Molecular EDA & Fingerprint Analysis (High-Quality, Seaborn)

- 주요 기능:
  - 기본 통계/결측치 점검
  - 핵심 변수(`MolWt` , `clogp` , `sa_score` , `qed`) 분포(KDE) & 라벨별 비교 (Box/Violin)
  - **상관관계 히트맵(코어 + `label`)**
  - **ECFP/PTFP/FCFP PCA(2D)**

SMILE CODE: 각 분자의 구조를 표현한 고유한 코드

MolWt: 분자량

QED: 물리화학적 특성, ADME

SA_Score: 합성 난이도

clogp: 지용성(lipophilicity) 을 정량적으로 나타내는 대표 지표

ECFP (Extended Connectivity) 원자 주변의 구조 패턴을 수학적으로 압축해서 표현

FCFP (Functional Class) 원자 자체가 아니라 원자의 "기능적 분류(Functional class)" 를 기반

PTFP (Pattern-based) 사전에 정의된 구조적 패턴(특정 작용기나 결합 조합) 을 분자에서 탐색하여, 그 존재 여부를 벡터로 표시

0) Setup & Load

In [ ]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Aesthetics
sns.set_theme(context="notebook", style="whitegrid")

# Paths
DATA_PATH = "train.csv"  # 필요시 수정
FIG_DIR = "./figs"
os.makedirs(FIG_DIR, exist_ok=True)

# Load
df = pd.read_csv(DATA_PATH)
```

```
core_features = ['MolWt', 'clogp', 'sa_score', 'qed', 'label']
ecfp_cols = [c for c in df.columns if c.startswith('ecfp_')]
ptfp_cols = [c for c in df.columns if c.startswith('ptfp_')]
fcfp_cols = [c for c in df.columns if c.startswith('fcfp_')]

print(f"rows={len(df)}, cols={len(df.columns)} | ECFP={len(ecfp_cols)}, PTFP={le
```

