False)

print(final\_org\_level.describe())

#Step 3 EDA

# 1

import pandas as pd

df = pd.read\_csv('organization\_level\_final\_data.csv')

# 2

pvd\_cols = [col for col in df.columns if col.endswith('\_PVD')]

ml\_cols = [col for col in df.columns if col.endswith('\_ValuesML')]

org\_feature\_cols = [

'followers 组织粉丝数',

'verified 是否为认证账户',

'engage 互动评分（赞转评）',

'rating.Financial 财务评分',

'rating.Accountability & Transparency 责任与透明度评分',

'rating.Overall Score & Rating 总评分',

'financialPerformance.Fundraising Efficiency 筹款效率',

'financialPerformance.Working Capital Ratio 营运资本比',

'financialPerformance.Program Expenses Growth 项目支出增长率',

'financialPerformance.Liabilities to Assets 负债资产比',

'incomeStatement.Government Grants 政府拨款',

'incomeStatement.Total Contributions 总捐赠',

'incomeStatement.Program Service Revenue 服务型收入',

'incomeStatement.TOTAL REVENUE 总收入',

'incomeStatement.Program Expenses 项目支出',

'incomeStatement.Administrative Expenses 行政支出',

'incomeStatement.Net Assets 净资产',

'incomeStatement.Excess (or Deficit) for the year 年度盈余/赤字',

'favorite\_count 点赞数',

'retweet\_count 转发数',

'listed\_count 被收录于他人列表的次数'

]

# 3

print(df[org\_feature\_cols].describe())

print(df[pvd\_cols + ml\_cols].describe())

import matplotlib.pyplot as plt

df\_org\_plot = df[org\_feature\_cols].copy()

df\_org\_plot.columns = ['\_'.join(col.split()[:-1]) for col in df\_org\_plot.columns] df\_org\_plot.hist(bins=30, figsize=(15,12))

plt.tight\_layout()

plt.show()

value\_dims = ['se', 'co', 'tr', 'be', 'un', 'sd', 'st', 'he', 'ac', 'po']

fig, axs = plt.subplots(2, 5, figsize=(20, 8))

axs = axs.flatten()

for i, dim in enumerate(value\_dims):

pvd\_col = f'values\_{dim}\_PVD'

ml\_col = f'values\_{dim}\_ValuesML'

axs[i].hist(df[pvd\_col], bins=30, alpha=0.6, label='PVD', color='tab:blue')

axs[i].hist(df[ml\_col], bins=30, alpha=0.6, label='ValuesML', color='tab:orange')

axs[i].set\_title(f'{dim.upper()} - Value Distribution')

axs[i].legend()

plt.tight\_layout()

plt.show()

# 4

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

log\_transform\_cols = [

'followers 组织粉丝数',

'favorite\_count 点赞数',

'retweet\_count 转发数',

'listed\_count 被收录于他人列表的次数',

'incomeStatement.Government Grants 政府拨款',

'incomeStatement.Total Contributions 总捐赠',

'incomeStatement.Program Service Revenue 服务型收入',

'incomeStatement.TOTAL REVENUE 总收入',

'incomeStatement.Program Expenses 项目支出',

'incomeStatement.Administrative Expenses 行政支出',

'incomeStatement.Net Assets 净资产',

'incomeStatement.Excess (or Deficit) for the year 年度盈余/赤字',

'financialPerformance.Fundraising Efficiency 筹款效率',

'financialPerformance.Working Capital Ratio 营运资本比',

'financialPerformance.Program Expenses Growth 项目支出增长率',

'financialPerformance.Liabilities to Assets 负债资产比'

]

no\_log\_cols = [

'verified 是否为认证账户',

'engage 互动评分（赞转评）',

'rating.Financial 财务评分',

'rating.Accountability & Transparency 责任与透明度评分',

'rating.Overall Score & Rating 总评分'

]

for col in log\_transform\_cols:

df[col] = df[col].astype(str)

df[col] = df[col].str.replace('< \$0.01', '0.005', regex=True)

df[col] = df[col].str.replace('< 0.01', '0.005', regex=True)

df[col] = df[col].str.replace('< 0.1%', '0.0005', regex=True)

df[col] = df[col].str.replace('[\$, %]', '', regex=True)

df[col] = pd.to\_numeric(df[col], errors='coerce')

def signed\_log1p(x):

return np.sign(x) \* np.log1p(np.abs(x))

for col in log\_transform\_cols:

df[f'log\_{col}'] = df[col].apply(signed\_log1p)

standardize\_cols = [f'log\_{col}' for col in log\_transform\_cols] + no\_log\_cols

X = df[standardize\_cols]

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(X)

scaled\_df = pd.DataFrame(

scaled\_features,

columns=[f'std\_{col}' for col in standardize\_cols]

)

scaled\_df['organization 组织名称'] = df['organization 组织名称']

scaled\_df.to\_csv('organization\_level\_features\_scaled.csv', index=False)

print("处理完成，已保存文件：organization\_level\_features\_scaled.csv")

#5

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

log\_cols = [f'log\_{col}' for col in log\_transform\_cols]

direct\_cols = no\_log\_cols

final\_feature\_cols = log\_cols + direct\_cols

available\_cols = [col for col in final\_feature\_cols if col in df.columns]

heatmap\_cols = [col.split()[0] for col in available\_cols]

df\_heatmap = df[available\_cols].copy()

df\_heatmap.columns = heatmap\_cols

plt.figure(figsize=(14, 12))

sns.heatmap(df\_heatmap.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)

plt.tight\_layout()

plt.show()

