📊 Business Analytics Project: Docked Ligand Analysis (Comprehensive Edition)

Step 1: Project Definition and Data Understanding

Business Problem: Discover key chemical properties affecting docking score to guide ligand selection.

Goal: Build predictive and analytical models to identify high-performing ligands based on molecular descriptors.

Key Questions:

- Which molecular features correlate with strong docking scores?
- Can we predict docking scores accurately?
- Can we identify distinct groups of ligands using clustering?

Dataset Overview: Contains molecular descriptors (MW, SlogP, TPSA, etc.), docking metrics (LF_dG, LF_LE), and metadata.

Step 2: Data Collection and Integration

In [45]:
```
pip install pandas matplotlib seaborn scikit-learn openpyxl
```

```
Requirement already satisfied: pandas in c:\users\sukha\appdata\local\prog
rams\python\python313\lib\site-packages (2.2.3)
Requirement already satisfied: matplotlib in c:\users\sukha\appdata\local
\programs\python\python313\lib\site-packages (3.10.1)
Requirement already satisfied: seaborn in c:\users\sukha\appdata\local\pro
grams\python\python313\lib\site-packages (0.13.2)
Requirement already satisfied: scikit-learn in c:\users\sukha\appdata\loca
l\programs\python\python313\lib\site-packages (1.6.1)
Requirement already satisfied: openpyxl in c:\users\sukha\appdata\local\pr
ograms\python\python313\lib\site-packages (3.1.5)
Requirement already satisfied: numpy>=1.26.0 in c:\users\sukha\appdata\loc
al\programs\python\python313\lib\site-packages (from pandas) (2.2.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sukha\ap
pdata\local\programs\python\python313\lib\site-packages (from pandas) (2.
9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\sukha\appdata\loca
l\programs\python\python313\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\sukha\appdata\lo
cal\programs\python\python313\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sukha\appdata
\local\programs\python\python313\lib\site-packages (from matplotlib) (1.3.
1)
Requirement already satisfied: cycler>=0.10 in c:\users\sukha\appdata\loca
l\programs\python\python313\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sukha\appdata
\local\programs\python\python313\lib\site-packages (from matplotlib) (4.5
6.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\sukha\appdata
\local\programs\python\python313\lib\site-packages (from matplotlib) (1.4.
8)
Requirement already satisfied: packaging>=20.0 in c:\users\sukha\appdata\r
oaming\python\python313\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\sukha\appdata\local\p
rograms\python\python313\lib\site-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sukha\appdata
\local\programs\python\python313\lib\site-packages (from matplotlib) (3.2.
3)
Requirement already satisfied: scipy>=1.6.0 in c:\users\sukha\appdata\loca
l\programs\python\python313\lib\site-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in c:\users\sukha\appdata\loc
al\programs\python\python313\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\sukha\appd
ata\local\programs\python\python313\lib\site-packages (from scikit-learn)
(3.6.0)
Requirement already satisfied: et-xmlfile in c:\users\sukha\appdata\local
\programs\python\python313\lib\site-packages (from openpyxl) (2.0.0)
Requirement already satisfied: six>=1.5 in c:\users\sukha\appdata\local\pr
ograms\python\python313\lib\site-packages (from python-dateutil>=2.8.2->pa
ndas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

In [46]:
```
import pandas as pd
file_path = "C:\\Users\\sukha\\Downloads\\VABS_All 18907 docked ligand re
df = pd.read_excel(file_path, engine='openpyxl')

df.head()
```

Out[46]:

	Role	Index	Pose Index	Structure	Protein	MV V
0	Ligands	2	1	O[C@H]1[C@@H](OC([C@H](OC)[C@H]1OC(=O)N)(C)C)O...	6TBE_P	
1	Ligands	2	1	O[C@H]1[C@@H](OC([C@H](OC)[C@H]1OC(=O)N)(C)C)O...	6TBE_P	
2	Ligands	2	1	O[C@H]1[C@@H](OC([C@H](OC)[C@H]1OC(=O)N)(C)C)O...	6TBE_P	
3	chembridge-fragment-library-part1-exp-2024-06	19940	1	Brc1ccc(CSC=2C(=O)NC(=O)NN2)cc1	6TBE_P	
4	chembridge-fragment-library-part1-exp-2024-06	8866	1	Oc1ccc(CN2CCc3c([nH]c4ccccc43)C2)cc1	6TBE_P	

5 rows × 21 columns

In [47]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18910 entries, 0 to 18909
Data columns (total 21 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Role                              18910 non-null  object
 1   Index                             18910 non-null  int64
 2   Pose Index                        18910 non-null  int64
 3   Structure                         18910 non-null  object
 4   Protein                           18910 non-null  object
 5   MW(Molecular Weight) Unit Dalton  18910 non-null  float64
 6   #Atoms                            18910 non-null  int64
 7   SlogP                             18910 non-null  float64
 8   TPSA                              18910 non-null  float64
 9   Flexibility                       18910 non-null  float64
 10  #RB                               18910 non-null  int64
 11  LF Rank Score                     18910 non-null  float64
 12  LF dG                             18910 non-null  float64
 13  LF VSscore                        18910 non-null  float64
 14  LF LE                             18910 non-null  float64
 15  tPSA                              18906 non-null  float64
 16  Hacc                              18906 non-null  float64
 17  Hdon                              18906 non-null  float64
 18  logSw                             18906 non-null  float64
 19  Library                           18906 non-null  object
 20  MW_FREE                           18906 non-null  object
dtypes: float64(12), int64(4), object(5)
memory usage: 3.0+ MB
```