```
rows=8349, cols=3078 | ECFP=1024, PTFP=1024, FCFP=1024
```

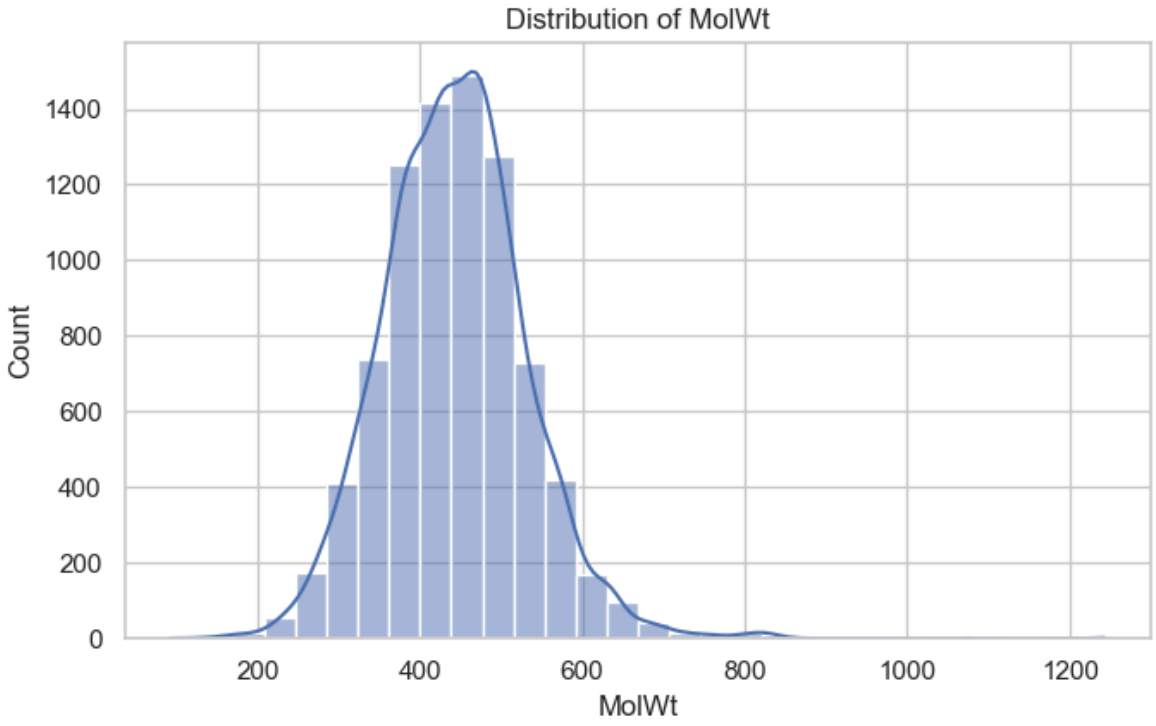
1) Quick audit: info / describe / missing

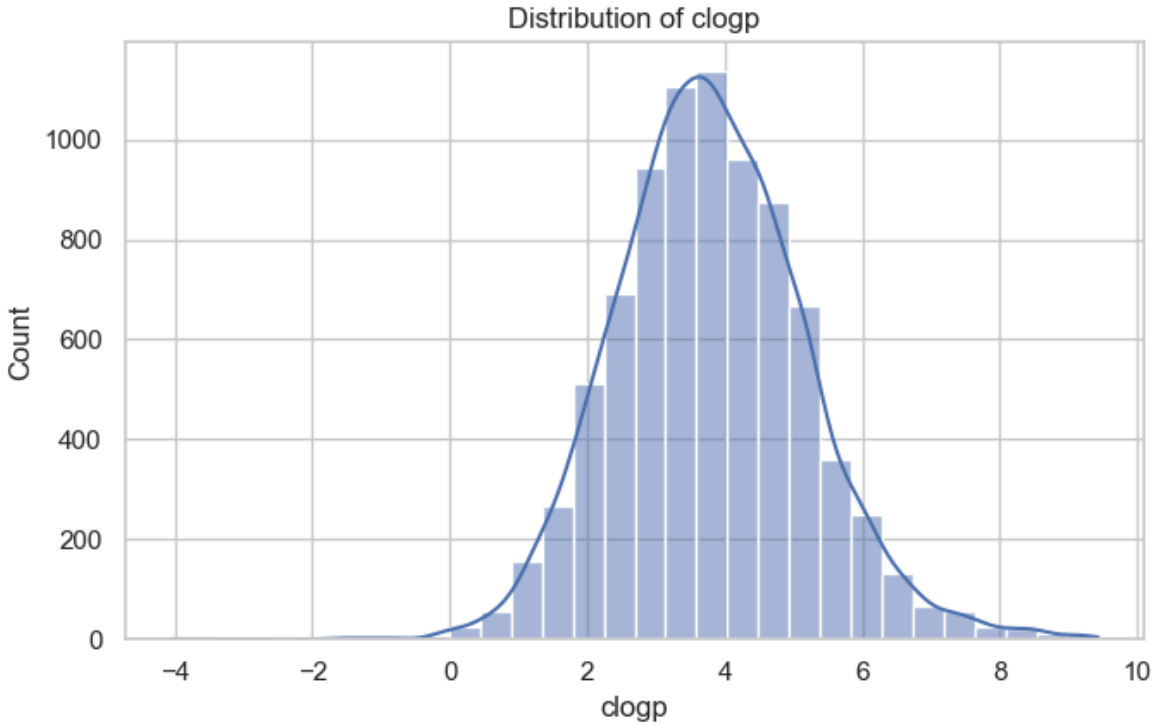
In [ ]:
```
display(df[core_features].describe())
```

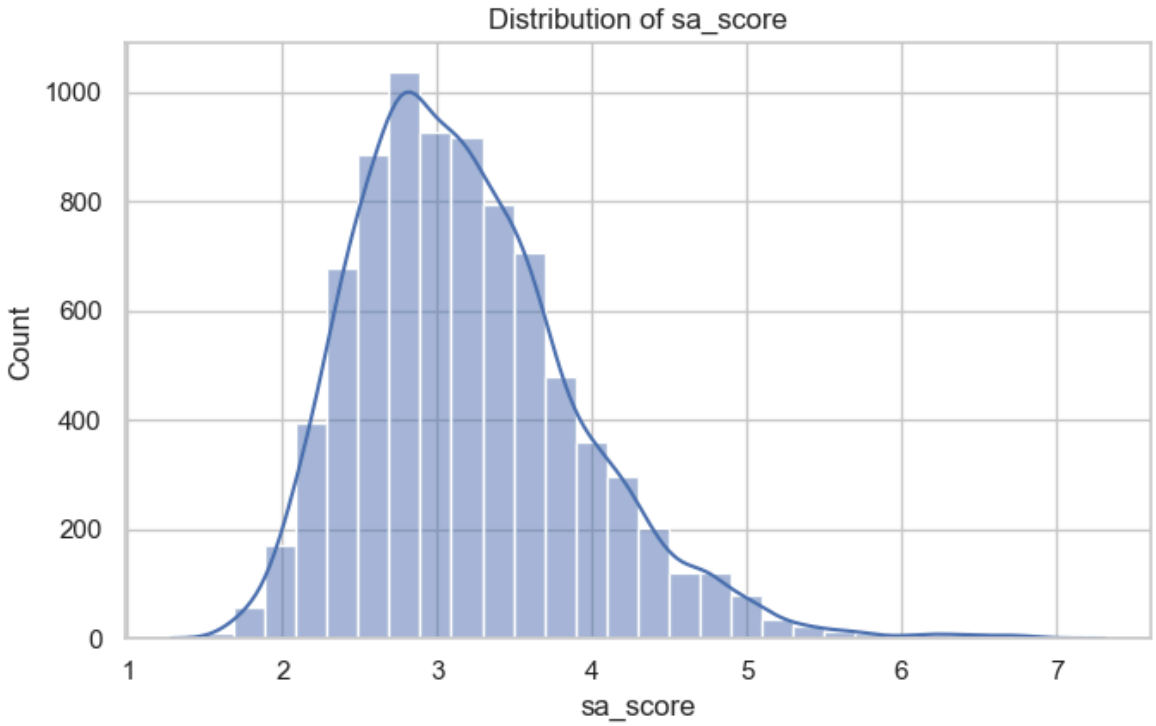
	MolWt	clogp	sa_score	qed	label
count	8349.000000	8349.000000	8349.000000	8349.000000	8349.000000
mean	443.248753	3.794829	3.187613	0.559151	0.544017
std	88.876374	1.379045	0.727768	0.185664	0.498089
min	94.117000	-4.048930	1.282432	0.024365	0.000000