X = df[available\_cols].copy().astype(float)

vif\_data = pd.DataFrame()

vif\_data["feature"] = available\_cols

vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

vif\_data["feature"] = vif\_data["feature"].apply(lambda x: x.split()[0])

print("\n VIF result：")

print(vif\_data)

#Step 4

#1 PCA

import re

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

X = org\_df[selected\_cols]

pca = PCA()

X\_pca = pca.fit\_transform(X)

plt.figure(figsize=(8, 5))

plt.plot(

range(1, len(pca.explained\_variance\_ratio\_) + 1),

np.cumsum(pca.explained\_variance\_ratio\_),

marker='o'

)

plt.xlabel('Number of Components')

plt.ylabel('Cumulative Explained Variance')

plt.title('Scree Plot')

plt.grid(True)

plt.show()

def strip\_std\_log\_prefix(name: str) -> str:

return re.sub(r'^(?:std|log)(?:[\_.-]+(?:std|log))\*[\_.-]\*', '', name, flags=re.IGNORECASE)

def remove\_chinese\_tail(name: str) -> str:

m = re.search(r'[\u4e00-\u9fff]', name)

if m:

name = name[:m.start()].rstrip()

name = re.sub(r'\s+', ' ', name).strip()

return name

def clean\_name(name: str) -> str:

return remove\_chinese\_tail(strip\_std\_log\_prefix(name))

english\_names = [clean\_name(col) for col in selected\_cols]

def ensure\_unique(seq):

seen, out = {}, []

for s in seq:

k = seen.get(s, 0)

out.append(s if k == 0 else f"{s}\_\_{k+1}")

seen[s] = k + 1

return out

english\_names = ensure\_unique(english\_names)

loadings = pd.DataFrame(

pca.components\_.T,

columns=[f'PC{i+1}' for i in range(pca.n\_components\_)],

index=english\_names

)

abs\_load = loadings.abs()

variable\_summary = pd.DataFrame({

'MostRelatedPC': abs\_load.idxmax(axis=1),

'MaxAbsLoading': abs\_load.max(axis=1)

}, index=english\_names)

variable\_summary['KeepThisVariable'] = variable\_summary['MostRelatedPC'].apply(

lambda pc: int(pc.replace('PC', '')) <= 10

)

print(" In which principal component each variable contributes the most and whether it should be retained:")

display(variable\_summary)

#2 EFA

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from factor\_analyzer import FactorAnalyzer, calculate\_kmo, calculate\_bartlett\_sphericity

df = pd.read\_csv("organization\_level\_features\_scaled.csv")

df = df.dropna()

selected\_cols = [

'std\_log\_followers 组织粉丝数', 'std\_verified 是否为认证账户', 'std\_engage 互动评分（赞转评）',

'std\_log\_favorite\_count 点赞数', 'std\_log\_retweet\_count 转发数', 'std\_log\_listed\_count 被收录于他人列表的次数',

'std\_rating.Financial 财务评分', 'std\_rating.Accountability & Transparency 责任与透明度评分', 'std\_rating.Overall Score & Rating 总评分',

'std\_log\_incomeStatement.Government Grants 政府拨款', 'std\_log\_incomeStatement.Total Contributions 总捐赠',

'std\_log\_incomeStatement.Program Service Revenue 服务型收入', 'std\_log\_incomeStatement.TOTAL REVENUE 总收入',

'std\_log\_incomeStatement.Program Expenses 项目支出', 'std\_log\_incomeStatement.Administrative Expenses 行政支出',

'std\_log\_incomeStatement.Net Assets 净资产', 'std\_log\_incomeStatement.Excess (or Deficit) for the year 年度盈余/赤字',

'std\_log\_financialPerformance.Fundraising Efficiency 筹款效率', 'std\_log\_financialPerformance.Working Capital Ratio 营运资本比',

'std\_log\_financialPerformance.Program Expenses Growth 项目支出增长率', 'std\_log\_financialPerformance.Liabilities to Assets 负债资产比',

]

col\_labels\_en = [

'Followers', 'Verified', 'Engagement',

'Favorite Count', 'Retweet Count', 'Listed Count',

'Financial Rating', 'Accountability Rating', 'Overall Rating',

'Gov Grants', 'Donations', 'Program Revenue',

'Total Revenue', 'Program Expenses', 'Admin Expenses',

'Net Assets', 'Annual Surplus', 'Fundraising Efficiency',

'Capital Ratio', 'Expense Growth', 'Liability Ratio'

]

df\_selected = df[selected\_cols]

kmo\_all, kmo\_model = calculate\_kmo(df\_selected)

bartlett\_chi, bartlett\_p = calculate\_bartlett\_sphericity(df\_selected)

print(f"KMO: {round(kmo\_model, 3)}, Bartlett's Test p: {round(bartlett\_p, 4)}")

fa = FactorAnalyzer(rotation=None)

fa.fit(df\_selected)

ev, v = fa.get\_eigenvalues()

plt.figure(figsize=(8, 4))

plt.plot(range(1, len(ev)+1), ev, marker='o')

plt.axhline(y=1, color='red', linestyle='--')

plt.title('Scree Plot')

plt.xlabel('Factor Number')

plt.ylabel('Eigenvalue')

plt.grid(True)

plt.tight\_layout()

plt.show()

fa = FactorAnalyzer(n\_factors=5, rotation='varimax')

fa.fit(df\_selected)

loadings = pd.DataFrame(fa.loadings\_, index=col\_labels\_en)

loadings.columns = [f"Factor{i+1}" for i in range(loadings.shape[1])]

plt.figure(figsize=(12, 8))

sns.heatmap(loadings, annot=True, cmap='coolwarm', center=0)

plt.title("Factor Loadings Heatmap")

plt.tight\_layout()

plt.show()

group\_assignments = loadings.abs().idxmax(axis=1)

grouped = pd.DataFrame({'Variable (EN)': loadings.index, 'Assigned Factor': group\_assignments})

print(grouped)