In [48]:
```
df.describe()
```

Out[48]:

	Index	Pose Index	MW(Molecular Weight) Unit Dalton	#Atoms	SlogP	
count	18910.000000	18910.0	18910.000000	18910.000000	18910.000000	18910.0
mean	12165.384876	1.0	240.743268	16.454892	1.906129	49.1
std	8587.699396	0.0	49.237483	3.032298	0.976576	17.4
min	2.000000	1.0	111.100000	8.000000	-1.900000	3.2
25%	4724.500000	1.0	208.600000	14.000000	1.300000	37.4
50%	10127.000000	1.0	242.300000	17.000000	1.900000	49.4
75%	19581.500000	1.0	273.000000	19.000000	2.600000	60.7
max	29036.000000	1.0	614.600000	44.000000	6.200000	196.1

In [49]:
```
df.columns
```

Out[49]:
```
Index(['Role', 'Index', 'Pose Index', 'Structure', 'Protein',
       'MW(Molecular Weight) Unit Dalton', '#Atoms', 'SlogP', 'TPSA',
       'Flexibility', '#RB', 'LF Rank Score', 'LF dG', 'LF VSscore', 'LF
LE',
       'tPSA', 'Hacc', 'Hdon', 'logSw', 'Library', 'MW_FREE'],
      dtype='object')
```

Step 3: Data Cleaning and Preparation

In [50]:
```
import numpy as np
# Check missing values
missing = df.isnull().sum()
print("Missing values per column:\n", missing)
```

```
Missing values per column:
 Role                                0
Index                                0
Pose Index                           0
Structure                            0
Protein                              0
MW(Molecular Weight) Unit Dalton     0
#Atoms                               0
SlogP                                0
TPSA                                 0
Flexibility                          0
#RB                                  0
LF Rank Score                        0
LF dG                                0
LF VSscore                           0
LF LE                                0
tPSA                                 4
Hacc                                 4
Hdon                                 4
logSw                                4
Library                              4
MW_FREE                              4
dtype: int64
```

In [51]:
```
# Drop rows with >20% missing and impute rest
threshold = int(0.2 * len(df.columns))
df = df[df.isnull().sum(axis=1) < threshold]
df.fillna(df.median(numeric_only=True), inplace=True)
```