25%	383.879000	2.874500	2.663425	0.425440	0.000000
50%	441.522000	3.735100	3.093155	0.556410	1.000000
75%	495.639000	4.652500	3.593547	0.700506	1.000000
max	1242.488000	9.429480	7.309297	0.947494	1.000000

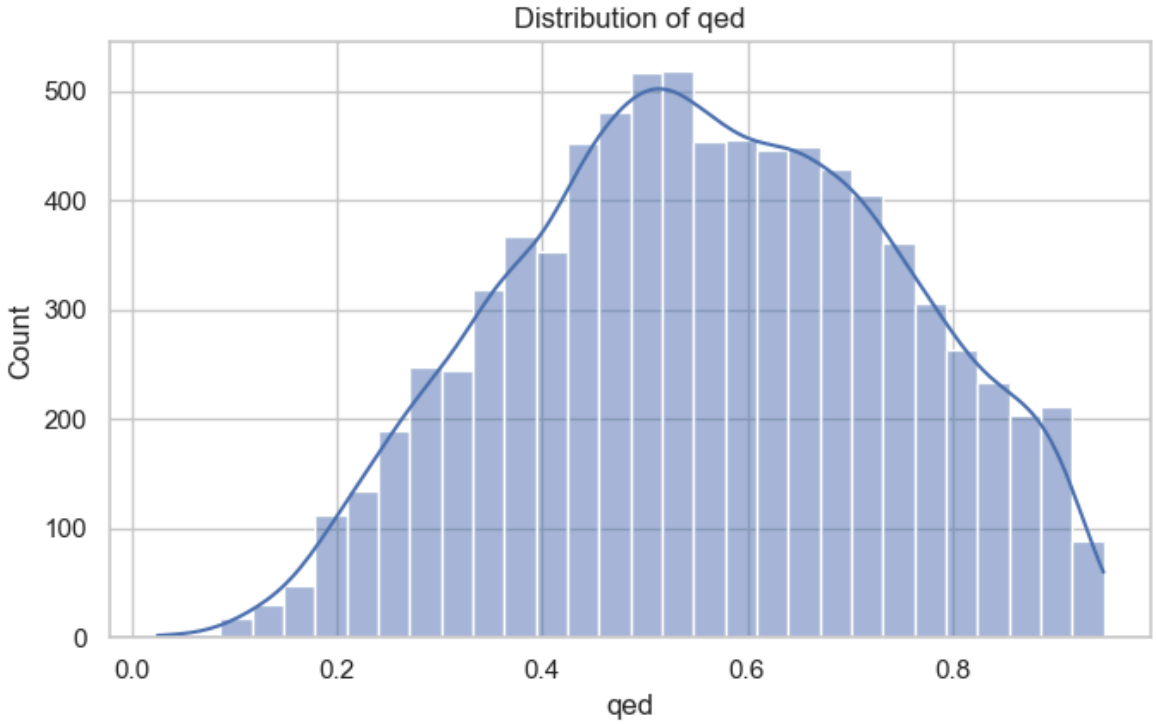
2) Core feature distributions (KDE + Histogram)

In [ ]:
```
for col in ['MolWt', 'clogp', 'sa_score', 'qed']:
    fig, ax = plt.subplots(figsize=(7, 4.5))
    sns.histplot(df[col], bins=30, kde=True, ax=ax)
    ax.set_title(f"Distribution of {col}")
    ax.set_xlabel(col); ax.set_ylabel("Count")
