Project Objective

Develop machine learning models to predict molecular toxicity across 12 biological assays using SMILES representations, enabling in-ico screening for drug discovery and chemical safety assessment.

Dataset Overview

- **Source:** Tox21 Challenge Dataset
- **Size:** 7,831 molecules
- **Features:** SMILES strings
- **Targets:** 12 binary toxicity assays (NR and SR pathways)
- **Challenge:** High class imbalance (2-16% positive class) and missing labels (7-26%)

The workflow includes:

- Loading and inspecting the Tox21 dataset
- Handling missing assay values and class imbalance
- Generating molecular descriptors and fingerprints from SMILES using RDKit
- Exploring descriptor distributions before and after normalization
- Training multiple machine learning models for each toxicity assay
- Selecting the best model per assay based on cross-validated F1 score

The final outcome is a set of optimized binary classification models that can predict toxic or non-toxic behavior of molecules for individual biological targets.

In [102…

```python
# -- General --
import joblib
import os

# -- EDA and Preprocessing --
import pandas as pd
import numpy as np
```

```
# -- Visualization --
import seaborn as sns
import matplotlib.pyplot as plt

# -- RDKit --
from rdkit import Chem
from rdkit.Chem import Draw, rdMolDescriptors, Lipinski, Descriptors, Crippen
from rdkit import DataStructs
from rdkit.Chem.rdFingerprintGenerator import GetMorganGenerator

# -- Machine Learning --
from sklearn.model_selection import train_test_split
from sklearn.base import clone
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline

# -- Preprocessing --
from sklearn.preprocessing import PowerTransformer, StandardScaler

# -- Metrics --
from sklearn.metrics import f1_score, confusion_matrix, classification_report

import warnings

warnings.filterwarnings(
    "ignore",
    message="X does not have valid feature names",
    category=UserWarning
)
```

1. Loading Data

In [103…

```python
# Loading Data
df_tox = pd.read_csv('../data/tox21.csv')
```

In [104…

```python
print(f'The rows in the dataset are {df_tox.shape[0]} and the columns are {df_tox.shape[1]}!\n')
print(f'The columns in the dataset are:')

for no, col in enumerate(df_tox.columns, start=1):
    print(f'{no}. {col}')
```

```
The rows in the dataset are 7831 and the columns are 14!

The columns in the dataset are:
1. NR-AR
2. NR-AR-LBD
3. NR-AhR
4. NR-Aromatase
5. NR-ER
6. NR-ER-LBD
7. NR-PPAR-gamma
8. SR-ARE
9. SR-ATAD5
10. SR-HSE
11. SR-MMP
12. SR-p53
13. mol_id
14. smiles
```

Tox21 Dataset Column Definitions

Nuclear Receptor (NR) Pathways

- `NR-AR` : Tests if a chemical affects the Androgen (male hormone) receptor.
- `NR-AR-LBD` : Tests if a chemical binds to the active site of the Androgen receptor.
- `NR-AhR` : Tests if a chemical acts like toxic environmental pollutants.
- `NR-Aromatase` : Tests if a chemical blocks estrogen formation.
- `NR-ER` : Tests if a chemical affects the Estrogen (female hormone) receptor.
- `NR-ER-LBD` : Tests if a chemical binds to the active site of the Estrogen receptor.

- `NR-PPAR-gamma` : Tests if a chemical affects fat and metabolism control.

Stress Response (SR) Pathways

- `SR-ARE` : Tests if a chemical causes oxidative (cell) stress.
- `SR-ATAD5` : Tests if a chemical damages DNA during replication.
- `SR-HSE` : Tests if a chemical causes protein stress or misfolding.
- `SR-MMP` : Tests if a chemical damages mitochondria (cell energy).
- `SR-p53` : Tests if a chemical triggers DNA damage response or cell death.

🔎 Key Observations

- The dataset contains 7,831 molecules, each identified by a unique `mol_id` and a corresponding `smiles` string.
- There are 12 toxicity assay columns, covering Nuclear Receptor (NR) and Stress Response (SR) pathways.
- The dataset structure is suitable for multi-target binary classification, where each assay is treated as an independent prediction task.

2. Data Inspection

In [105…
```
df_tox.shape
```

Out[105…
```
(7831, 14)
```

In [106…
```
df_tox.head()
```

Out[106…

	NR-AR	NR-AR-LBD	NR-AhR	NR-Aromatase	NR-ER	NR-ER-LBD	NR-PPAR-gamma	SR-ARE	SR-ATAD5	SR-HSE	SR-MMP	SR-p53	mol_id	
0	0.0	0.0	1.0	NaN	NaN	0.0	0.0	1.0	0.0	0.0	0.0	0.0	TOX3021	CCOc1ccc2nc(
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0	NaN	0.0	0.0	TOX3020	CCN1C(=O)N
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	0.0	NaN	NaN	TOX3024	CC[C@]1(O)CC[C@H]2[C@@H]3CCC4=CCCC
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0	NaN	0.0	0.0	TOX3027	CCCN(CC)C(CC)C(
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	TOX20800	CC(O)(P(=

In [107…

```python
# Reordering columns
cols = df_tox.columns.tolist()
cols = ['mol_id', 'smiles'] + [c for c in cols if c != 'smiles' and c != 'mol_id']
df_tox = df_tox[cols]
```

In [108…

```python
df_tox.head()
```

Out[108…

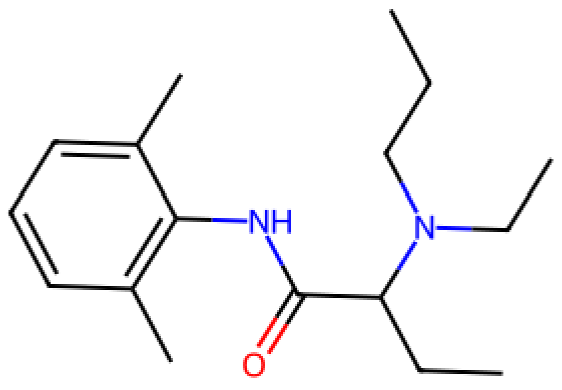
	mol_id	smiles	NR-AR	NR-AR-LBD	NR-AhR	NR-Aromatase	NR-ER	NR-ER-LBD	NR-PPAR-gamma	SR-ARE	SR-ATAD5	S H
0	TOX3021	CCOc1ccc2nc(S(N)(=O)=O)sc2c1	0.0	0.0	1.0	NaN	NaN	0.0	0.0	1.0	0.0	
1	TOX3020	CCN1C(=O)NC(c2ccccc2)C1=O	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0	N
2	TOX3024	CC[C@]1(O)CC[C@H]2[C@@H]3CCC4=CCCC[C@@H]4[C@H]...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	NaN	
3	TOX3027	CCCN(CC)C(CC)C(=O)Nc1c(C)cccc1C	0.0	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0	N
4	TOX20800	CC(O)(P(=O)(O)O)P(=O)(O)O	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

In [109…

```python
# Example  of Visualization of a molecule from DataSet
mol = Chem.MolFromSmiles("CCCN(CC)C(CC)C(=O)Nc1c(C)cccc1C")
```

```
Draw.MolToImage(mol)
```

Out[109…



In [110…

```
df_tox.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7831 entries, 0 to 7830
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   mol_id         7831 non-null   object 
 1   smiles         7831 non-null   object 
 2   NR-AR          7265 non-null   float64
 3   NR-AR-LBD      6758 non-null   float64
 4   NR-AhR         6549 non-null   float64
 5   NR-Aromatase   5821 non-null   float64
 6   NR-ER          6193 non-null   float64
 7   NR-ER-LBD      6955 non-null   float64
 8   NR-PPAR-gamma  6450 non-null   float64
 9   SR-ARE         5832 non-null   float64
 10  SR-ATAD5       7072 non-null   float64
 11  SR-HSE         6467 non-null   float64
 12  SR-MMP         5810 non-null   float64
 13  SR-p53         6774 non-null   float64
dtypes: float64(12), object(2)
memory usage: 856.6+ KB
```

In [111…

```
df_tox.columns
```

Out[111…

```
Index(['mol_id', 'smiles', 'NR-AR', 'NR-AR-LBD', 'NR-AhR', 'NR-Aromatase',
       'NR-ER', 'NR-ER-LBD', 'NR-PPAR-gamma', 'SR-ARE', 'SR-ATAD5', 'SR-HSE',
       'SR-MMP', 'SR-p53'],
      dtype='object')
```

In [112…

```
target_columns = ['NR-AR', 'NR-AR-LBD', 'NR-AhR', 'NR-Aromatase',
                  'NR-ER', 'NR-ER-LBD', 'NR-PPAR-gamma', 'SR-ARE',
                  'SR-ATAD5', 'SR-HSE', 'SR-MMP', 'SR-p53']

summary = pd.DataFrame({
    "null_count": df_tox[target_columns].isna().sum(),
    "non_null_count": df_tox[target_columns].notna().sum(),
    "null_%": (df_tox[target_columns].isna().mean() * 100).round(2),

    # class counts (ignore NaNs automatically)
    "count_0": (df_tox[target_columns] == 0).sum(),
    "count_1": (df_tox[target_columns] == 1).sum(),
    # percentage of positive class
    "%_ones": ((df_tox[target_columns] == 1).sum()
                / df_tox[target_columns].notna().sum()
                * 100).round(2)

}).sort_values("null_count", ascending=False)

summary
```