# 3

exposure\_vars = [

'std\_log\_followers 组织粉丝数',

'std\_verified 是否为认证账户',

'std\_log\_favorite\_count 点赞数',

'std\_log\_retweet\_count 转发数',

'std\_log\_listed\_count 被收录于他人列表的次数'

]

credibility\_vars = [

'std\_rating.Financial 财务评分',

'std\_rating.Accountability & Transparency 责任与透明度评分',

'std\_rating.Overall Score & Rating 总评分'

]

financial\_size\_vars = [

'std\_log\_incomeStatement.Government Grants 政府拨款',

'std\_log\_incomeStatement.Total Contributions 总捐赠',

'std\_log\_incomeStatement.TOTAL REVENUE 总收入',

'std\_log\_incomeStatement.Program Expenses 项目支出',

'std\_log\_incomeStatement.Administrative Expenses 行政支出',

]

sustainability\_vars = [

'std\_log\_incomeStatement.Net Assets 净资产',

'std\_log\_financialPerformance.Working Capital Ratio 营运资本比',

]

from sklearn.preprocessing import StandardScaler

def strip\_chinese(colname):

return colname.split()[0]

exp\_cols = [f"zz\_{strip\_chinese(col)}" for col in exposure\_vars]

org\_df[exp\_cols] = org\_df[exposure\_vars].values

size\_cols = [f"zz\_{strip\_chinese(col)}" for col in financial\_size\_vars]

org\_df[size\_cols] = org\_df[financial\_size\_vars].values

cred\_cols = [f"zz\_{strip\_chinese(col)}" for col in credibility\_vars]

org\_df[cred\_cols] = org\_df[credibility\_vars].values

sust\_cols = [f"zz\_{strip\_chinese(col)}" for col in sustainability\_vars]

org\_df[sust\_cols] = org\_df[sustainability\_vars].values

org\_df['Exposure\_Index'] = org\_df[exp\_cols].mean(axis=1)

org\_df['FinancialSize\_Index'] = org\_df[size\_cols].mean(axis=1)

org\_df['Credibility\_Index'] = org\_df[cred\_cols].mean(axis=1)

org\_df['FinancialSustainability\_Index'] = org\_df[sust\_cols].mean(axis=1)

final\_cols = [

'organization 组织名称',

'Exposure\_Index',

'FinancialSize\_Index',

'Credibility\_Index',

'FinancialSustainability\_Index'

]

final\_df = org\_df[final\_cols]

print(final\_df.head())

sns.heatmap(final\_df.drop(columns='organization 组织名称').corr(), annot=True, cmap='coolwarm')

plt.title('Correlation between Final Indices')

plt.show()

final\_df.to\_csv('organization\_4\_levels.csv', index=False)

#4

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from pingouin import cronbach\_alpha

df = pd.read\_csv("organization\_level\_features\_scaled.csv")

column\_name\_map = {

'std\_log\_followers 组织粉丝数': 'Followers',

'std\_verified 是否为认证账户': 'Verified',

'std\_engage 互动评分（赞转评）': 'Engagement',

'std\_log\_favorite\_count 点赞数': 'Favorite Count',

'std\_log\_retweet\_count 转发数': 'Retweet Count',

'std\_log\_listed\_count 被收录于他人列表的次数': 'Listed Count',

'std\_rating.Financial 财务评分': 'Financial Rating',

'std\_rating.Accountability & Transparency 责任与透明度评分': 'Accountability Rating',

'std\_rating.Overall Score & Rating 总评分': 'Overall Rating',

'std\_log\_incomeStatement.Government Grants 政府拨款': 'Gov Grants',

'std\_log\_incomeStatement.Total Contributions 总捐赠': 'Donations',

'std\_log\_incomeStatement.Program Service Revenue 服务型收入': 'Program Revenue',

'std\_log\_incomeStatement.TOTAL REVENUE 总收入': 'Total Revenue',

'std\_log\_incomeStatement.Program Expenses 项目支出': 'Program Expenses',

'std\_log\_incomeStatement.Administrative Expenses 行政支出': 'Admin Expenses',

'std\_log\_incomeStatement.Net Assets 净资产': 'Net Assets',

'std\_log\_incomeStatement.Excess (or Deficit) for the year 年度盈余/赤字': 'Annual Surplus',

'std\_log\_financialPerformance.Fundraising Efficiency 筹款效率': 'Fundraising Efficiency',

'std\_log\_financialPerformance.Working Capital Ratio 营运资本比': 'Capital Ratio',

'std\_log\_financialPerformance.Program Expenses Growth 项目支出增长率': 'Expense Growth',