    fig.tight_layout()
    fig.savefig(f"{FIG_DIR}/dist_{col}.png", dpi=200)
    plt.show()
```


Distribution of MolWt
Count
MolWt
0
200
400
600
800
1000
1200
1400
200
400
600
800
1000
1200

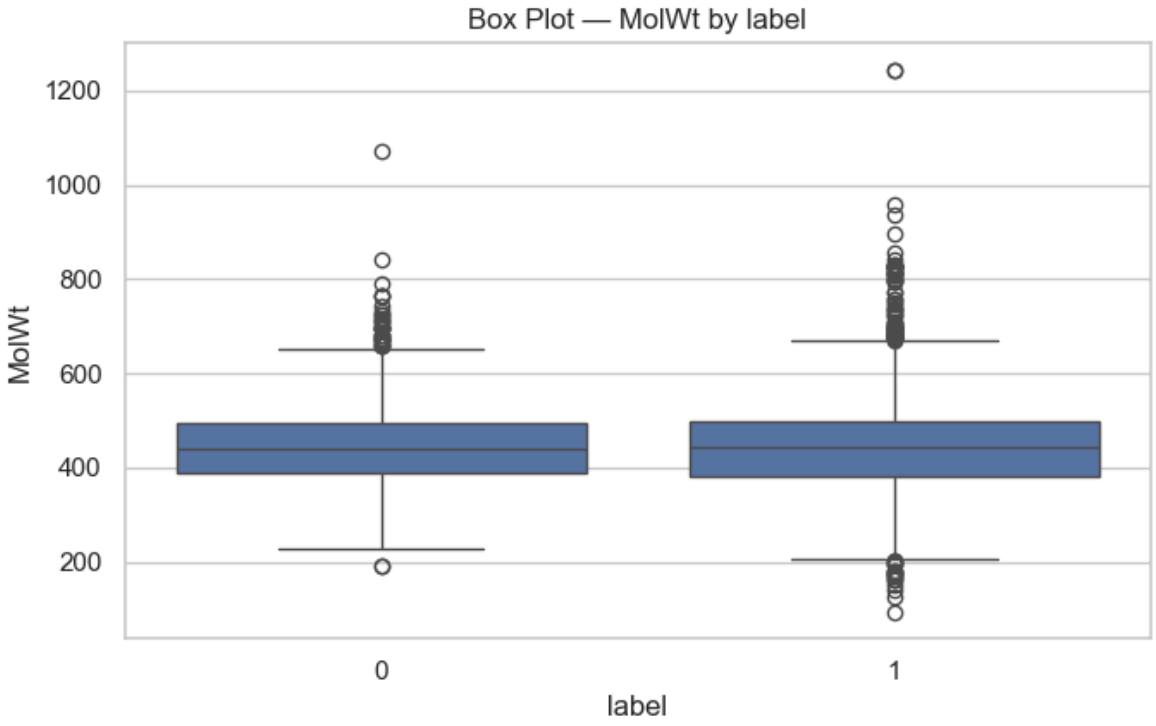

Distribution of clogp
Count
clogp
0
200
400
600
800
1000
−4
−2
0
2
4
6
8
10

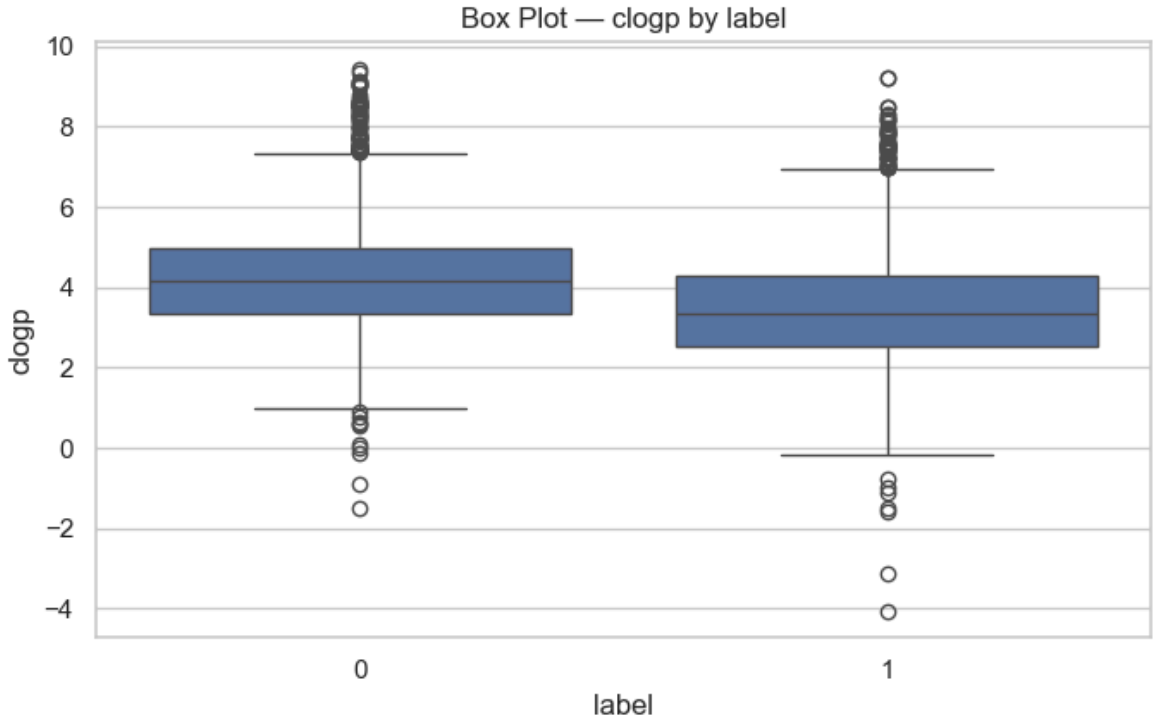


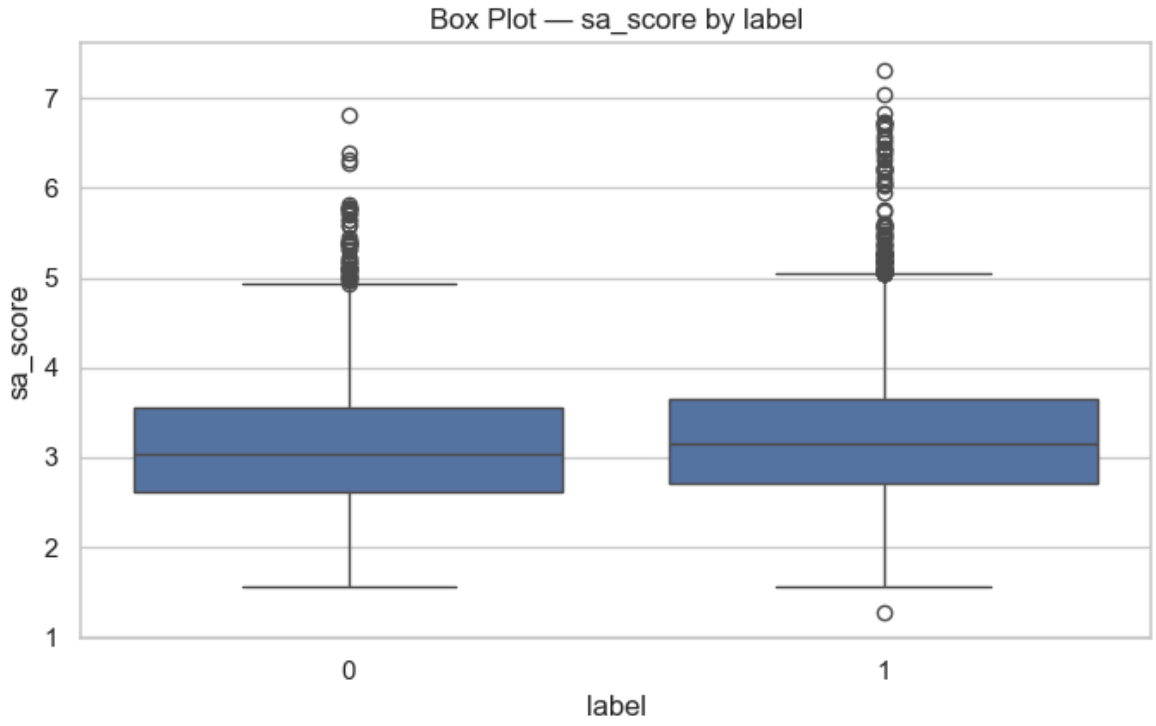


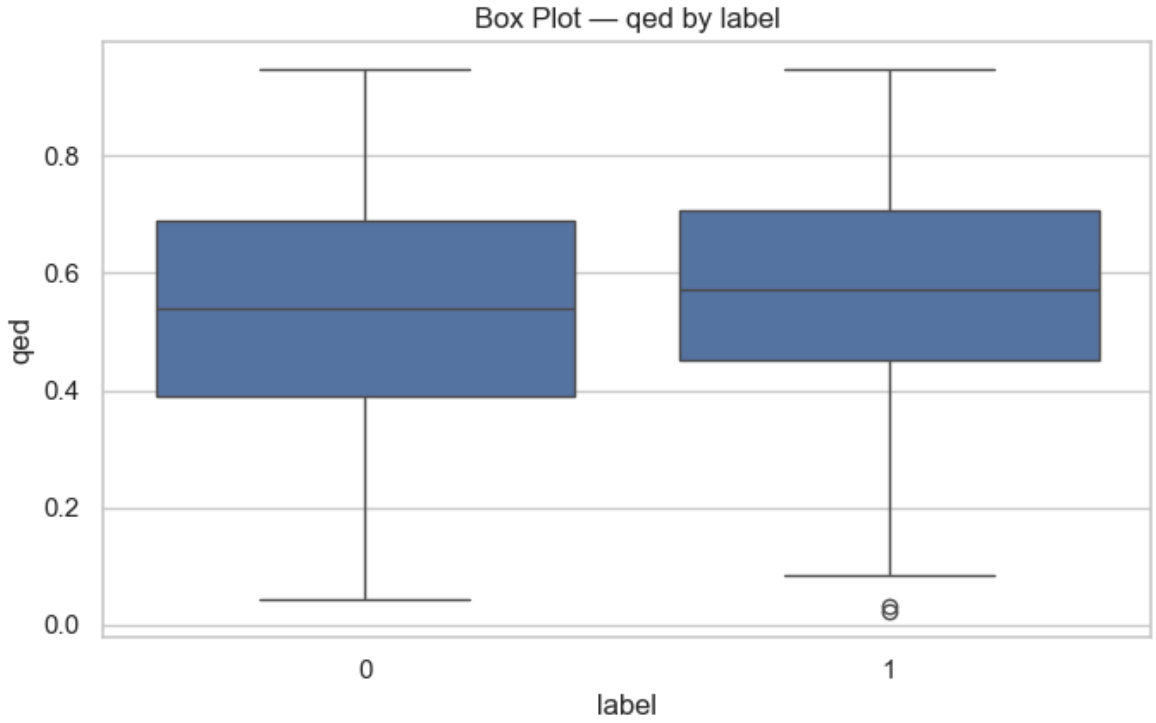
3) Label-wise comparison (Box & Violin)

In [17]:
```
for col in ['MolWt', 'clogp', 'sa_score', 'qed']:

    fig, ax = plt.subplots(figsize=(7, 4.5))
    sns.boxplot(data=df, x='label', y=col, showfliers=True, ax=ax)
    ax.set_title(f"Box Plot — {col} by label")
    fig.tight_layout()
    fig.savefig(f"{FIG_DIR}/box_{col}_by_label.png", dpi=200)
    plt.show()
```


Box Plot — MolWt by label
MolWt
1200
1000
800
600
400
200
0
1
label

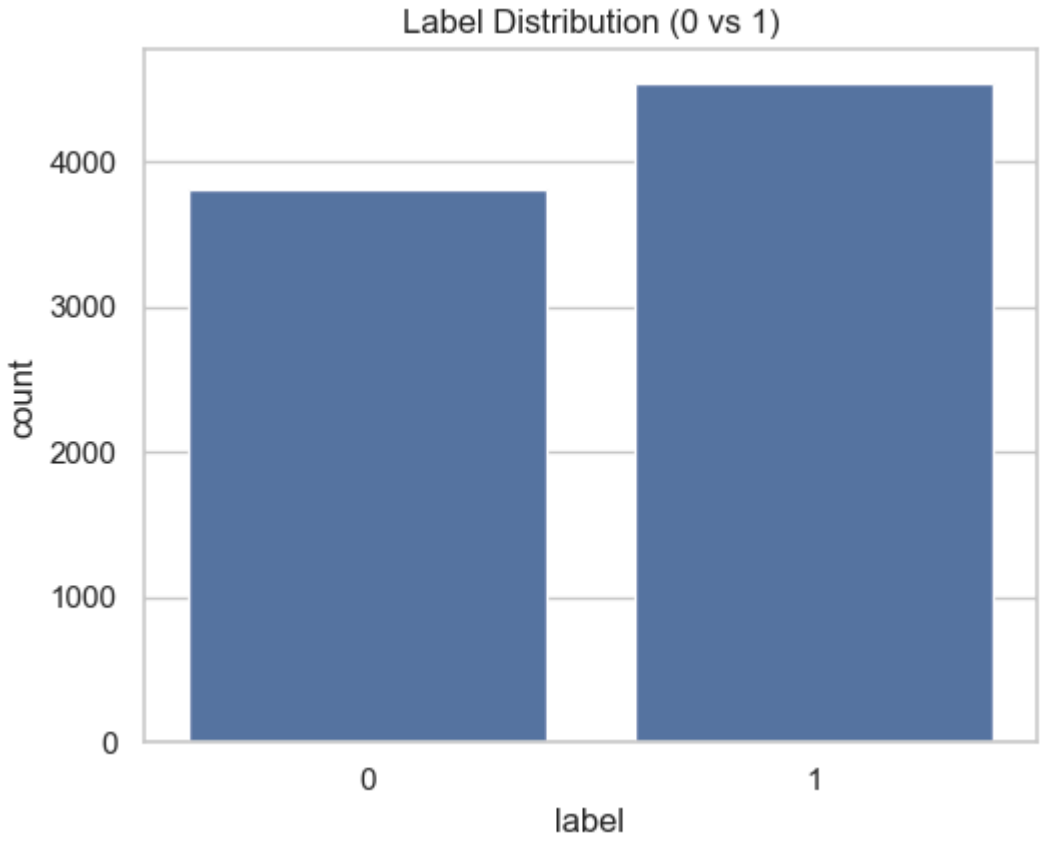

Box Plot — clogp by label
clogp
10
8
6
4
2
0
−2
−4
0
1
label





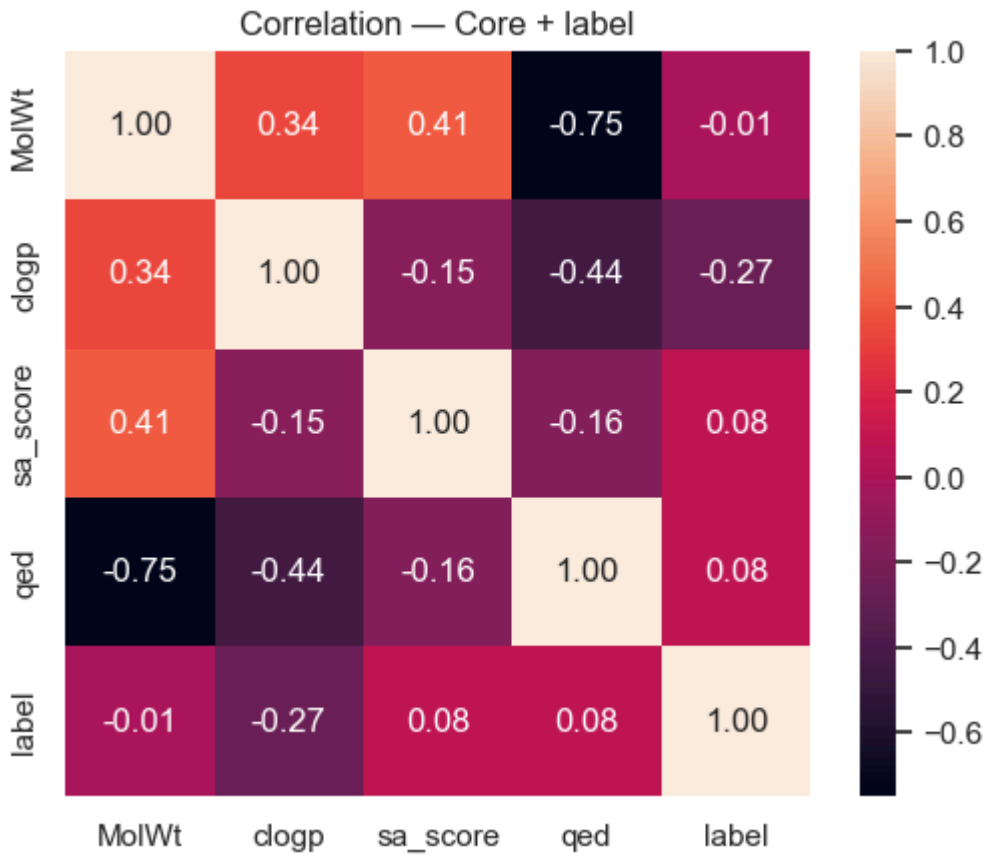
4) Label distribution

In [7]:
```
fig, ax = plt.subplots(figsize=(5.5, 4.5))
sns.countplot(data=df, x='label', ax=ax)
ax.set_title("Label Distribution (0 vs 1)")
fig.tight_layout()
fig.savefig(f"{FIG_DIR}/label_distribution.png", dpi=200)
plt.show()
```



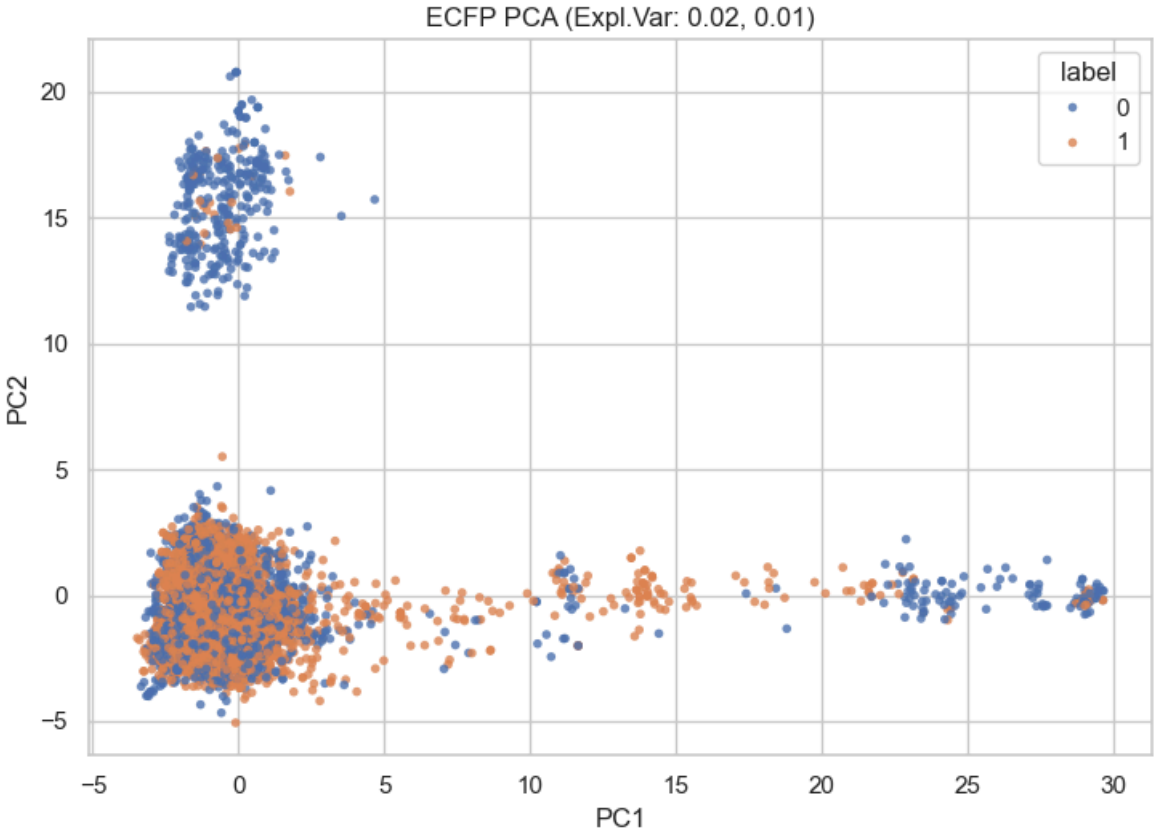
5) Correlations (include `label`)

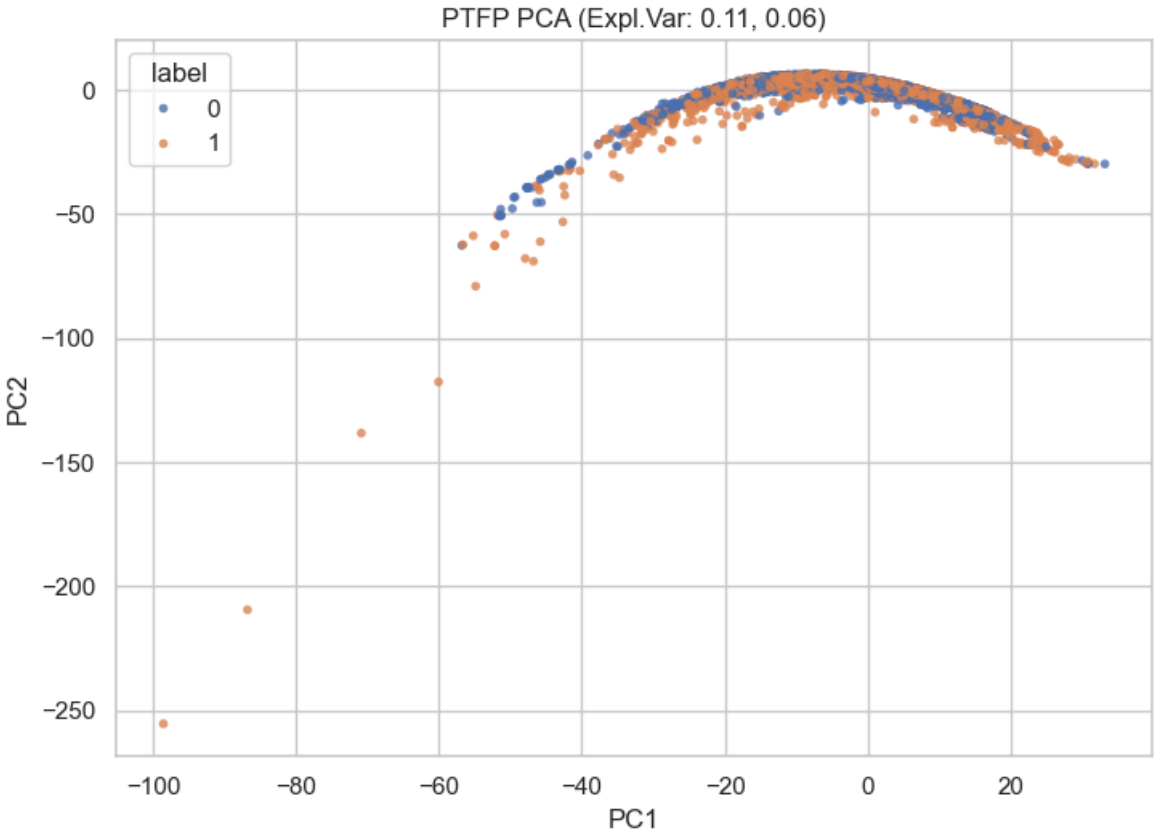
In [10]:
```
vars_core_label = ['MolWt','clogp','sa_score','qed','label']
corr = df[vars_core_label].corr(numeric_only=True)