Out[112…

	null_count	non_null_count	null_%	count_0	count_1	%_ones
SR-MMP	2021	5810	25.81	4892	918	15.80
NR-Aromatase	2010	5821	25.67	5521	300	5.15
SR-ARE	1999	5832	25.53	4890	942	16.15
NR-ER	1638	6193	20.92	5400	793	12.80
NR-PPAR-gamma	1381	6450	17.64	6264	186	2.88
SR-HSE	1364	6467	17.42	6095	372	5.75
NR-AhR	1282	6549	16.37	5781	768	11.73
NR-AR-LBD	1073	6758	13.70	6521	237	3.51
SR-p53	1057	6774	13.50	6351	423	6.24
NR-ER-LBD	876	6955	11.19	6605	350	5.03
SR-ATAD5	759	7072	9.69	6808	264	3.73
NR-AR	566	7265	7.23	6956	309	4.25

In [113…

```
df_tox.columns
```

Out[113…

```
Index(['mol_id', 'smiles', 'NR-AR', 'NR-AR-LBD', 'NR-AhR', 'NR-Aromatase',
       'NR-ER', 'NR-ER-LBD', 'NR-PPAR-gamma', 'SR-ARE', 'SR-ATAD5', 'SR-HSE',
       'SR-MMP', 'SR-p53'],
      dtype='object')
```

In [114…

```
df_tox['smiles'].duplicated().sum()
```

Out[114…

```
np.int64(0)
```

🔎 Key Observations

- Each assay is a binary label where:
  - `0` indicates non-toxic (inactive)
  - `1` indicates toxic (active)
- All assay columns contain missing values, indicating that not every molecule was tested against every biological target.
- The percentage of missing values varies across assays, with some targets having more than 25% missing data.
- The class distribution is highly imbalanced:
  - Most molecules are labeled as non-toxic (0)
  - Toxic (1) samples are comparatively rare
- This imbalance highlights the need for appropriate evaluation metrics (F1 score) and balanced learning strategies during modeling.

3. Feature Engineering

In [115…

```
def calculate_descriptors_safe(smiles):
    """
    Calculates multiple molecular descriptors from a SMILES string.
    Returns a pandas Series so it can add multiple columns at once.
    Output schema is IDENTICAL to the original function.
    """
    morgan_gen = GetMorganGenerator(radius=3, fpSize=2048)
    try:
        # 1. Safely create and sanitize molecule
        mol = Chem.MolFromSmiles(smiles, sanitize=True)
        if mol is None:
            raise ValueError("Invalid SMILES")

        # 2. Remove explicit hydrogens (fixes H-without-neighbors warning)
        mol = Chem.RemoveHs(mol)
        # 3. Calculate descriptors (same as before)
        return pd.Series({
            'tpsa': rdMolDescriptors.CalcTPSA(mol),
            'mw':   Descriptors.MolWt(mol),
            'hbd':  Descriptors.NumHDonors(mol),
            'hba':  Descriptors.NumHAcceptors(mol),
            'rot':  Descriptors.NumRotatableBonds(mol),
            'logp': Crippen.MolLogP(mol),
            'mr':   Crippen.MolMR(mol),
            'hac':  Lipinski.HeavyAtomCount(mol),
            'fp':   morgan_gen.GetFingerprint(mol),  # 🔥 NO deprecation warning
            'ring_count': rdMolDescriptors.CalcNumRings(mol)
        })

    except Exception:
        # 4. Fail safely (same NaN structure as original)
        return pd.Series({
            'tpsa': np.nan,
            'mw': np.nan,
            'hbd': np.nan,
            'hba': np.nan,
            'rot': np.nan,
            'logp': np.nan,
            'mr': np.nan,
            'hac': np.nan,
            'fp': np.nan,
            'ring_count': np.nan
        })

# Apply the function (Creating all columns in one go)
# This returns a new DataFrame with the descriptor columns
descriptor_cols = df_tox['smiles'].apply(calculate_descriptors_safe)

# Join these new columns back to your original DataFrame
df_tox = pd.concat([df_tox, descriptor_cols], axis=1)

# Check for failures (just checking one column is enough since they all fail together)
print(f"Failed rows: {df_tox['tpsa'].isna().sum()}")
```

```
[14:00:42] WARNING: not removing hydrogen atom without neighbors
[14:00:42] WARNING: not removing hydrogen atom without neighbors
[14:00:47] Explicit valence for atom # 8 Al, 6, is greater than permitted
[14:00:49] Explicit valence for atom # 3 Al, 6, is greater than permitted
[14:00:49] Explicit valence for atom # 4 Al, 6, is greater than permitted
[14:00:52] Explicit valence for atom # 4 Al, 6, is greater than permitted
[14:00:53] Explicit valence for atom # 9 Al, 6, is greater than permitted
[14:00:53] Explicit valence for atom # 5 Al, 6, is greater than permitted
[14:00:56] Explicit valence for atom # 16 Al, 6, is greater than permitted
[14:00:58] Explicit valence for atom # 20 Al, 6, is greater than permitted
Failed rows: 8
```

In [116…

```
# Filter for rows where TPSA is NaN (empty)
invalid_rows = df_tox[df_tox['hbd'].isna()].index

# Display just the SMILES strings that failed
print("Invalid SMILES strings:")
print(invalid_rows)

df_tox = df_tox.drop(index=invalid_rows).reset_index(drop=True)
```

```
Invalid SMILES strings:
Index([1322, 2290, 2297, 3558, 4565, 4649, 5538, 6723], dtype='int64')
```

In [117…

```
# Filter for rows where TPSA is NaN (empty)
invalid_rows = df_tox[df_tox['hbd'].isna()].index

# Display just the SMILES strings that failed
print("Invalid SMILES strings:")
print(invalid_rows)
```

```
Invalid SMILES strings:
Index([], dtype='int64')
```

In [118…

```
df_tox.shape
```

Out[118…

```
(7823, 24)
```

In [119…

```
df_tox.head(1)
```

Out[119…

	mol_id	smiles	NR-AR	NR-AR-LBD	NR-AhR	NR-Aromatase	NR-ER	NR-ER-LBD	NR-PPAR-gamma	SR-ARE	...	tpsa	mw	hbd	hba	rot	logp	mr
0	TOX3021	CCOc1ccc2nc(S(N)(=O)=O)sc2c1	0.0	0.0	1.0	NaN	NaN	0.0	0.0	1.0	...	82.28	258.324	1.0	5.0	3.0	1.3424	62.1622

1 rows × 24 columns

Why these descriptors?

- **TPSA, LogP, MW:** Drug-likeness indicators (Lipinski's Rule of Five)
- **HBD/HBA:** Membrane permeability predictors
- **Morgan Fingerprints:** Capture structural similarity and substructure presence
- **Rotatable Bonds:** Molecular flexibility indicator

🔎 Key Observations

- SMILES strings were converted into numerical features because machine learning models cannot directly interpret chemical strings.
- Molecular descriptors such as TPSA, molecular weight, hydrogen bond counts, and rotatable bonds capture physicochemical properties of molecules.

- Morgan fingerprints (2048 bits) encode structural sub-patterns and substructures present in each molecule.
- A small number of invalid SMILES failed RDKit processing and were safely removed to maintain data quality.
- The combination of scalar descriptors and fingerprints provides both global chemical properties and detailed structural information for toxicity prediction.