'std\_log\_financialPerformance.Liabilities to Assets 负债资产比': 'Liability Ratio',

}

factor\_groups = {

'financial\_size': [

'std\_log\_incomeStatement.Total Contributions 总捐赠',

'std\_log\_incomeStatement.TOTAL REVENUE 总收入',

'std\_log\_incomeStatement.Program Expenses 项目支出',

'std\_log\_incomeStatement.Administrative Expenses 行政支出',

'std\_log\_incomeStatement.Government Grants 政府拨款',

],

'exposure': [

'std\_log\_followers 组织粉丝数',

'std\_verified 是否为认证账户',

'std\_log\_favorite\_count 点赞数',

'std\_log\_listed\_count 被收录于他人列表的次数',

'std\_log\_retweet\_count 转发数',

],

'credibility': [

'std\_rating.Financial 财务评分',

'std\_rating.Accountability & Transparency 责任与透明度评分',

'std\_rating.Overall Score & Rating 总评分',

],

'asset\_structure': [

'std\_log\_financialPerformance.Working Capital Ratio 营运资本比',

'std\_log\_incomeStatement.Net Assets 净资产',

]

}

alpha\_results = {}

for name, cols in factor\_groups.items():

sub\_df = df[cols].dropna()

alpha, \_ = cronbach\_alpha(sub\_df)

alpha\_results[name] = round(alpha, 2)

alpha\_df = pd.DataFrame.from\_dict(alpha\_results, orient='index', columns=['Cronbach Alpha'])

alpha\_df = alpha\_df.sort\_values('Cronbach Alpha', ascending=True)

plt.figure(figsize=(10, 6))

sns.barplot(x='Cronbach Alpha', y=alpha\_df.index, data=alpha\_df, palette='Blues\_d')

for i, val in enumerate(alpha\_df['Cronbach Alpha']):

plt.text(val + 0.01, i, str(val), va='center')

plt.title("Internal Consistency of Five Influence Factors (Cronbach's Alpha)", fontsize=14)

plt.xlabel("Cronbach Alpha")

plt.ylabel("Factor Category")

plt.xlim(0, 1)

plt.tight\_layout()

plt.show()

#Step 5

# 1PVD

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import networkx as nx

sns.set(style="whitegrid", font\_scale=1.2)

df = pd.read\_csv("organization\_level\_full\_analysis\_data.csv")

influence\_cols = ['Exposure\_Index', 'FinancialSize\_Index', 'Credibility\_Index', 'FinancialSustainability\_Index']

pvd\_cols = ['values\_se\_PVD', 'values\_co\_PVD', 'values\_tr\_PVD', 'values\_be\_PVD', 'values\_un\_PVD', 'values\_sd\_PVD', 'values\_st\_PVD', 'values\_po\_PVD', 'values\_he\_PVD', 'values\_ac\_PVD']

corr\_matrix = pd.DataFrame(index=influence\_cols, columns=pvd\_cols)

for factor in influence\_cols:

for pvd in pvd\_cols:

corr\_matrix.loc[factor, pvd] = df[factor].corr(df[pvd])

corr\_matrix = corr\_matrix.astype(float)

print("\n=== Pearson Correlation Matrix ===")

print(corr\_matrix)

corr\_matrix.to\_csv('PVD\_correlation.csv')

plt.figure(figsize=(12, 8))

sns.heatmap(corr\_matrix, annot=True, fmt=".2f", cmap='coolwarm', center=0, linewidths=0.5, cbar\_kws={'label': 'Correlation'})

plt.title("Pearson Correlation Heatmap: Influence Factors vs PVD Dimensions")

plt.xlabel("PVD Dimensions")

plt.ylabel("Influence Factors")

plt.tight\_layout()

plt.show()

# 2 ValuesML

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import networkx as nx

sns.set(style="whitegrid", font\_scale=1.2)

influence\_cols = ['Exposure\_Index', 'FinancialSize\_Index', 'Credibility\_Index', 'FinancialSustainability\_Index']

valuesml\_cols = ['values\_se\_ValuesML', 'values\_co\_ValuesML', 'values\_tr\_ValuesML', 'values\_be\_ValuesML', 'values\_un\_ValuesML',

'values\_sd\_ValuesML', 'values\_st\_ValuesML', 'values\_po\_ValuesML', 'values\_he\_ValuesML', 'values\_ac\_ValuesML']

corr\_matrix\_ml = pd.DataFrame(index=influence\_cols, columns=valuesml\_cols)

for factor in influence\_cols:

for val\_ml in valuesml\_cols:

corr\_matrix\_ml.loc[factor, val\_ml] = df[factor].corr(df[val\_ml])

corr\_matrix\_ml = corr\_matrix\_ml.astype(float)

print("\n=== Pearson Correlation Matrix (ValuesML) ===")

print(corr\_matrix\_ml)

corr\_matrix\_ml.to\_csv('ValuesML\_correlation.csv')

plt.figure(figsize=(12, 8))

sns.heatmap(corr\_matrix\_ml, annot=True, fmt=".2f", cmap='coolwarm', center=0, linewidths=0.5, cbar\_kws={'label': 'Correlation'})

plt.title("Pearson Correlation Heatmap: Influence Factors vs ValuesML Dimensions")

plt.xlabel("ValuesML Dimensions")

plt.ylabel("Influence Factors")

plt.tight\_layout()

plt.show()

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.colors import Normalize

from matplotlib.cm import ScalarMappable

# 3 PVD

data = {

'Influence\_Factor': ['Exposure\_Index'] \* 10 + ['FinancialSize\_Index'] \* 10 +