fig, ax = plt.subplots(figsize=(5.5, 4.5))
sns.heatmap(corr, annot=True, fmt=".2f", square=True, ax=ax)
ax.set_title("Correlation — Core + label")
fig.tight_layout()
fig.savefig(f"{FIG_DIR}/corr_core_label.png", dpi=200)
plt.show()
```

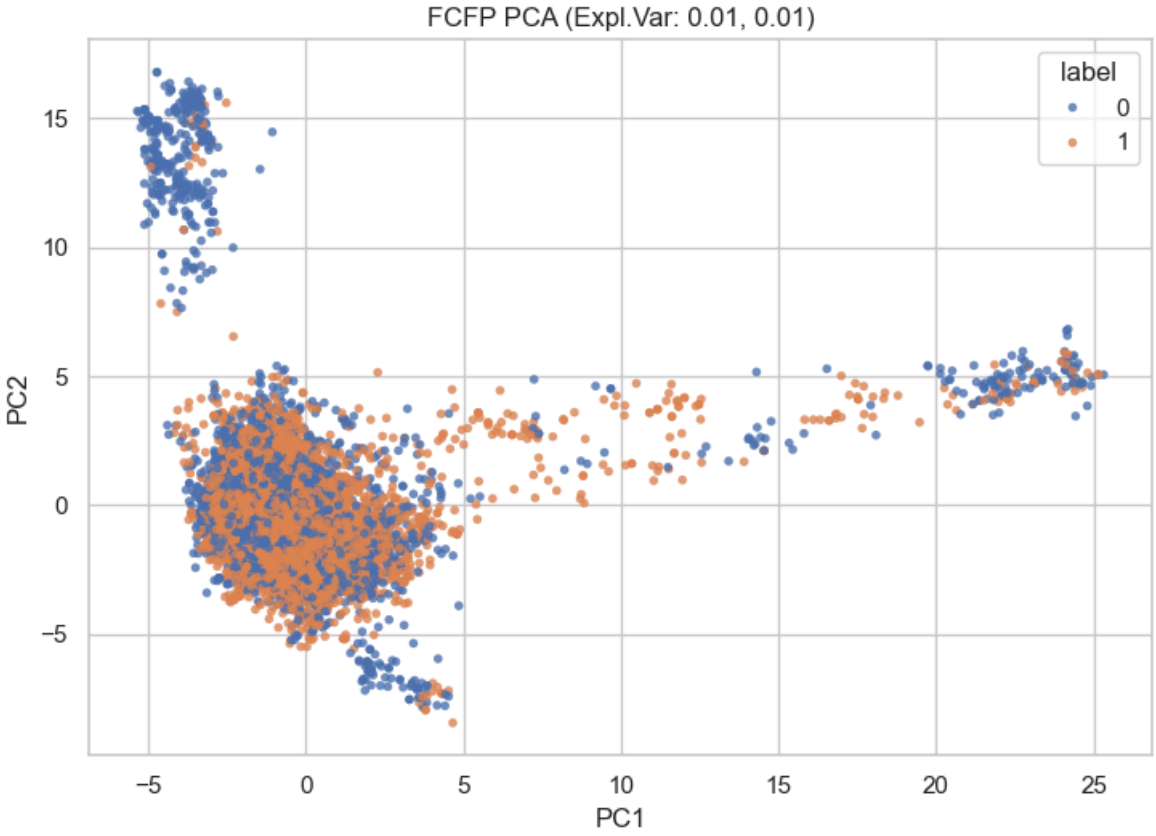


6) PCA — ECFP / PTFP / FCFP (2D)

In [ ]:
```
def pca_plot(X, labels, title, fname):
    scaler = StandardScaler(with_mean=False)
    Xs = scaler.fit_transform(X)
    pca = PCA(n_components=2, random_state=42)
    Xp = pca.fit_transform(Xs)