4. Descriptor Distribution Analysis

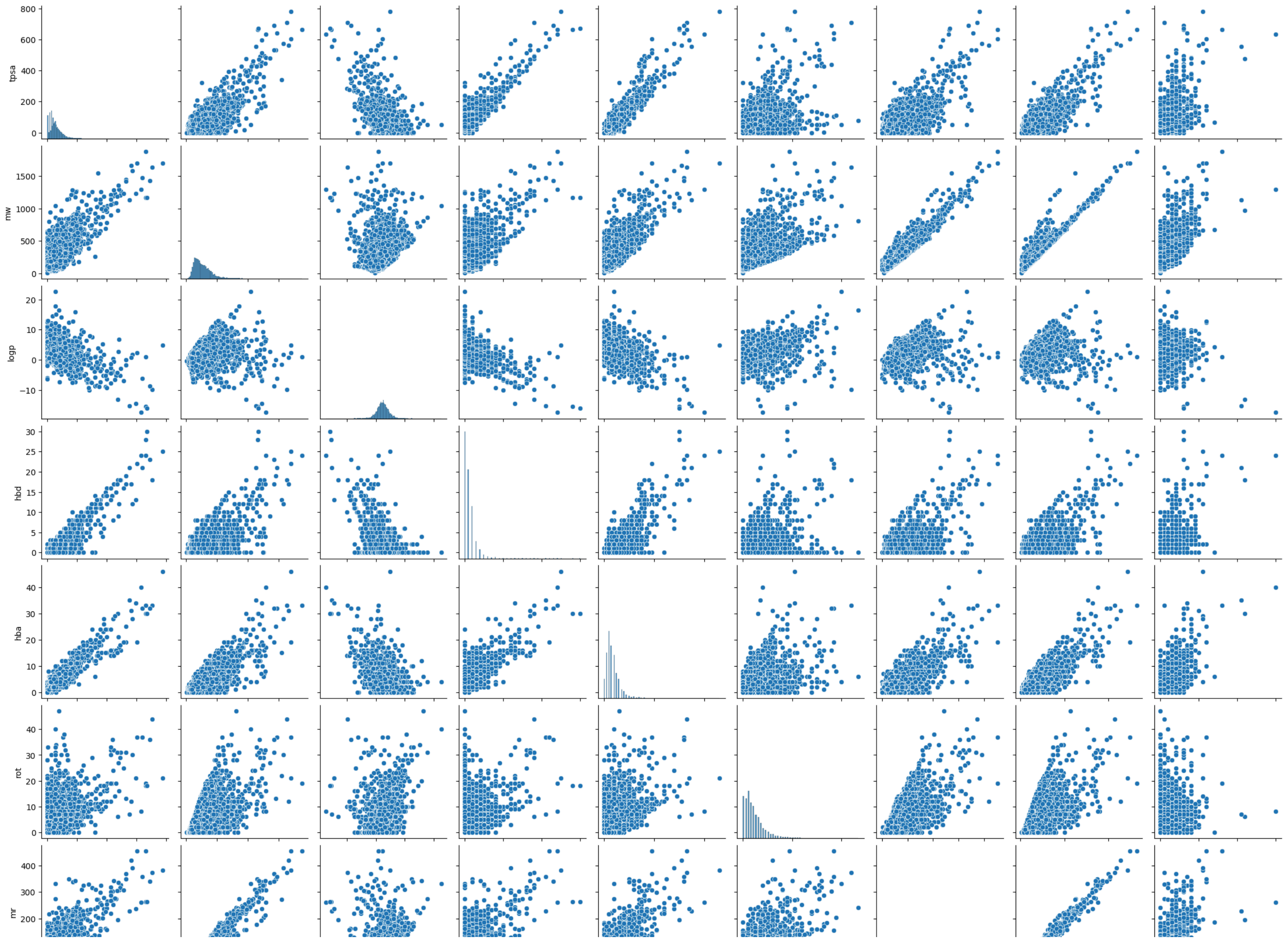
In [120…

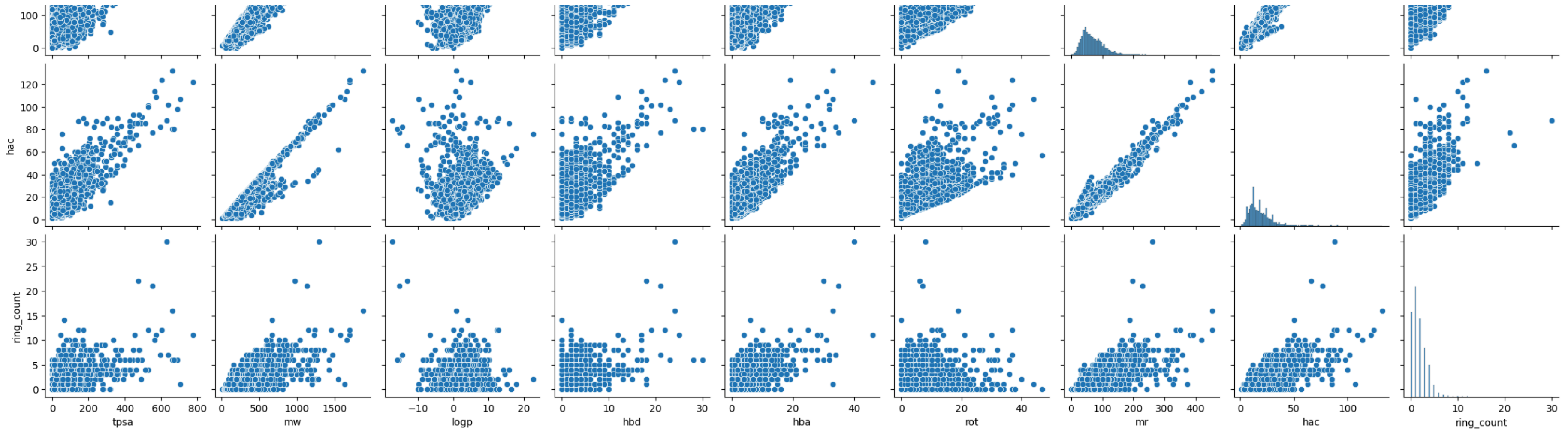
```
cols = ['tpsa', 'mw', 'logp', 'hbd', 'hba', 'rot', 'mr', 'hac', 'ring_count']
```

In [121…

```
sns.pairplot(df_tox[cols])

# Add title
plt.suptitle('Pairwise Relationship between Descriptors', y=1.02)

# Show plot
plt.show()
```

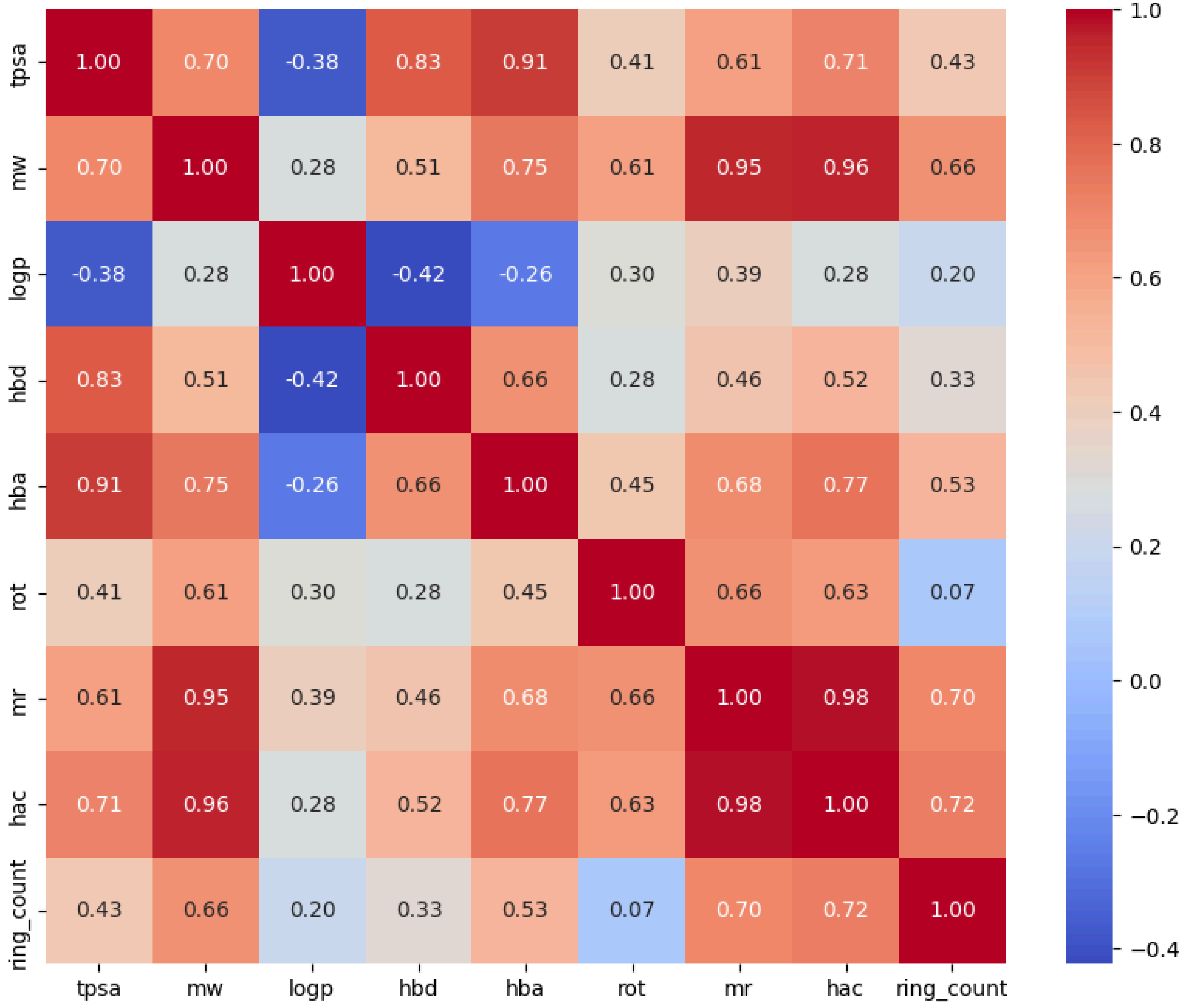
Pairwise Relationship between Descriptors