['Credibility\_Index'] \* 10 + ['FinancialSustainability\_Index'] \* 10,

'Value\_Dimension': ['values\_se\_PVD', 'values\_co\_PVD', 'values\_tr\_PVD', 'values\_be\_PVD', 'values\_un\_PVD',

'values\_sd\_PVD', 'values\_st\_PVD', 'values\_po\_PVD', 'values\_he\_PVD', 'values\_ac\_PVD'] \* 4,

'Correlation': [

0.15, 0.08, 0.06, 0.06, 0.24, 0.09, 0.05, 0.12, -0.09, -0.09,

0.04, 0.05, 0.04, 0.16, 0.11, 0.06, 0.05, 0.08, 0.04, 0.03,

0.09, 0.05, -0.02, -0.03, 0.06, 0.07, -0.09, 0.09, -0.07, 0.01,

-0.05, -0.07, -0.05, -0.02, -0.06, 0.02, -0.14, -0.05, 0.10, -0.11

]

}

df\_corr = pd.DataFrame(data)

sns.set(style="whitegrid")

vmin = df\_corr['Correlation'].min()

vmax = df\_corr['Correlation'].max()

norm = Normalize(vmin=vmin, vmax=vmax)

cmap = plt.get\_cmap('coolwarm')

def barh\_with\_value\_colormap(data, y\_order=None, \*\*kwargs):

y = data['Value\_Dimension']

x = data['Correlation']

if y\_order is not None:

y = pd.Categorical(y, categories=y\_order, ordered=True)

data = data.copy()

data['Value\_Dimension'] = y

data = data.sort\_values('Value\_Dimension')

y = data['Value\_Dimension']

x = data['Correlation']

colors = cmap(norm(x.values))

plt.barh(y, x, color=colors, edgecolor='none')

y\_order = df\_corr['Value\_Dimension'].unique()[::-1]

g = sns.FacetGrid(df\_corr, col="Influence\_Factor", col\_wrap=2, height=4, sharey=True, sharex=True)

g.map\_dataframe(barh\_with\_value\_colormap, y\_order=y\_order)

for ax in g.axes.flatten():

ax.axvline(0, color='black', linewidth=0.8, linestyle='--')

ax.set\_xlabel("Correlation Coefficient")

ax.set\_ylabel("")

g.set\_titles(col\_template="{col\_name}")

plt.tight\_layout()

plt.subplots\_adjust(top=0.90)

g.fig.suptitle("Correlation Between Influence Factors and PVD Value Dimensions", fontsize=16)

sm = ScalarMappable(norm=norm, cmap=cmap)

sm.set\_array([])

cbar = g.fig.colorbar(sm, ax=g.axes, shrink=0.9, pad=0.02)

cbar.set\_label("Correlation")

plt.show()

#4 ValuesML

data\_valuesml = {

'Influence\_Factor': ['Exposure\_Index'] \* 10 + ['FinancialSize\_Index'] \* 10 +

['Credibility\_Index'] \* 10 + ['FinancialSustainability\_Index'] \* 10,

'Value\_Dimension': ['values\_se\_ValuesML', 'values\_co\_ValuesML', 'values\_tr\_ValuesML', 'values\_be\_ValuesML', 'values\_un\_ValuesML',

'values\_sd\_ValuesML', 'values\_st\_ValuesML', 'values\_po\_ValuesML', 'values\_he\_ValuesML', 'values\_ac\_ValuesML'] \* 4,

'Correlation': [

0.23, 0.14, 0.12, 0.08, 0.20, 0.10, 0.07, 0.15, -0.05, -0.02,

0.08, 0.08, 0.09, 0.16, 0.10, 0.07, 0.09, 0.08, 0.07, 0.02,

0.07, 0.07, 0.02, 0.00, 0.04, 0.05, -0.05, 0.11, -0.06, 0.05,

-0.06, -0.10, -0.07, -0.02, -0.08, -0.05, -0.13, -0.07, 0.09, -0.09

]

}

df\_valuesml = pd.DataFrame(data\_valuesml)

sns.set(style="whitegrid")

vmin = df\_valuesml['Correlation'].min()

vmax = df\_valuesml['Correlation'].max()

norm = Normalize(vmin=vmin, vmax=vmax)

cmap = plt.get\_cmap('coolwarm')

def barh\_with\_value\_colormap(data, y\_order=None, \*\*kwargs):

y = data['Value\_Dimension']

x = data['Correlation']

if y\_order is not None:

y = pd.Categorical(y, categories=y\_order, ordered=True)

data = data.copy()

data['Value\_Dimension'] = y

data = data.sort\_values('Value\_Dimension')

y = data['Value\_Dimension']

x = data['Correlation']

colors = cmap(norm(x.values))

plt.barh(y, x, color=colors, edgecolor='none')

y\_order = df\_valuesml['Value\_Dimension'].unique()[::-1]

g = sns.FacetGrid(df\_valuesml, col="Influence\_Factor", col\_wrap=2, height=4, sharey=True, sharex=True)

g.map\_dataframe(barh\_with\_value\_colormap, y\_order=y\_order)

for ax in g.axes.flatten():

ax.axvline(0, color='black', linewidth=0.8, linestyle='--')

ax.set\_xlabel("Correlation Coefficient")

ax.set\_ylabel("")

g.set\_titles(col\_template="{col\_name}")

plt.tight\_layout()

plt.subplots\_adjust(top=0.90)

g.fig.suptitle("Correlation Between Influence Factors and Value Dimensions (ValuesML Model)", fontsize=16)

sm = ScalarMappable(norm=norm, cmap=cmap)

sm.set\_array([])

cbar = g.fig.colorbar(sm, ax=g.axes, shrink=0.9, pad=0.02)

cbar.set\_label("Correlation")

plt.show()