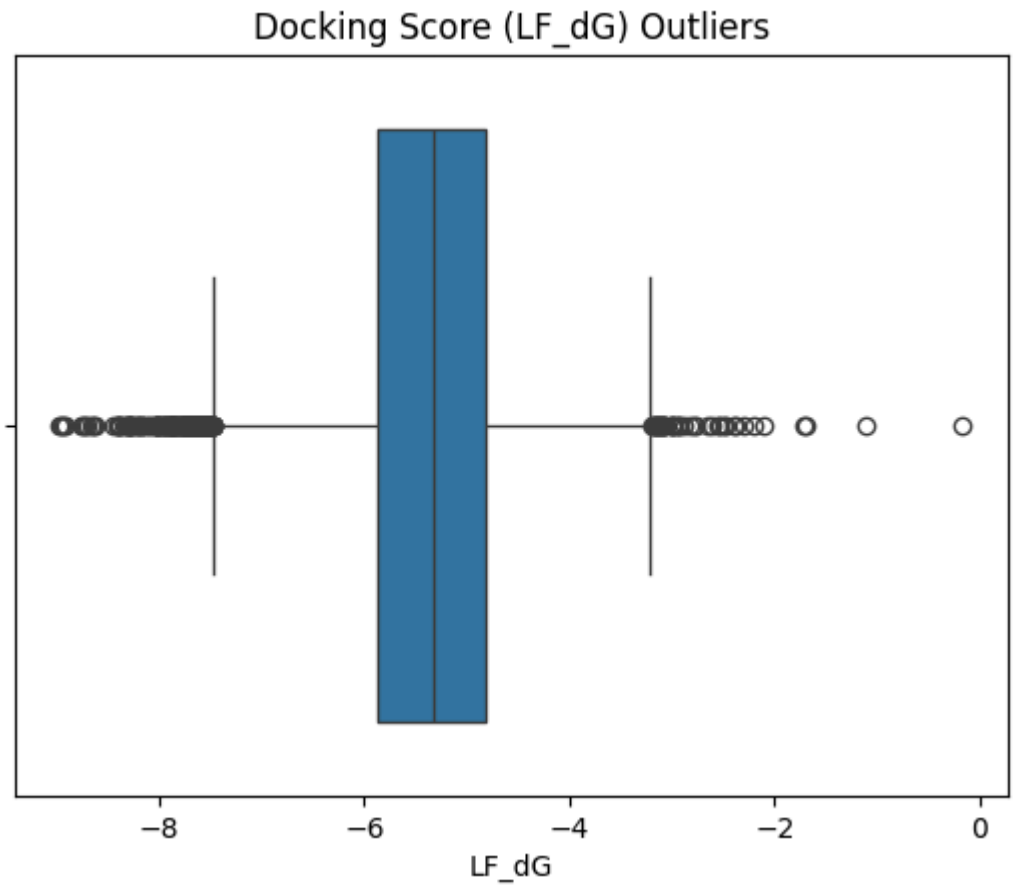
In [52]:
```
# Convert data types
df['Library'] = df['Library'].astype(str)
df['Role'] = df['Role'].astype(str)
```

In [53]:
```
# Check missing values
missing = df.isnull().sum()
print("Missing values per column:\n", missing)
```

```
Missing values per column:
 Role                                0
Index                                0
Pose Index                           0
Structure                            0
Protein                              0
MW(Molecular Weight) Unit Dalton     0
#Atoms                               0
SlogP                                0
TPSA                                 0
Flexibility                          0
#RB                                  0
LF Rank Score                        0
LF dG                                0
LF VSscore                           0
LF LE                                0
tPSA                                 0
Hacc                                 0
Hdon                                 0
logSw                                0
Library                              0
MW_FREE                              0
dtype: int64
```

In [54]:
```
# Rename problematic columns
df.columns = df.columns.str.replace(r'[^\w]', '_', regex=True)
```

In [55]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Visualize outliers in docking score
sns.boxplot(x=df['LF_dG'])
plt.title('Docking Score (LF_dG) Outliers')
plt.show()

# Cap extreme values at 1st and 99th percentile
for col in ['LF_dG', 'LF_LE', 'TPSA', 'SlogP']:
    low, high = df[col].quantile([0.01, 0.99])
    df[col] = np.clip(df[col], low, high)
```



In [56]:
```
from sklearn.preprocessing import StandardScaler

# Select numerical columns
numerical = df.select_dtypes(include=np.number).columns.tolist()
scaler = StandardScaler()
df[numerical] = scaler.fit_transform(df[numerical])