In [122…

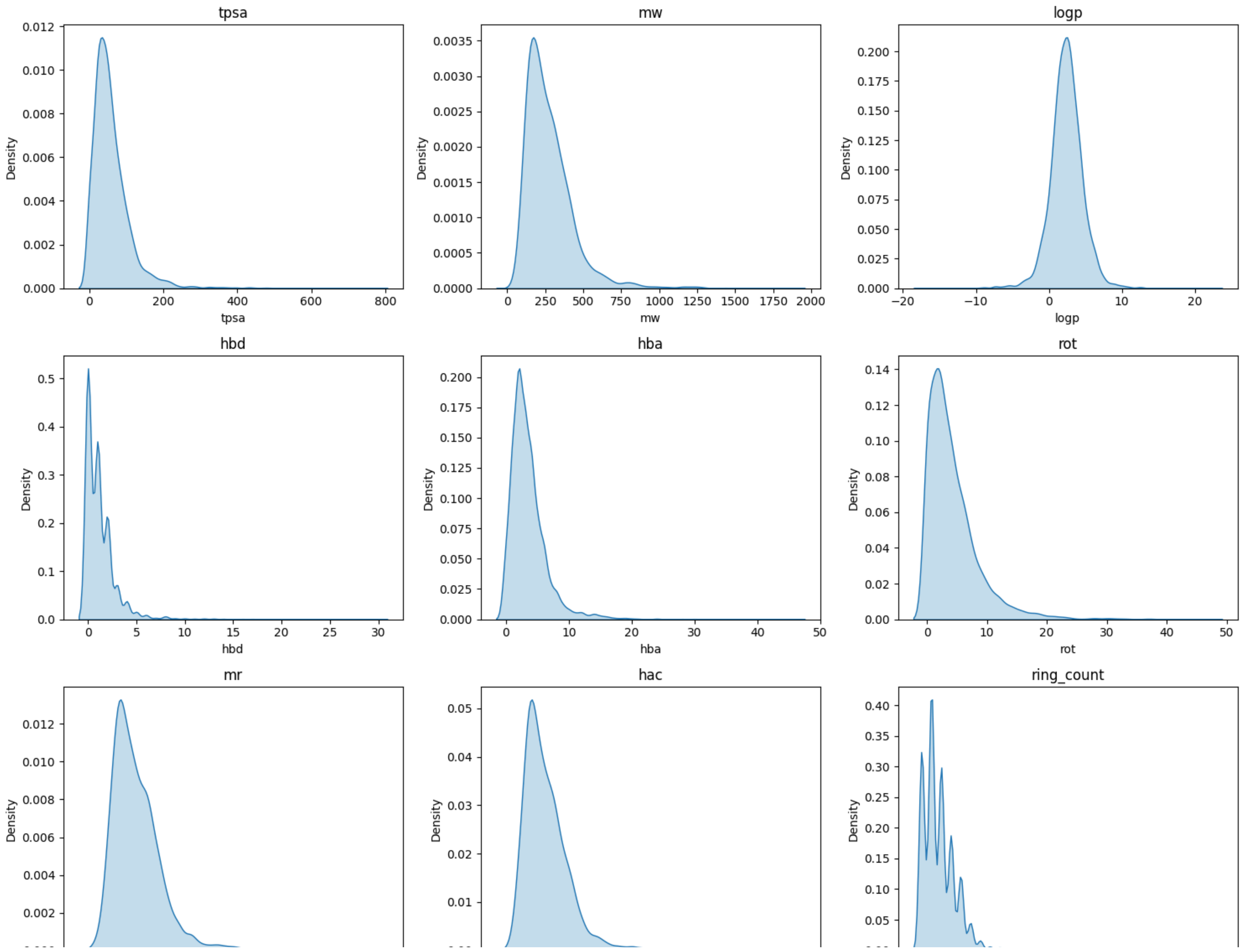
```python
corr_matrix = df_tox[cols].corr()

plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```

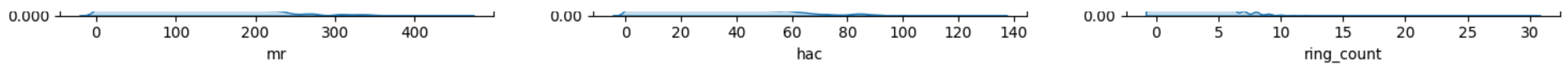


	tpsa	mw	logp	hbd	hba	rot	mr	hac	ring_count
tpsa	1.00	0.70	-0.38	0.83	0.91	0.41	0.61	0.71	0.43
mw	0.70	1.00	0.28	0.51	0.75	0.61	0.95	0.96	0.66
logp	-0.38	0.28	1.00	-0.42	-0.26	0.30	0.39	0.28	0.20
hbd	0.83	0.51	-0.42	1.00	0.66	0.28	0.46	0.52	0.33
hba	0.91	0.75	-0.26	0.66	1.00	0.45	0.68	0.77	0.53
rot	0.41	0.61	0.30	0.28	0.45	1.00	0.66	0.63	0.07
mr	0.61	0.95	0.39	0.46	0.68	0.66	1.00	0.98	0.70
hac	0.71	0.96	0.28	0.52	0.77	0.63	0.98	1.00	0.72
ring_count	0.43	0.66	0.20	0.33	0.53	0.07	0.70	0.72	1.00

In [123…

```
fig, axes = plt.subplots(3, 3, figsize=(15, 12))
axes = axes.flatten()

for i, col in enumerate(cols):
    sns.kdeplot(data=df_tox, x=col, fill=True, ax=axes[i])
    axes[i].set_title(col)

# Remove empty subplots if cols < 9
for j in range(len(cols), 9):
    fig.delaxes(axes[j])

plt.tight_layout()
# plt.savefig(r'..\plots\1.Descriptor_without_normalization.png')
plt.show()
```


tpsa
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0
200
400
600
800
tpsa
mw
Density
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
0
250
500
750
1000
1250
1500
1750
2000
mw
logp
Density
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
−20
−10
0
10
20
logp
hbd
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
5
10
15
20
25
30
hbd
hba
Density
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0
10
20
30
40
50
hba
rot
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
10
20
30
40
50
rot
mr
Density
0.002
0.004
0.006
0.008
0.010
0.012
hac
Density
0.01
0.02
0.03
0.04
0.05
ring_count
Density
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40



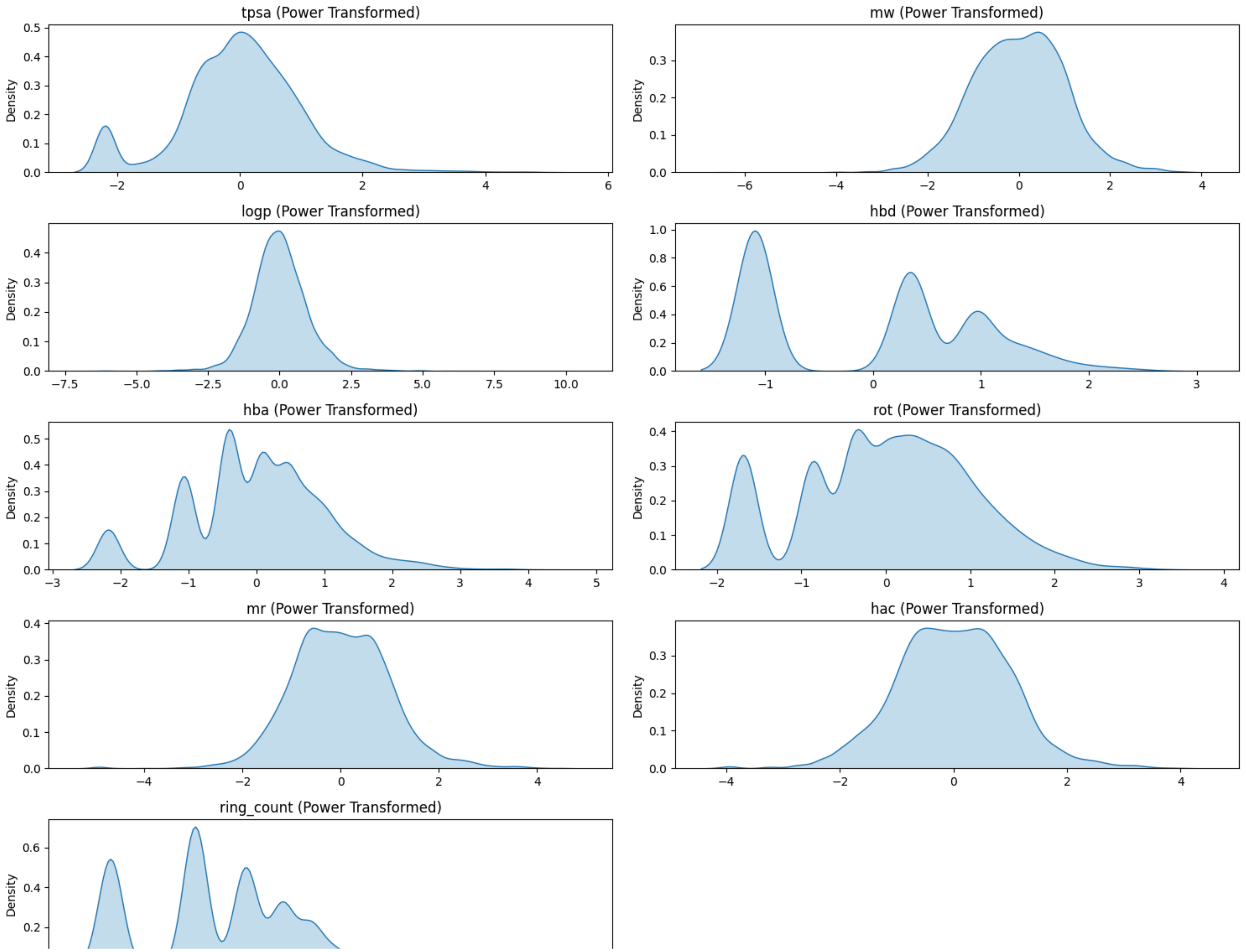
🔎 Key Observations

- Most molecular descriptors show right-skewed distributions, which is common for chemical property data.
- Properties such as molecular weight, TPSA, and rotatable bonds span a wide range of values.
- The presence of skewness suggests that direct use of raw values may negatively impact certain machine learning models.
- Visual inspection confirms the need for normalization to stabilize variance and improve model learning.

5. Power Transformation & Normalization

In [124…