# 5 PVD

import pandas as pd

import numpy as np

import statsmodels.api as sm

import matplotlib.pyplot as plt

import seaborn as sns

influence\_cols = ['Exposure\_Index', 'FinancialSize\_Index', 'Credibility\_Index', 'FinancialSustainability\_Index']

pvd\_cols = [

'values\_co\_PVD', 'values\_tr\_PVD', 'values\_be\_PVD', 'values\_un\_PVD','values\_se\_PVD',

'values\_sd\_PVD', 'values\_st\_PVD', 'values\_po\_PVD', 'values\_he\_PVD', 'values\_ac\_PVD'

]

print("\n 自变量列（组织影响力四因子）:")

print(influence\_cols)

print("\n 因变量列（PVD的10个价值观维度）:")

print(pvd\_cols)

results\_list = []

for pvd\_var in pvd\_cols:

X = df[influence\_cols]

X = sm.add\_constant(X)

y\_raw = df[pvd\_var]

y = np.log1p(y\_raw)

model = sm.OLS(y, X).fit()

residuals = model.resid

fitted = model.fittedvalues

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

sns.histplot(residuals, bins=30, kde=True, ax=axes[0])

axes[0].axvline(0, color='red', linestyle='--', linewidth=1)

axes[0].set\_title(f'Residuals Histogram: {pvd\_var}')

axes[0].set\_xlabel('Residual')

axes[1].scatter(fitted, residuals, alpha=0.5)

axes[1].axhline(0, color='red', linestyle='--')

axes[1].set\_title(f'Residuals vs Fitted: {pvd\_var}')

axes[1].set\_xlabel('Fitted Values')

axes[1].set\_ylabel('Residuals')

plt.suptitle(f'Regression Diagnostics: {pvd\_var}', fontsize=16)

plt.tight\_layout(rect=[0, 0, 1, 0.95])

plt.show()

for predictor in ['const'] + influence\_cols:

results\_list.append({

'PVD\_Dimension': pvd\_var,

'Predictor': predictor,

'Coefficient': model.params[predictor],

'P-value': model.pvalues[predictor],

'StdErr': model.bse[predictor],

'R-squared': model.rsquared

})

results\_df = pd.DataFrame(results\_list)

print("\n 回归结果预览：")

print(results\_df.head(15))

r2\_summary = results\_df[['PVD\_Dimension', 'R-squared']].drop\_duplicates()

print("\n 自变量回归解释力（R平方）：")

print(r2\_summary)

results\_df.to\_csv('PVD\_regression.csv', index=False)

#6 ValuesML

import pandas as pd

import numpy as np

import statsmodels.api as sm

import matplotlib.pyplot as plt

import seaborn as sns

influence\_cols = ['Exposure\_Index', 'FinancialSize\_Index', 'Credibility\_Index', 'FinancialSustainability\_Index']

pvd\_cols = [

'values\_co\_ValuesML', 'values\_se\_ValuesML', 'values\_tr\_ValuesML', 'values\_be\_ValuesML', 'values\_un\_ValuesML',

'values\_sd\_ValuesML', 'values\_st\_ValuesML', 'values\_po\_ValuesML', 'values\_he\_ValuesML', 'values\_ac\_ValuesML'

]

print("\n自变量列（组织影响力四因子）:")

print(influence\_cols)

print("\n 因变量列（ValuesML的10个价值观维度）:")

print(pvd\_cols)

results\_list = []

for pvd\_var in pvd\_cols:

X = df[influence\_cols]

X = sm.add\_constant(X)

y\_raw = df[pvd\_var]

y = np.log1p(y\_raw)

model = sm.OLS(y, X).fit()

residuals = model.resid

fitted = model.fittedvalues

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

sns.histplot(residuals, bins=30, kde=True, ax=axes[0])

axes[0].axvline(0, color='red', linestyle='--', linewidth=1)

axes[0].set\_title(f'Residuals Histogram: {pvd\_var}')

axes[0].set\_xlabel('Residual')

axes[1].scatter(fitted, residuals, alpha=0.5)

axes[1].axhline(0, color='red', linestyle='--')

axes[1].set\_title(f'Residuals vs Fitted: {pvd\_var}')

axes[1].set\_xlabel('Fitted Values')

axes[1].set\_ylabel('Residuals')

plt.suptitle(f'Regression Diagnostics: {pvd\_var}', fontsize=16)

plt.tight\_layout(rect=[0, 0, 1, 0.95])

plt.show()

for predictor in ['const'] + influence\_cols:

results\_list.append({

'PVD\_Dimension': pvd\_var,

'Predictor': predictor,

'Coefficient': model.params[predictor],

'P-value': model.pvalues[predictor],

'StdErr': model.bse[predictor],

'R-squared': model.rsquared

})

results\_df = pd.DataFrame(results\_list)

print("\n回归结果预览:")

print(results\_df.head(15))

r2\_summary = results\_df[['PVD\_Dimension', 'R-squared']].drop\_duplicates()

print("\n 各ValuesML维度回归模型的R²:")

print(r2\_summary)

results\_df.to\_csv('ValuesML\_regression.csv', index=False)