    fig, ax = plt.subplots(figsize=(7.5, 5.5))
    sns.scatterplot(x=Xp[:,0], y=Xp[:,1], hue=labels, alpha=0.8, s=16, linewidth
    var = pca.explained_variance_ratio_
    ax.set_title(f"{title} (Expl.Var: {var[0]:.2f}, {var[1]:.2f})")
    ax.set_xlabel("PC1"); ax.set_ylabel("PC2")
    ax.legend(title="label", loc="best", frameon=True)
    fig.tight_layout()
    fig.savefig(f"{FIG_DIR}/{fname}.png", dpi=220)
    plt.show()

labels = df['label'].values.astype(int)

if len(ecfp_cols)>0:
    pca_plot(df[ecfp_cols].values, labels, "ECFP PCA", "pca_ecfp")

if len(ptfp_cols)>0:
    pca_plot(df[ptfp_cols].values, labels, "PTFP PCA", "pca_ptfp")

if len(fcfp_cols)>0:
    pca_plot(df[fcfp_cols].values, labels, "FCFP PCA", "pca_fcfp")
```

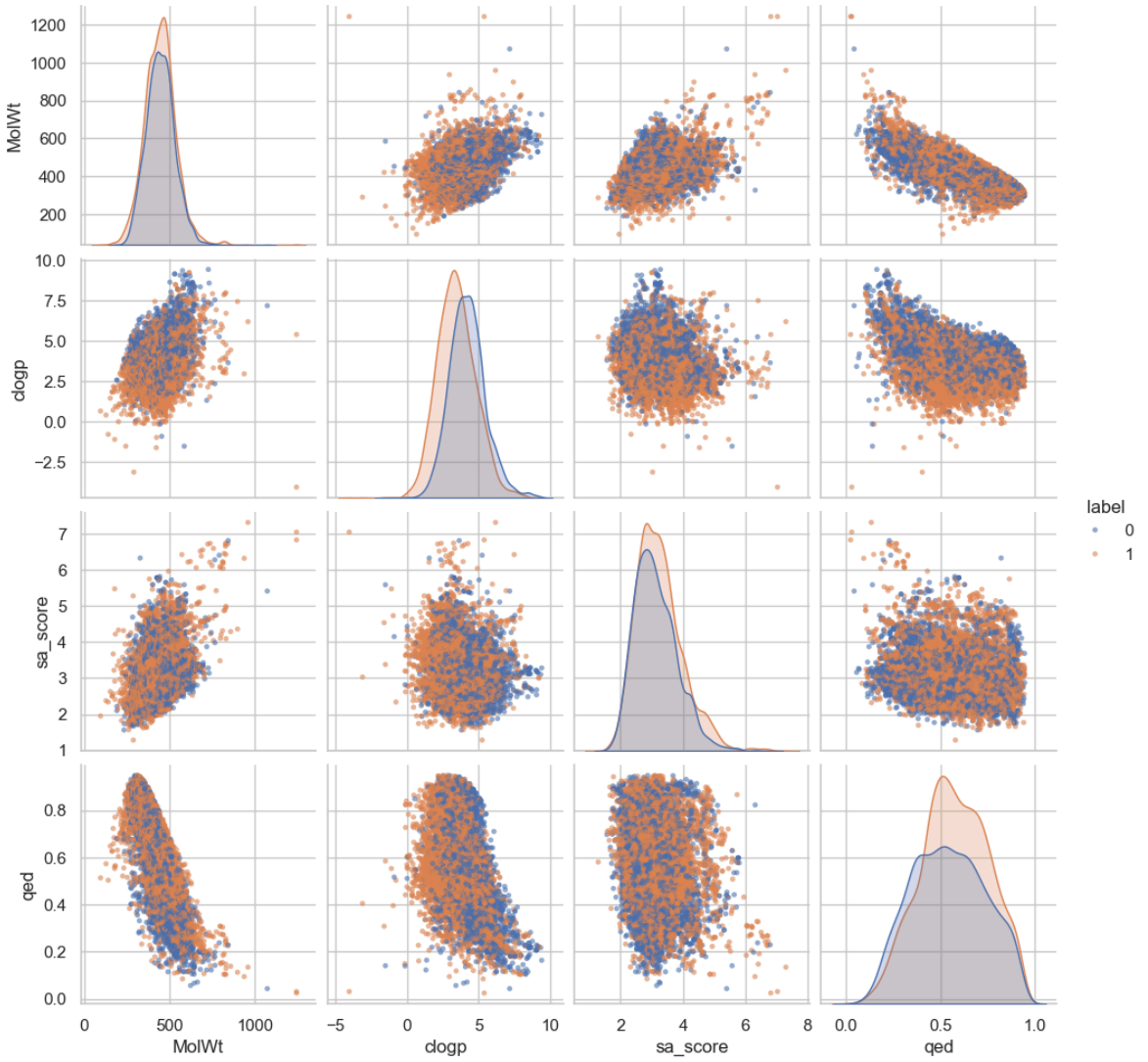

ECFP PCA (Expl.Var: 0.02, 0.01)
label
0
1
PC2
PC1
20
15
10
5
0
−5
−5
0
5
10
15
20
25
30


PTFP PCA (Expl.Var: 0.11, 0.06)
label
0
1
PC2
PC1
0
−50
−100
−150
−200
−250
−100
−80
−60
−40
−20
0
20



7) (Optional) Pairplot — Core variables by label

In [ ]:
```
sns.pairplot(df[['MolWt','clogp','sa_score','qed','label']], hue='label', diag_k
             plot_kws={'alpha':0.6, 's':12, 'linewidth':0})
plt.savefig(f"{FIG_DIR}/pairplot_core.png", dpi=180)
plt.show()
```



In [ ]:
```
%pip install rdkit
```