```
n_rows = (len(cols) + 1) // 2
n_cols = 2

fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 12))
axes = axes.flatten()

for i, col in enumerate(cols):
    pt = PowerTransformer()
    transformed = pt.fit_transform(df_tox[[col]]).ravel()

    sns.kdeplot(x=transformed, fill=True, ax=axes[i])
    axes[i].set_title(f"{col} (Power Transformed)")

# remove unused axes
for j in range(len(cols), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
# plt.savefig(r'..\plots\2.Descriptor_with_normalization.png')
plt.show()
```


tpsa (Power Transformed)
mw (Power Transformed)
logp (Power Transformed)
hbd (Power Transformed)
hba (Power Transformed)
rot (Power Transformed)
mr (Power Transformed)
hac (Power Transformed)
ring_count (Power Transformed)
Density

In [125…

```
# Converting fingerprint to numpy array
def to_array(fp):
    arr = np.zeros(fp.GetNumBits(), dtype=np.uint8)
    DataStructs.ConvertToNumpyArray(fp, arr)
    return arr

fp_array = df_tox['fp'].apply(to_array)

df_tox['fp'] = fp_array
```

🔎 Key Observations

- PowerTransformer (Yeo-Johnson) normalization was applied to selected molecular descriptors.
- After transformation, feature distributions become more symmetric and closer to a normal shape.
- Normalization is especially beneficial for distance-based and linear models such as Logistic Regression and KNN.
- Both normalized and non-normalized feature sets were evaluated to compare model performance.

6. Model Training & Evaluation

In [126…

```
# Tox21 binary classification targets
targets = ['NR-AR', 'NR-AR-LBD', 'NR-AhR', 'NR-Aromatase',
           'NR-ER', 'NR-ER-LBD', 'NR-PPAR-gamma',
           'SR-ARE', 'SR-ATAD5', 'SR-HSE', 'SR-MMP', 'SR-p53'
            ]

# Scalar molecular descriptors
scalar_cols = ['tpsa', 'mw', 'hbd', 'hba', 'rot',
               'logp', 'ring_count', #'mr', 'hac'
                ]
```

In [127…

```
models_config = {
    "LR": Pipeline([("scaler", StandardScaler()),("clf", LogisticRegression(max_iter=1000))]),
    "KNN": Pipeline([("scaler", StandardScaler()),("clf", KNeighborsClassifier(n_neighbors=5))]),
    "RF": RandomForestClassifier(random_state=24),
    "XGB": XGBClassifier(random_state=24),
    "LGBM_Bal": LGBMClassifier(class_weight="balanced", verbosity=-1, random_state=24),
    "LGBM_Unbal": LGBMClassifier(is_unbalance=True, verbosity=-1, random_state=24)
}
```

In [128…

```
MODEL_DIR = "../models"
os.makedirs(MODEL_DIR, exist_ok=True)
```

In [129…

```
results = {t: {} for t in targets}
best_models_store_normalize = {}


for target in targets:
    print(f"\n================ Target: {target} ================")

    df_sub = df_tox[df_tox[target].notna()]
    y = df_sub[target].values

    X_scalar = df_sub[scalar_cols].values
    X_fp = np.stack(df_sub["fp"].values)

    # ---------------- Train / Test Split ----------------
    Xs_train, Xs_test, Xfp_train, Xfp_test, y_train, y_test = train_test_split(
        X_scalar,
        X_fp,
        y,
        test_size=0.2,
        stratify=y,
        random_state=24
    )

    # ---------------- Power Transform (FIT ONLY ON TRAIN) ----------------
    pt = PowerTransformer(method="yeo-johnson")
    Xs_train_pt = pt.fit_transform(Xs_train)
    Xs_test_pt = pt.transform(Xs_test)

    # ---------------- Combine Scalar + FP ----------------
    X_train = np.hstack([Xs_train_pt, Xfp_train])
```

```
X_test = np.hstack([Xs_test_pt, Xfp_test])

best_f1 = -1
best_model = None
best_model_name = None

for name, model in models_config.items():
    print(f"\n>>> Model: {name}")

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # -------- Metrics --------
    report = classification_report(y_test, y_pred, digits=4)
    cm = confusion_matrix(y_test, y_pred)
    f1_class1 = f1_score(y_test, y_pred, pos_label=1)

    print("Classification Report:")
    print(report)

    print("Confusion Matrix:")
    print(cm)

    print(f"F1 (Class 1): {f1_class1:.4f}")

    results[target][name] = f1_class1

    if f1_class1 > best_f1:
        best_f1 = f1_class1
        best_model = model
        best_model_name = name

# ---------------- Save Best Model + Transformer ----------------
model_path = f"{MODEL_DIR}/{target}_best_model.joblib"
transformer_path = f"{MODEL_DIR}/{target}_power_transformer.joblib"

joblib.dump(best_model, model_path)
joblib.dump(pt, transformer_path)

best_models_store_normalize[target] = {
    "model_name": best_model_name,
    "model_path": model_path,
    "transformer_path": transformer_path,
    "best_f1_class1": best_f1
}

print(f"\n💾 Saved best model: {best_model_name}")
print(f"🏆 Best F1 (Class 1): {best_f1:.4f}")
```

```
================ Target: NR-AR ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9714    0.9791    0.9753      1390
         1.0     0.4314    0.3548    0.3894        62

    accuracy                         0.9525      1452
   macro avg     0.7014    0.6670    0.6823      1452
weighted avg     0.9484    0.9525    0.9503      1452

Confusion Matrix:
[[1361   29]
 [  40   22]]
F1 (Class 1): 0.3894

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9686    1.0000    0.9841      1390
         1.0     1.0000    0.2742    0.4304        62

    accuracy                         0.9690      1452
   macro avg     0.9843    0.6371    0.7072      1452
weighted avg     0.9700    0.9690    0.9604      1452

Confusion Matrix:
[[1390    0]
 [  45   17]]
F1 (Class 1): 0.4304

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9726    0.9971    0.9847      1390
         1.0     0.8519    0.3710    0.5169        62
    accuracy                         0.9704      1452
   macro avg     0.9122    0.6840    0.7508      1452
weighted avg     0.9675    0.9704    0.9647      1452

Confusion Matrix:
[[1386    4]
 [  39   23]]
F1 (Class 1): 0.5169

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9726    0.9950    0.9836      1390
         1.0     0.7667    0.3710    0.5000        62

    accuracy                         0.9683      1452
   macro avg     0.8696    0.6830    0.7418      1452
weighted avg     0.9638    0.9683    0.9630      1452

Confusion Matrix:
[[1383    7]
 [  39   23]]
F1 (Class 1): 0.5000

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9730    0.9856    0.9793      1390
         1.0     0.5455    0.3871    0.4528        62

    accuracy                         0.9601      1452
   macro avg     0.7592    0.6864    0.7161      1452
weighted avg     0.9548    0.9601    0.9568      1452

Confusion Matrix:
[[1370   20]
 [  38   24]]
F1 (Class 1): 0.4528
```

```
>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9724    0.9871    0.9797      1390
         1.0     0.5610    0.3710    0.4466        62

    accuracy                         0.9607      1452
   macro avg     0.7667    0.6790    0.7131      1452
weighted avg     0.9548    0.9607    0.9569      1452

Confusion Matrix:
[[1372   18]
 [  39   23]]
F1 (Class 1): 0.4466

💾 Saved best model: RF
🏆 Best F1 (Class 1): 0.5169

================ Target: NR-AR-LBD ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9854    0.9831    0.9843      1304
         1.0     0.5600    0.5957    0.5773        47

    accuracy                         0.9697      1351
   macro avg     0.7727    0.7894    0.7808      1351
weighted avg     0.9706    0.9697    0.9701      1351

Confusion Matrix:
[[1282   22]
 [  19   28]]
F1 (Class 1): 0.5773

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support
         0.0     0.9745    0.9962    0.9852      1304
         1.0     0.7222    0.2766    0.4000        47

    accuracy                         0.9711      1351
   macro avg     0.8484    0.6364    0.6926      1351
weighted avg     0.9657    0.9711    0.9649      1351

Confusion Matrix:
[[1299    5]
 [  34   13]]
F1 (Class 1): 0.4000

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9826    0.9954    0.9890      1304
         1.0     0.8000    0.5106    0.6234        47

    accuracy                         0.9785      1351
   macro avg     0.8913    0.7530    0.8062      1351
weighted avg     0.9762    0.9785    0.9762      1351

Confusion Matrix:
[[1298    6]
 [  23   24]]
F1 (Class 1): 0.6234

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9855    0.9931    0.9893      1304
         1.0     0.7568    0.5957    0.6667        47

    accuracy                         0.9793      1351
   macro avg     0.8711    0.7944    0.8280      1351
weighted avg     0.9776    0.9793    0.9781      1351

Confusion Matrix:
[[1295    9]
```

```
 [  19   28]]
F1 (Class 1): 0.6667

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9870    0.9900    0.9885      1304
         1.0     0.6977    0.6383    0.6667        47

    accuracy                         0.9778      1351
   macro avg     0.8423    0.8142    0.8276      1351
weighted avg     0.9769    0.9778    0.9773      1351

Confusion Matrix:
[[1291   13]
 [  17   30]]
F1 (Class 1): 0.6667

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9870    0.9885    0.9877      1304
         1.0     0.6667    0.6383    0.6522        47

    accuracy                         0.9763      1351
   macro avg     0.8268    0.8134    0.8200      1351
weighted avg     0.9758    0.9763    0.9761      1351