#Step6

#1

import pandas as pd

import matplotlib.pyplot as plt

pvd\_full = pd.read\_csv('PVD\_regression.csv')

valuesml\_full = pd.read\_csv('ValuesML\_regression.csv')

pvd\_full['PVD\_Dimension'] = pvd\_full['PVD\_Dimension'].str.replace('\_PVD', '', regex=False)

valuesml\_full['PVD\_Dimension'] = valuesml\_full['PVD\_Dimension'].str.replace('\_ValuesML', '', regex=False)

alpha = 0.05

pvd\_full['Sig\_PVD'] = (pvd\_full['P-value'] < alpha).astype(int)

valuesml\_full['Sig\_ValuesML'] = (valuesml\_full['P-value'] < alpha).astype(int)

merged\_sig = pd.merge(

pvd\_full[['PVD\_Dimension', 'Predictor', 'Sig\_PVD']],

valuesml\_full[['PVD\_Dimension', 'Predictor', 'Sig\_ValuesML']],

on=['PVD\_Dimension', 'Predictor']

)

merged\_sig['Sig\_Category'] = merged\_sig['Sig\_PVD'].astype(str) + merged\_sig['Sig\_ValuesML'].astype(str)

category\_order = ['11', '00', '01', '10']

sig\_counts = merged\_sig['Sig\_Category'].value\_counts().reindex(category\_order, fill\_value=0)

plt.figure(figsize=(7, 5))

colors = {

'11': 'green',

'00': 'grey',

'01': 'orange',

'10': 'orange'

}

bar\_colors = [colors[cat] for cat in category\_order]

bars = plt.bar(category\_order, sig\_counts.values, color=bar\_colors)

for bar in bars:

height = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, height + 0.2, str(height),

ha='center', va='bottom', fontsize=10)

legend\_labels = {

'11': 'Both Significant',

'00': 'Both Not Significant',

'01': 'Only ValuesML Significant',

'10': 'Only PVD Significant'

}

handles = [plt.Rectangle((0,0),1,1, color=colors[k]) for k in category\_order]

plt.legend(handles, [legend\_labels[k] for k in category\_order],

title='Significance Category', loc='upper right')

plt.xticks(ticks=range(4), labels=category\_order)

plt.title('Significance Consistency Between PVD and ValuesML')

plt.xlabel('Sig\_PVD / Sig\_ValuesML')

plt.ylabel('Number of Coefficients')

plt.ylim(0, max(sig\_counts.values) + 2)

plt.tight\_layout()

plt.figtext(0.5, -0.05, f'Total compared coefficients: {len(merged\_sig)}',

ha='center', fontsize=10)

plt.show()

#2

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

pvd\_full = pd.read\_csv('PVD\_regression.csv')

valuesml\_full = pd.read\_csv('ValuesML\_regression.csv')

pvd\_r2\_summary = pvd\_full[['PVD\_Dimension', 'R-squared']].drop\_duplicates()

valuesml\_r2\_summary = valuesml\_full[['PVD\_Dimension', 'R-squared']].drop\_duplicates()

pvd\_r2\_summary['PVD\_Dimension'] = pvd\_r2\_summary['PVD\_Dimension'].str.replace('\_PVD', '', regex=False)

valuesml\_r2\_summary['PVD\_Dimension'] = valuesml\_r2\_summary['PVD\_Dimension'].str.replace('\_ValuesML', '', regex=False)

r2\_compare = pd.merge(

pvd\_r2\_summary,

valuesml\_r2\_summary,

on='PVD\_Dimension',

suffixes=('\_PVD', '\_ValuesML')

)

plt.figure(figsize=(9, 7))

sns.set(style='whitegrid')

ax = sns.scatterplot(

data=r2\_compare,

x='R-squared\_PVD',

y='R-squared\_ValuesML',

s=80,

color='steelblue'

)

max\_val = r2\_compare[['R-squared\_PVD', 'R-squared\_ValuesML']].max().max() \* 1.05

plt.plot([0, max\_val], [0, max\_val], color='red', linestyle='--', label='y = x')

for i in range(r2\_compare.shape[0]):

dim = r2\_compare.loc[i, 'PVD\_Dimension']

x = r2\_compare.loc[i, 'R-squared\_PVD']

y = r2\_compare.loc[i, 'R-squared\_ValuesML']

ax.text(x + 0.001, y + 0.001, dim, fontsize=9, color='black')

mean\_pvd = r2\_compare['R-squared\_PVD'].mean()

mean\_valuesml = r2\_compare['R-squared\_ValuesML'].mean()

plt.text(0.01, max\_val \* 0.95, f'Mean R² (PVD): {mean\_pvd:.4f}', fontsize=10, color='gray')

plt.text(0.01, max\_val \* 0.90, f'Mean R² (ValuesML): {mean\_valuesml:.4f}', fontsize=10, color='gray')

plt.title('Comparison of R² between PVD and ValuesML', fontsize=14)

plt.xlabel('Explained Variance (R²) - PVD Method', fontsize=12)

plt.ylabel('Explained Variance (R²) - ValuesML Method', fontsize=12)

plt.legend()

plt.tight\_layout()

plt.show()