```
Collecting rdkit
  Downloading rdkit-2025.9.1-cp311-cp311-win_amd64.whl.metadata (4.2 kB)
Requirement already satisfied: numpy in c:\users\administrator\appdata\local\prog
rams\python\python311\lib\site-packages (from rdkit) (1.26.4)
Requirement already satisfied: Pillow in c:\users\administrator\appdata\local\pro
grams\python\python311\lib\site-packages (from rdkit) (11.2.1)
Downloading rdkit-2025.9.1-cp311-cp311-win_amd64.whl (23.6 MB)
   ---------------------------------------- 0.0/23.6 MB ? eta -:--:--
   ---------------------------- ----------- 17.0/23.6 MB 76.6 MB/s eta 0:00:01
   ---------------------------------------  23.3/23.6 MB 64.3 MB/s eta 0:00:01
   ---------------------------------------- 23.6/23.6 MB 51.4 MB/s  0:00:00
Installing collected packages: rdkit
Successfully installed rdkit-2025.9.1
Note: you may need to restart the kernel to use updated packages.
```

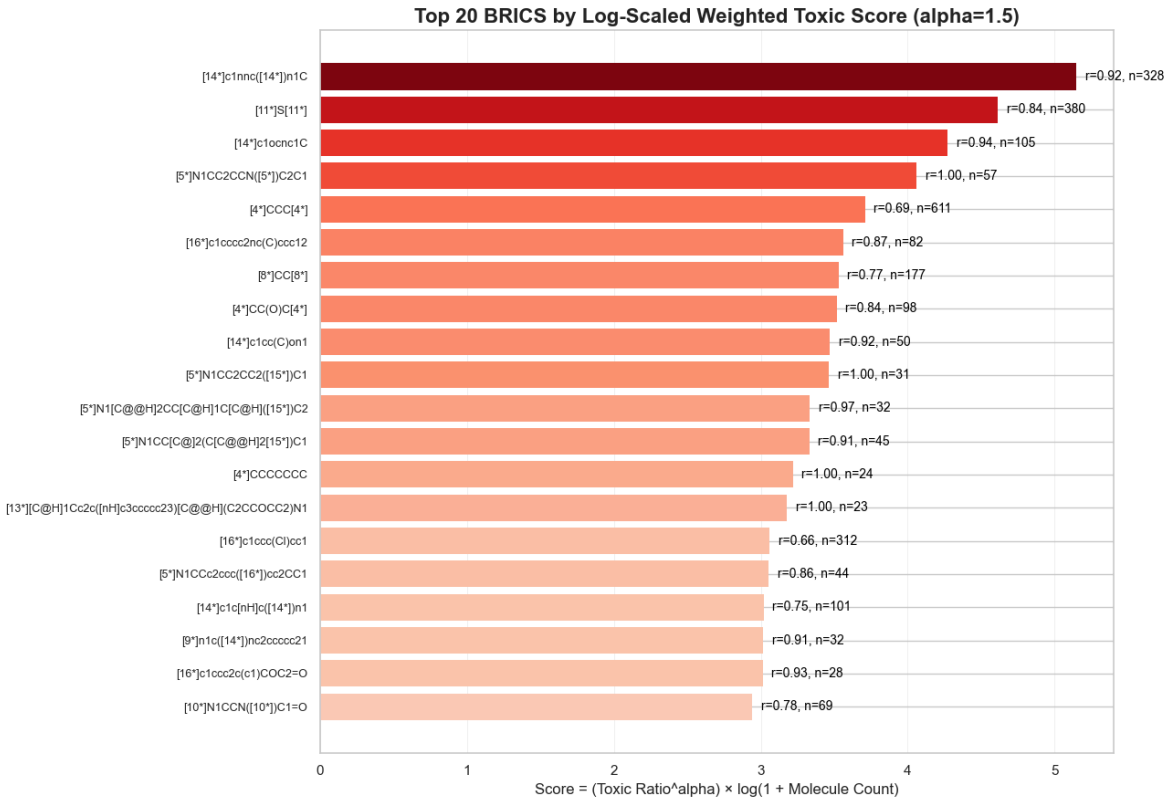
8) TOXIC RATIO PER BRICS

BRICS(Breaking of Retrosynthetically Interesting Chemical Substructures)는 화합물을 단순히 랜덤하게 자르지 않고, 유기화학적 결합 규칙에 따라 잘라주는 알고리즘

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from collections import defaultdict, Counter
from rdkit import Chem
from rdkit.Chem import BRICS

CSV_PATH = "train.csv"
OUT_PNG  = "bar_top20_wts_log_alpha.png"
ALPHA    = 1.5

# ---------- BRICS utils ----------
def extract_precise_brics(smiles: str):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return set()
    try:
        frag_mol = BRICS.BreakBRICSBonds(mol)
        frags = Chem.GetMolFrags(frag_mol, asMols=True, sanitizeFrags=True)
        return set(Chem.MolToSmiles(f, canonical=True) for f in frags)
    except Exception:
        return set()

def build_fragment_table(df: pd.DataFrame) -> pd.DataFrame:
    # toxic is label==0
    frag_to_tox = Counter()
    frag_to_non = Counter()
    frag_to_molset = defaultdict(set)

    for idx, (smi, lab) in enumerate(zip(df["SMILES"], df["label"])):
        frags = extract_precise_brics(smi)
        if not frags:
            continue
        for frag in frags:
            frag_to_molset[frag].add(idx)
        if int(lab) == 0:
            for frag in frags: frag_to_tox[frag] += 1
        else:
            for frag in frags: frag_to_non[frag] += 1

    rows = []
    for frag, mol_ids in frag_to_molset.items():
        n = len(mol_ids)
        tox = frag_to_tox[frag]
        non = frag_to_non[frag]
        total = tox + non
        r = tox / total if total > 0 else 0.0
        rows.append((frag, r, n))
    return pd.DataFrame(rows, columns=["fragment", "toxic_ratio", "count_molecul

# ---------- Plot (matplotlib only; red gradient, labels on bars) ----------
def plot_top20_like_right(frag_df: pd.DataFrame, alpha: float, out_png: str):
    df = frag_df.copy()
    df["score"] = (df["toxic_ratio"] ** alpha) * np.log1p(df["count_molecules"])
    top = df.sort_values("score", ascending=False).head(20).reset_index(drop=Tru

    # color map by score (Reds gradient)
    scores = top["score"].to_numpy()
    smin, smax = float(scores.min()), float(scores.max())
    norm = (scores - smin) / (smax - smin + 1e-12)
    colors = plt.cm.Reds(0.2 + 0.75 * norm)

    fig, ax = plt.subplots(figsize=(13, 9))
```

```
    y = np.arange(len(top))
    bars = ax.barh(y, scores, color=colors, edgecolor="none", linewidth=0)

    # y tick labels (fragments)
    ax.set_yticks(y)
    ax.set_yticklabels(top["fragment"].tolist(), fontsize=9)

    # text labels: r=…, n=…
    for i, (b, r, n) in enumerate(zip(bars, top["toxic_ratio"], top["count_molec
        x = b.get_width()
        ax.text(x + 0.06, b.get_y() + b.get_height()/2,
                f"r={r:.2f}, n={int(n)}", va="center", fontsize=10, color="black

    ax.invert_yaxis()
    ax.set_xlabel("Score = (Toxic Ratio^alpha) × log(1 + Molecule Count)", fonts
    ax.set_title(f"Top 20 BRICS by Log-Scaled Weighted Toxic Score (alpha={alpha
                 fontsize=16, weight="bold")

    # light grid & tidy margins (like the right figure)
    ax.xaxis.grid(True, linewidth=0.5, alpha=0.35)
    ax.set_axisbelow(True)
    fig.tight_layout()
    fig.savefig(out_png, dpi=200)


df = pd.read_csv(CSV_PATH)          # needs columns: SMILES, label
frag_df = build_fragment_table(df)  # fragment, toxic_ratio, count_molecules
plot_top20_like_right(frag_df, ALPHA, OUT_PNG)
```



Extra) 인사이트 정리

- clogp값이 가장 높은(지용성) label과의 연관성을 띄었다.

- Fingerprint PCA 분석 결과, 단일 특성보다 물리화학적 요인과 구조적 패턴을 결합된 유의미한 데이터를 얻을 수 있었다.
- SMILES CODE에서 BRICS를 추출해 사용하여 TOXIC 판별에 사용할 수 있다.
- SMILES CODE가 극단적으로 짧거나 긴 값은 유의할 필요가 있다.