Confusion Matrix:
[[1289   15]
 [  17   30]]
F1 (Class 1): 0.6522

💾 Saved best model: XGB
🏆 Best F1 (Class 1): 0.6667

================ Target: NR-AhR ================

>>> Model: LR
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9259    0.9307    0.9283      1155
         1.0     0.4595    0.4416    0.4503       154

    accuracy                         0.8732      1309
   macro avg     0.6927    0.6861    0.6893      1309
weighted avg     0.8710    0.8732    0.8721      1309

Confusion Matrix:
[[1075   80]
 [  86   68]]
F1 (Class 1): 0.4503

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8896    0.9974    0.9404      1155
         1.0     0.7857    0.0714    0.1310       154

    accuracy                         0.8885      1309
   macro avg     0.8376    0.5344    0.5357      1309
weighted avg     0.8774    0.8885    0.8452      1309

Confusion Matrix:
[[1152    3]
 [ 143   11]]
F1 (Class 1): 0.1310

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9048    0.9957    0.9481      1155
         1.0     0.8684    0.2143    0.3438       154

    accuracy                         0.9037      1309
   macro avg     0.8866    0.6050    0.6459      1309
weighted avg     0.9005    0.9037    0.8770      1309
```

```
Confusion Matrix:
[[1150    5]
 [ 121   33]]
F1 (Class 1): 0.3438

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9206    0.9835    0.9510      1155
         1.0     0.7467    0.3636    0.4891       154

    accuracy                         0.9106      1309
   macro avg     0.8336    0.6736    0.7201      1309
weighted avg     0.9001    0.9106    0.8967      1309

Confusion Matrix:
[[1136   19]
 [  98   56]]
F1 (Class 1): 0.4891

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9503    0.9273    0.9387      1155
         1.0     0.5385    0.6364    0.5833       154

    accuracy                         0.8930      1309
   macro avg     0.7444    0.7818    0.7610      1309
weighted avg     0.9019    0.8930    0.8968      1309

Confusion Matrix:
[[1071   84]
 [  56   98]]
F1 (Class 1): 0.5833

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support
         0.0     0.9486    0.9264    0.9374      1155
         1.0     0.5304    0.6234    0.5731       154

    accuracy                         0.8908      1309
   macro avg     0.7395    0.7749    0.7552      1309
weighted avg     0.8994    0.8908    0.8945      1309

Confusion Matrix:
[[1070   85]
 [  58   96]]
F1 (Class 1): 0.5731

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.5833

================ Target: NR-Aromatase ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9610    0.9837    0.9722      1103
         1.0     0.4706    0.2667    0.3404        60

    accuracy                         0.9467      1163
   macro avg     0.7158    0.6252    0.6563      1163
weighted avg     0.9357    0.9467    0.9396      1163

Confusion Matrix:
[[1085   18]
 [  44   16]]
F1 (Class 1): 0.3404

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9533    0.9991    0.9757      1103
         1.0     0.8571    0.1000    0.1791        60
    accuracy                         0.9527      1163
   macro avg     0.9052    0.5495    0.5774      1163
weighted avg     0.9483    0.9527    0.9346      1163

Confusion Matrix:
[[1102    1]
 [  54    6]]
F1 (Class 1): 0.1791

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9565    0.9973    0.9765      1103
         1.0     0.7692    0.1667    0.2740        60

    accuracy                         0.9544      1163
   macro avg     0.8629    0.5820    0.6252      1163
weighted avg     0.9469    0.9544    0.9402      1163

Confusion Matrix:
[[1100    3]
 [  50   10]]
F1 (Class 1): 0.2740

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9588    0.9918    0.9750      1103
         1.0     0.5909    0.2167    0.3171        60

    accuracy                         0.9518      1163
   macro avg     0.7749    0.6043    0.6461      1163
weighted avg     0.9398    0.9518    0.9411      1163

Confusion Matrix:
[[1094    9]
 [  47   13]]
F1 (Class 1): 0.3171
```

```
>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9641    0.9746    0.9693      1103
         1.0     0.4167    0.3333    0.3704        60

    accuracy                         0.9415      1163
   macro avg     0.6904    0.6540    0.6699      1163
weighted avg     0.9359    0.9415    0.9384      1163

Confusion Matrix:
[[1075   28]
 [  40   20]]
F1 (Class 1): 0.3704

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9644    0.9837    0.9740      1103
         1.0     0.5263    0.3333    0.4082        60

    accuracy                         0.9501      1163
   macro avg     0.7454    0.6585    0.6911      1163
weighted avg     0.9418    0.9501    0.9448      1163

Confusion Matrix:
[[1085   18]
 [  40   20]]
F1 (Class 1): 0.4082

💾 Saved best model: LGBM_Unbal
🏆 Best F1 (Class 1): 0.4082

================ Target: NR-ER ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support
         0.0     0.9058    0.8722    0.8887      1080
         1.0     0.3030    0.3797    0.3371       158

    accuracy                         0.8094      1238
   macro avg     0.6044    0.6260    0.6129      1238
weighted avg     0.8288    0.8094    0.8183      1238

Confusion Matrix:
[[942 138]
 [ 98  60]]
F1 (Class 1): 0.3371

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8869    0.9944    0.9376      1080
         1.0     0.7778    0.1329    0.2270       158

    accuracy                         0.8845      1238
   macro avg     0.8323    0.5637    0.5823      1238
weighted avg     0.8729    0.8845    0.8469      1238

Confusion Matrix:
[[1074    6]
 [ 137   21]]
F1 (Class 1): 0.2270

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8944    0.9963    0.9426      1080
         1.0     0.8857    0.1962    0.3212       158

    accuracy                         0.8942      1238
   macro avg     0.8901    0.5962    0.6319      1238
weighted avg     0.8933    0.8942    0.8633      1238

Confusion Matrix:
[[1076    4]
```

```
 [ 127   31]]
F1 (Class 1): 0.3212

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9048    0.9861    0.9437      1080
         1.0     0.7541    0.2911    0.4201       158

    accuracy                         0.8974      1238
   macro avg     0.8295    0.6386    0.6819      1238
weighted avg     0.8856    0.8974    0.8769      1238

Confusion Matrix:
[[1065   15]
 [ 112   46]]
F1 (Class 1): 0.4201

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9175    0.9065    0.9120      1080
         1.0     0.4094    0.4430    0.4255       158

    accuracy                         0.8473      1238
   macro avg     0.6634    0.6748    0.6688      1238
weighted avg     0.8527    0.8473    0.8499      1238

Confusion Matrix:
[[979 101]
 [ 88  70]]
F1 (Class 1): 0.4255

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9173    0.9037    0.9104      1080
         1.0     0.4023    0.4430    0.4217       158
    accuracy                         0.8449      1238
   macro avg     0.6598    0.6734    0.6661      1238
weighted avg     0.8516    0.8449    0.8481      1238

Confusion Matrix:
[[976 104]
 [ 88  70]]
F1 (Class 1): 0.4217

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.4255

================ Target: NR-ER-LBD ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9651    0.9833    0.9741      1320
         1.0     0.5111    0.3286    0.4000        70

    accuracy                         0.9504      1390
   macro avg     0.7381    0.6560    0.6871      1390
weighted avg     0.9422    0.9504    0.9452      1390

Confusion Matrix:
[[1298   22]
 [  47   23]]
F1 (Class 1): 0.4000

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9550    0.9970    0.9755      1320
         1.0     0.6667    0.1143    0.1951        70

    accuracy                         0.9525      1390
   macro avg     0.8108    0.5556    0.5853      1390
weighted avg     0.9405    0.9525    0.9362      1390
```

```
Confusion Matrix:
[[1316    4]
 [  62    8]]
F1 (Class 1): 0.1951

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9620    0.9970    0.9792      1320
         1.0     0.8182    0.2571    0.3913        70

    accuracy                         0.9597      1390
   macro avg     0.8901    0.6271    0.6852      1390
weighted avg     0.9547    0.9597    0.9496      1390

Confusion Matrix:
[[1316    4]
 [  52   18]]
F1 (Class 1): 0.3913

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9661    0.9932    0.9795      1320
         1.0     0.7273    0.3429    0.4660        70

    accuracy                         0.9604      1390
   macro avg     0.8467    0.6680    0.7227      1390
weighted avg     0.9541    0.9604    0.9536      1390

Confusion Matrix:
[[1311    9]
 [  46   24]]
F1 (Class 1): 0.4660

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support
         0.0     0.9727    0.9705    0.9716      1320
         1.0     0.4658    0.4857    0.4755        70

    accuracy                         0.9460      1390
   macro avg     0.7192    0.7281    0.7235      1390
weighted avg     0.9471    0.9460    0.9466      1390

Confusion Matrix:
[[1281   39]
 [  36   34]]
F1 (Class 1): 0.4755

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9712    0.9720    0.9716      1320
         1.0     0.4638    0.4571    0.4604        70

    accuracy                         0.9460      1390
   macro avg     0.7175    0.7146    0.7160      1390
weighted avg     0.9457    0.9460    0.9459      1390

Confusion Matrix:
[[1283   37]
 [  38   32]]
F1 (Class 1): 0.4604

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.4755

================ Target: NR-PPAR-gamma ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9756    0.9920    0.9838      1252
         1.0     0.3750    0.1622    0.2264        37
    accuracy                         0.9682      1289
   macro avg     0.6753    0.5771    0.6051      1289
weighted avg     0.9584    0.9682    0.9620      1289