#3

import pandas as pd

import numpy as np

pvd = pd.read\_csv('PVD\_regression.csv')

vml = pd.read\_csv('ValuesML\_regression.csv')

ci = pd.read\_csv('CI\_overlap\_analysis.csv')

pvd['PVD\_Dimension'] = pvd['PVD\_Dimension'].str.replace('\_PVD','',regex=False)

vml['PVD\_Dimension'] = vml['PVD\_Dimension'].str.replace('\_ValuesML','',regex=False)

keep = ['PVD\_Dimension','Predictor','Coefficient','P-value']

pvd = pvd[keep].rename(columns={'Coefficient':'Coef','P-value':'P'})

vml = vml[keep].rename(columns={'Coefficient':'Coef','P-value':'P'})

pvd = pvd[pvd['Predictor']!='const'].copy()

vml = vml[vml['Predictor']!='const'].copy()

ci = ci[ci['Predictor']!='const'].copy()

ci\_pvd = ci[['PVD\_Dimension','Predictor','CI\_Lower\_PVD','CI\_Upper\_PVD']].rename(

columns={'CI\_Lower\_PVD':'CI\_L','CI\_Upper\_PVD':'CI\_U'}

)

ci\_vml = ci[['PVD\_Dimension','Predictor','CI\_Lower\_ValuesML','CI\_Upper\_ValuesML']].rename(

columns={'CI\_Lower\_ValuesML':'CI\_L','CI\_Upper\_ValuesML':'CI\_U'}

)

def bh\_fdr(pvals, q=0.05):

"""Benjamini–Hochberg procedure, return boolean array for significance"""

pvals = np.asarray(pvals)

n = len(pvals)

idx = np.argsort(pvals)

thresh = (np.arange(1, n+1) / n) \* q

passed = pvals[idx] <= thresh

if not passed.any():

return np.zeros(n, dtype=bool)

max\_i = np.where(passed)[0].max()

cutoff = thresh[max\_i]

return pvals <= cutoff

def stability\_metrics(df, ci\_df, method\_name):

"""Compute stability metrics (Flip rate, Margin score, CI width) and raw SI"""

out = {}

sig\_10 = (df['P'] < 0.10).astype(int)

sig\_05 = (df['P'] < 0.05).astype(int)

sig\_01 = (df['P'] < 0.01).astype(int)

sig\_fdr = bh\_fdr(df['P'].values, q=0.05).astype(int)

sig\_mat = np.vstack([sig\_10, sig\_05, sig\_01, sig\_fdr]).T

flips\_per\_coef = (sig\_mat.max(axis=1) != sig\_mat.min(axis=1)).astype(int)

flip\_rate = flips\_per\_coef.mean() # lower = better

out['flip\_rate'] = flip\_rate

margins = df['P'].apply(lambda p: min(abs(p-0.10), abs(p-0.05), abs(p-0.01)))

m\_clip = np.minimum(margins, margins.quantile(0.95))

margin\_score = (m\_clip / m\_clip.max()).mean() # higher = better

out['margin\_score'] = margin\_score

ci\_merge = df.merge(ci\_df, on=['PVD\_Dimension','Predictor'], how='left')

ci\_width = (ci\_merge['CI\_U'] - ci\_merge['CI\_L']).abs()

scale = (ci\_width.quantile(0.75) - ci\_width.quantile(0.25))

scale = scale if scale > 0 else ci\_width.std(ddof=1) if ci\_width.std(ddof=1) > 0 else 1.0

ci\_width\_norm = (ci\_width / scale).mean()

out['ci\_width\_norm'] = ci\_width\_norm

out['raw\_for\_norm'] = ci\_width\_norm

return out

pvd\_metrics = stability\_metrics(pvd, ci\_pvd, 'PVD')

vml\_metrics = stability\_metrics(vml, ci\_vml, 'ValuesML')

ci\_vals = np.array([pvd\_metrics['raw\_for\_norm'], vml\_metrics['raw\_for\_norm']])

ci\_min, ci\_max = ci\_vals.min(), ci\_vals.max()

def inv\_minmax(x, xmin, xmax):

if xmax == xmin:

return 1.0

return 1.0 - (x - xmin) / (xmax - xmin) # smaller width = higher score

pvd\_ci\_score = inv\_minmax(pvd\_metrics['raw\_for\_norm'], ci\_min, ci\_max)

vml\_ci\_score = inv\_minmax(vml\_metrics['raw\_for\_norm'], ci\_min, ci\_max)

def assemble\_SI(m, ci\_score):

flip\_score = 1.0 - m['flip\_rate'] # higher = better

margin\_score = m['margin\_score']

ci\_compact = ci\_score

SI = np.mean([flip\_score, margin\_score, ci\_compact])

return flip\_score, margin\_score, ci\_compact, SI

pvd\_flip, pvd\_margin, pvd\_ci, pvd\_SI = assemble\_SI(pvd\_metrics, pvd\_ci\_score)

vml\_flip, vml\_margin, vml\_ci, vml\_SI = assemble\_SI(vml\_metrics, vml\_ci\_score)

result = pd.DataFrame([

['PVD', round(1-pvd\_metrics['flip\_rate'],3), round(pvd\_metrics['margin\_score'],3), round(pvd\_ci,3), round(pvd\_SI,3)],

['ValuesML', round(1-vml\_metrics['flip\_rate'],3), round(vml\_metrics['margin\_score'],3), round(vml\_ci,3), round(vml\_SI,3)],

], columns=['Method','FlipScore(↑)','MarginScore(↑)','CICompact(↑)','StabilityIndex\_SI(↑)'])

print("\n=== Stability Metrics Comparison (higher = more stable) ===")

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

winner = result.loc[result['StabilityIndex\_SI(↑)'].idxmax(),'Method']

print(f"\n>>> More stable method: {winner}")