Confusion Matrix:
[[1242   10]
 [  31    6]]
F1 (Class 1): 0.2264

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9728    1.0000    0.9862      1252
         1.0     1.0000    0.0541    0.1026        37

    accuracy                         0.9728      1289
   macro avg     0.9864    0.5270    0.5444      1289
weighted avg     0.9736    0.9728    0.9609      1289

Confusion Matrix:
[[1252    0]
 [  35    2]]
F1 (Class 1): 0.1026

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9728    1.0000    0.9862      1252
         1.0     1.0000    0.0541    0.1026        37

    accuracy                         0.9728      1289
   macro avg     0.9864    0.5270    0.5444      1289
weighted avg     0.9736    0.9728    0.9609      1289

Confusion Matrix:
[[1252    0]
 [  35    2]]
F1 (Class 1): 0.1026
```

```
>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9757    0.9952    0.9854      1252
         1.0     0.5000    0.1622    0.2449        37

    accuracy                         0.9713      1289
   macro avg     0.7379    0.5787    0.6151      1289
weighted avg     0.9621    0.9713    0.9641      1289

Confusion Matrix:
[[1246    6]
 [  31    6]]
F1 (Class 1): 0.2449

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9794    0.9896    0.9845      1252
         1.0     0.4583    0.2973    0.3607        37

    accuracy                         0.9697      1289
   macro avg     0.7189    0.6435    0.6726      1289
weighted avg     0.9645    0.9697    0.9666      1289

Confusion Matrix:
[[1239   13]
 [  26   11]]
F1 (Class 1): 0.3607

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9794    0.9888    0.9841      1252
         1.0     0.4400    0.2973    0.3548        37

    accuracy                         0.9690      1289
   macro avg     0.7097    0.6431    0.6695      1289
```

```
weighted avg     0.9639    0.9690    0.9660      1289

Confusion Matrix:
[[1238   14]
 [  26   11]]
F1 (Class 1): 0.3548

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.3607

================ Target: SR-ARE ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8839    0.8495    0.8664       977
         1.0     0.3496    0.4202    0.3816       188

    accuracy                         0.7803      1165
   macro avg     0.6167    0.6349    0.6240      1165
weighted avg     0.7977    0.7803    0.7882      1165

Confusion Matrix:
[[830 147]
 [109  79]]
F1 (Class 1): 0.3816

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8434    0.9980    0.9142       977
         1.0     0.7778    0.0372    0.0711       188

    accuracy                         0.8429      1165
   macro avg     0.8106    0.5176    0.4926      1165
weighted avg     0.8328    0.8429    0.7781      1165

Confusion Matrix:
[[975   2]
```

```
 [181   7]]
F1 (Class 1): 0.0711

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8604    0.9969    0.9237       977
         1.0     0.9091    0.1596    0.2715       188

    accuracy                         0.8618      1165
   macro avg     0.8848    0.5783    0.5976      1165
weighted avg     0.8683    0.8618    0.8184      1165

Confusion Matrix:
[[974   3]
 [158  30]]
F1 (Class 1): 0.2715

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8890    0.9754    0.9302       977
         1.0     0.7419    0.3670    0.4911       188

    accuracy                         0.8773      1165
   macro avg     0.8155    0.6712    0.7107      1165
weighted avg     0.8653    0.8773    0.8593      1165

Confusion Matrix:
[[953  24]
 [119  69]]
F1 (Class 1): 0.4911

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9140    0.8813    0.8973       977
         1.0     0.4798    0.5691    0.5207       188
    accuracy                         0.8309      1165
   macro avg     0.6969    0.7252    0.7090      1165
weighted avg     0.8439    0.8309    0.8366      1165

Confusion Matrix:
[[861 116]
 [ 81 107]]
F1 (Class 1): 0.5207

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9151    0.8710    0.8925       977
         1.0     0.4638    0.5798    0.5154       188

    accuracy                         0.8240      1165
   macro avg     0.6894    0.7254    0.7039      1165
weighted avg     0.8422    0.8240    0.8316      1165

Confusion Matrix:
[[851 126]
 [ 79 109]]
F1 (Class 1): 0.5154

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.5207

================ Target: SR-ATAD5 ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9682    0.9846    0.9763      1360
         1.0     0.3000    0.1698    0.2169        53

    accuracy                         0.9540      1413
   macro avg     0.6341    0.5772    0.5966      1413
weighted avg     0.9431    0.9540    0.9478      1413
```

```
Confusion Matrix:
[[1339   21]
 [  44    9]]
F1 (Class 1): 0.2169

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9658    0.9978    0.9816      1360
         1.0     0.6250    0.0943    0.1639        53

    accuracy                         0.9639      1413
   macro avg     0.7954    0.5461    0.5727      1413
weighted avg     0.9531    0.9639    0.9509      1413

Confusion Matrix:
[[1357    3]
 [  48    5]]
F1 (Class 1): 0.1639

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9652    1.0000    0.9823      1360
         1.0     1.0000    0.0755    0.1404        53

    accuracy                         0.9653      1413
   macro avg     0.9826    0.5377    0.5613      1413
weighted avg     0.9665    0.9653    0.9507      1413

Confusion Matrix:
[[1360    0]
 [  49    4]]
F1 (Class 1): 0.1404

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support
         0.0     0.9664    0.9949    0.9804      1360
         1.0     0.4615    0.1132    0.1818        53

    accuracy                         0.9618      1413
   macro avg     0.7140    0.5540    0.5811      1413
weighted avg     0.9475    0.9618    0.9505      1413

Confusion Matrix:
[[1353    7]
 [  47    6]]
F1 (Class 1): 0.1818

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9701    0.9787    0.9744      1360
         1.0     0.2927    0.2264    0.2553        53

    accuracy                         0.9505      1413
   macro avg     0.6314    0.6025    0.6148      1413
weighted avg     0.9447    0.9505    0.9474      1413

Confusion Matrix:
[[1331   29]
 [  41   12]]
F1 (Class 1): 0.2553

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9688    0.9824    0.9755      1360
         1.0     0.2941    0.1887    0.2299        53

    accuracy                         0.9526      1413
   macro avg     0.6315    0.5855    0.6027      1413
weighted avg     0.9435    0.9526    0.9476      1413

Confusion Matrix:
```

```
[[1336   24]
 [  43   10]]
F1 (Class 1): 0.2299

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.2553

================ Target: SR-HSE ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9528    0.9614    0.9571      1218
         1.0     0.2540    0.2162    0.2336        74

    accuracy                         0.9187      1292
   macro avg     0.6034    0.5888    0.5953      1292
weighted avg     0.9128    0.9187    0.9157      1292

Confusion Matrix:
[[1171   47]
 [  58   16]]
F1 (Class 1): 0.2336

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9492    0.9975    0.9728      1218
         1.0     0.7500    0.1216    0.2093        74

    accuracy                         0.9474      1292
   macro avg     0.8496    0.5596    0.5910      1292
weighted avg     0.9378    0.9474    0.9290      1292

Confusion Matrix:
[[1215    3]
 [  65    9]]
F1 (Class 1): 0.2093
```

```
>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9478    0.9992    0.9728      1218
         1.0     0.8750    0.0946    0.1707        74

    accuracy                         0.9474      1292
   macro avg     0.9114    0.5469    0.5718      1292
weighted avg     0.9436    0.9474    0.9269      1292

Confusion Matrix:
[[1217    1]
 [  67    7]]
F1 (Class 1): 0.1707

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9543    0.9934    0.9735      1218
         1.0     0.6667    0.2162    0.3265        74

    accuracy                         0.9489      1292
   macro avg     0.8105    0.6048    0.6500      1292
weighted avg     0.9378    0.9489    0.9364      1292

Confusion Matrix:
[[1210    8]
 [  58   16]]
F1 (Class 1): 0.3265

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9625    0.9696    0.9661      1218
         1.0     0.4308    0.3784    0.4029        74

    accuracy                         0.9358      1292
   macro avg     0.6966    0.6740    0.6845      1292
```

```
weighted avg     0.9321    0.9358    0.9338      1292

Confusion Matrix:
[[1181   37]
 [  46   28]]
F1 (Class 1): 0.4029

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9600    0.9655    0.9628      1218
         1.0     0.3731    0.3378    0.3546        74

    accuracy                         0.9296      1292
   macro avg     0.6666    0.6517    0.6587      1292
weighted avg     0.9264    0.9296    0.9279      1292

Confusion Matrix:
[[1176   42]
 [  49   25]]
F1 (Class 1): 0.3546

💾 Saved best model: LGBM_Bal
🏆 Best F1 (Class 1): 0.4029

================ Target: SR-MMP ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9114    0.9263    0.9188       977
         1.0     0.5714    0.5217    0.5455       184

    accuracy                         0.8622      1161
   macro avg     0.7414    0.7240    0.7321      1161
weighted avg     0.8575    0.8622    0.8596      1161

Confusion Matrix:
[[905  72]
```

```
 [ 88  96]]
F1 (Class 1): 0.5455

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8526    0.9949    0.9183       977
         1.0     0.7619    0.0870    0.1561       184

    accuracy                         0.8510      1161
   macro avg     0.8073    0.5409    0.5372      1161
weighted avg     0.8383    0.8510    0.7975      1161

Confusion Matrix:
[[972   5]
 [168  16]]
F1 (Class 1): 0.1561

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support

         0.0     0.8862    0.9959    0.9378       977
         1.0     0.9365    0.3207    0.4777       184

    accuracy                         0.8889      1161
   macro avg     0.9113    0.6583    0.7078      1161
weighted avg     0.8941    0.8889    0.8649      1161

Confusion Matrix:
[[973   4]
 [125  59]]
F1 (Class 1): 0.4777

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9209    0.9652    0.9425       977
         1.0     0.7518    0.5598    0.6417       184
    accuracy                         0.9009      1161
   macro avg     0.8364    0.7625    0.7921      1161
weighted avg     0.8941    0.9009    0.8949      1161

Confusion Matrix:
[[943  34]
 [ 81 103]]
F1 (Class 1): 0.6417

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9411    0.9161    0.9284       977
         1.0     0.6095    0.6957    0.6497       184

    accuracy                         0.8811      1161
   macro avg     0.7753    0.8059    0.7891      1161
weighted avg     0.8886    0.8811    0.8843      1161

Confusion Matrix:
[[895  82]
 [ 56 128]]
F1 (Class 1): 0.6497

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9436    0.9243    0.9338       977
         1.0     0.6373    0.7065    0.6701       184

    accuracy                         0.8898      1161
   macro avg     0.7904    0.8154    0.8020      1161
weighted avg     0.8950    0.8898    0.8920      1161

Confusion Matrix:
[[903  74]
 [ 54 130]]
F1 (Class 1): 0.6701
```

```
💾 Saved best model: LGBM_Unbal
🏆 Best F1 (Class 1): 0.6701

================ Target: SR-p53 ================

>>> Model: LR
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9529    0.9574    0.9552      1269
         1.0     0.3165    0.2941    0.3049        85

    accuracy                         0.9158      1354
   macro avg     0.6347    0.6258    0.6300      1354
weighted avg     0.9130    0.9158    0.9144      1354

Confusion Matrix:
[[1215   54]
 [  60   25]]
F1 (Class 1): 0.3049

>>> Model: KNN
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9393    0.9992    0.9683      1269
         1.0     0.7500    0.0353    0.0674        85

    accuracy                         0.9387      1354
   macro avg     0.8446    0.5173    0.5179      1354
weighted avg     0.9274    0.9387    0.9118      1354

Confusion Matrix:
[[1268    1]
 [  82    3]]
F1 (Class 1): 0.0674

>>> Model: RF
Classification Report:
              precision    recall  f1-score   support
         0.0     0.9441    0.9976    0.9701      1269
         1.0     0.7692    0.1176    0.2041        85

    accuracy                         0.9424      1354
   macro avg     0.8567    0.5576    0.5871      1354
weighted avg     0.9331    0.9424    0.9220      1354

Confusion Matrix:
[[1266    3]
 [  75   10]]
F1 (Class 1): 0.2041

>>> Model: XGB
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9500    0.9882    0.9687      1269
         1.0     0.5588    0.2235    0.3193        85

    accuracy                         0.9402      1354
   macro avg     0.7544    0.6059    0.6440      1354
weighted avg     0.9254    0.9402    0.9279      1354

Confusion Matrix:
[[1254   15]
 [  66   19]]
F1 (Class 1): 0.3193

>>> Model: LGBM_Bal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9612    0.9574    0.9593      1269
         1.0     0.4000    0.4235    0.4114        85

    accuracy                         0.9239      1354
   macro avg     0.6806    0.6905    0.6854      1354
weighted avg     0.9260    0.9239    0.9249      1354

Confusion Matrix:
```

```
[[1215   54]
 [  49   36]]
F1 (Class 1): 0.4114

>>> Model: LGBM_Unbal
Classification Report:
              precision    recall  f1-score   support

         0.0     0.9651    0.9582    0.9616      1269
         1.0     0.4362    0.4824    0.4581        85

    accuracy                         0.9284      1354
   macro avg     0.7006    0.7203    0.7099      1354
weighted avg     0.9319    0.9284    0.9300      1354

Confusion Matrix:
[[1216   53]
 [  44   41]]
F1 (Class 1): 0.4581

💾 Saved best model: LGBM_Unbal
🏆 Best F1 (Class 1): 0.4581
```

In [130…

```
results_df_normalize = pd.DataFrame(results).T

numeric_cols = results_df_normalize.select_dtypes(include="number")

results_df_normalize["Best_Model"] = numeric_cols.idxmax(axis=1)
results_df_normalize["Best_CV_F1"] = numeric_cols.max(axis=1)

results_df_normalize.to_csv("results_with_normalize.csv", index=False)
display(results_df_normalize.sort_values("Best_CV_F1", ascending=False))
```

	LR	KNN	RF	XGB	LGBM_Bal	LGBM_Unbal	Best_Model	Best_CV_F1
SR-MMP	0.545455	0.156098	0.477733	0.641745	0.649746	0.670103	LGBM_Unbal	0.670103
NR-AR-LBD	0.577320	0.400000	0.623377	0.666667	0.666667	0.652174	XGB	0.666667
NR-AhR	0.450331	0.130952	0.343750	0.489083	0.583333	0.573134	LGBM_Bal	0.583333
SR-ARE	0.381643	0.071066	0.271493	0.491103	0.520681	0.515366	LGBM_Bal	0.520681
NR-AR	0.389381	0.430380	0.516854	0.500000	0.452830	0.446602	RF	0.516854
NR-ER-LBD	0.400000	0.195122	0.391304	0.466019	0.475524	0.460432	LGBM_Bal	0.475524
SR-p53	0.304878	0.067416	0.204082	0.319328	0.411429	0.458101	LGBM_Unbal	0.458101
NR-ER	0.337079	0.227027	0.321244	0.420091	0.425532	0.421687	LGBM_Bal	0.425532
NR-Aromatase	0.340426	0.179104	0.273973	0.317073	0.370370	0.408163	LGBM_Unbal	0.408163
SR-HSE	0.233577	0.209302	0.170732	0.326531	0.402878	0.354610	LGBM_Bal	0.402878
NR-PPAR-gamma	0.226415	0.102564	0.102564	0.244898	0.360656	0.354839	LGBM_Bal	0.360656
SR-ATAD5	0.216867	0.163934	0.140351	0.181818	0.255319	0.229885	LGBM_Bal	0.255319

Why Power Transformation?

- Molecular descriptors are naturally right-skewed
- Linear models (LR) and distance-based models (KNN) assume normally distributed features
- Yeo-Johnson handles zero values (unlike Box-Cox)
- **Critical:** Fitted only on training data to prevent leakage

🔎 Key Observations

- Each toxicity assay was treated as an independent binary classification problem.
- Multiple machine learning models were evaluated, including Logistic Regression, Random Forest, KNN, XGBoost, and LightGBM.
- Stratified K-Fold cross-validation was used to handle class imbalance during model evaluation.
- F1 score was selected as the primary metric to balance precision and recall for the minority (toxic) class.

- LightGBM (balanced or unbalanced) and Random Forest models consistently showed the best performance across most assays.
- For each assay, the model with the highest cross-validated F1 score was selected as the final best model.
- Normalized molecular descriptors generally improved or maintained model performance compared to raw features.
- The selected best models were retrained on the full available dataset for each assay.
- Final trained models were saved to disk for future prediction, reuse, or deployment workflows.

Limitations & Future Work

- **Missing data:** 7-26% of labels missing per assay (handled by per-target modeling)
- **Class imbalance:** Minority class as low as 2.88% (addressed via F1 metric and balanced sampling)
- **External validation:** Models not tested on external datasets
- **Feature engineering:** Could explore 3D descriptors, graph neural networks, or transformer-based molecular embeddings

Future improvements:

- Multi-task learning to leverage correlations between assays
- Ensemble stacking of best models

Project Conclusion

This project successfully built machine learning models to predict chemical toxicity across 12 different biological tests using molecular structures.

What we did:

- Converted 7,823 chemical formulas (SMILES) into numerical features that computers can understand
- Created two types of features: basic properties (like weight, size) and structural fingerprints (molecular patterns)
- Applied power transformation to make data more normally distributed, which helps models learn better
- Tested 6 different machine learning algorithms on each of the 12 toxicity tests

- Selected the best-performing model for each test based on F1 score (balances precision and recall)

Key findings:

- **LightGBM and Random Forest** worked best overall, handling the imbalanced data effectively
- **F1 scores ranged from 0.26 to 0.67**, with SR-MMP toxicity being easiest to predict (67% F1)
- **Power transformation improved results** for models like Logistic Regression and KNN
- **Proper data handling mattered** - we transformed features ONLY on training data to avoid leakage

Challenges addressed:

- **Missing labels:** 7-26% of molecules weren't tested for every assay - solved by building separate models for each target
- **Class imbalance:** Only 3-16% of molecules were toxic - handled using balanced sampling and F1 scoring
- **Feature scaling:** Different molecular properties had very different ranges - normalized them properly

Real-world impact: These models can now screen new chemicals for toxicity **before** expensive lab testing, speeding up drug discovery and chemical safety assessment. All models are saved and ready for deployment in production environments.

Future improvements:

- Test on external datasets to verify generalization
- Try advanced techniques like graph neural networks that directly learn from molecular structures
- Use multi-task learning to leverage relationships between different toxicity tests