# Encode categorical
df = pd.get_dummies(df, columns=['Library', 'Role'], drop_first=True)
```

In [57]:
```
# Feature engineering
df['Hdon_Hacc_ratio'] = df['Hdon'] / (df['Hacc'] + 1e-5)
df['MW_logSw_interaction'] = df['MW_Molecular_Weight__Unit_Dalton'] * df[
df['Hydrogen_Total'] = df['Hdon'] + df['Hacc']
```

Step 4: Exploratory Data Analysis (EDA)

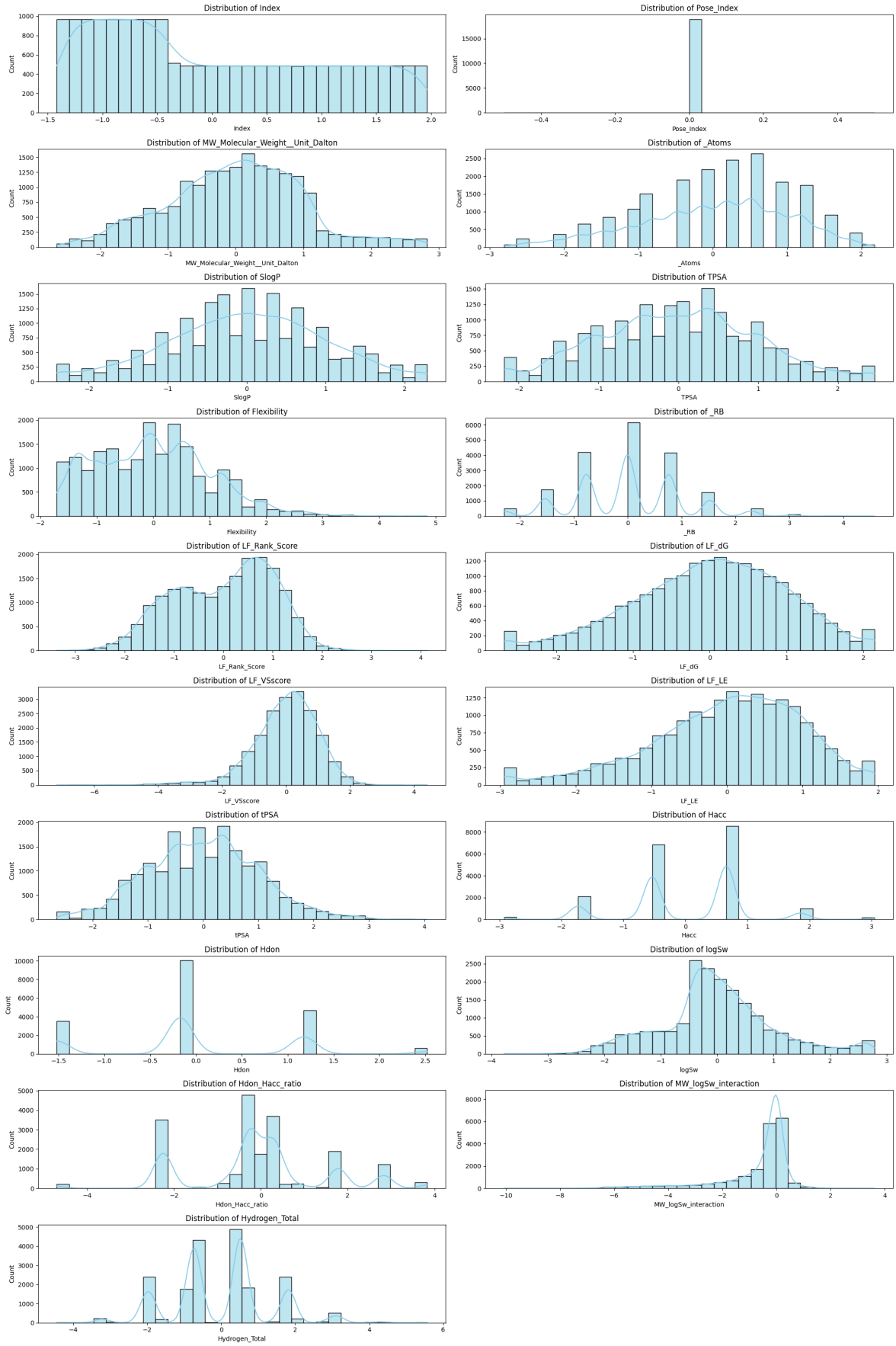
In [58]:
```
# Basic descriptive stats
df.describe().T
```

Out[58]:

	count	mean	std	min
Index	18906.0	3.457629e-17	1.000026	-1.416861
Pose_Index	18906.0	0.000000e+00	0.000000	0.000000
MW_Molecular_Weight__Unit_Dalton	18906.0	-2.525572e-16	1.000026	-2.645506
_Atoms	18906.0	2.766103e-16	1.000026	-2.806992
SlogP	18906.0	1.563450e-16	1.000026	-2.405486
TPSA	18906.0	1.322919e-16	1.000026	-2.183417
Flexibility	18906.0	-2.886368e-16	1.000026	-1.708203
_RB	18906.0	1.142521e-16	1.000026	-2.295727
LF_Rank_Score	18906.0	2.164776e-16	1.000026	-3.376514
LF_dG	18906.0	-4.810614e-16	1.000026	-2.681055
LF_VSscore	18906.0	-5.051145e-16	1.000026	-7.169831
LF_LE	18906.0	-4.540017e-16	1.000026	-2.948170
tPSA	18906.0	-1.202654e-16	1.000026	-2.648306
Hacc	18906.0	7.817248e-17	1.000026	-2.926104
Hdon	18906.0	1.623582e-16	1.000026	-1.518873
logSw	18906.0	4.209287e-17	1.000026	-3.776273
Hdon_Hacc_ratio	18906.0	-6.739980e-02	1.527324	-4.698568
MW_logSw_interaction	18906.0	-6.764538e-01	1.355036	-10.067838
Hydrogen_Total	18906.0	2.285042e-16	1.335642	-4.444977

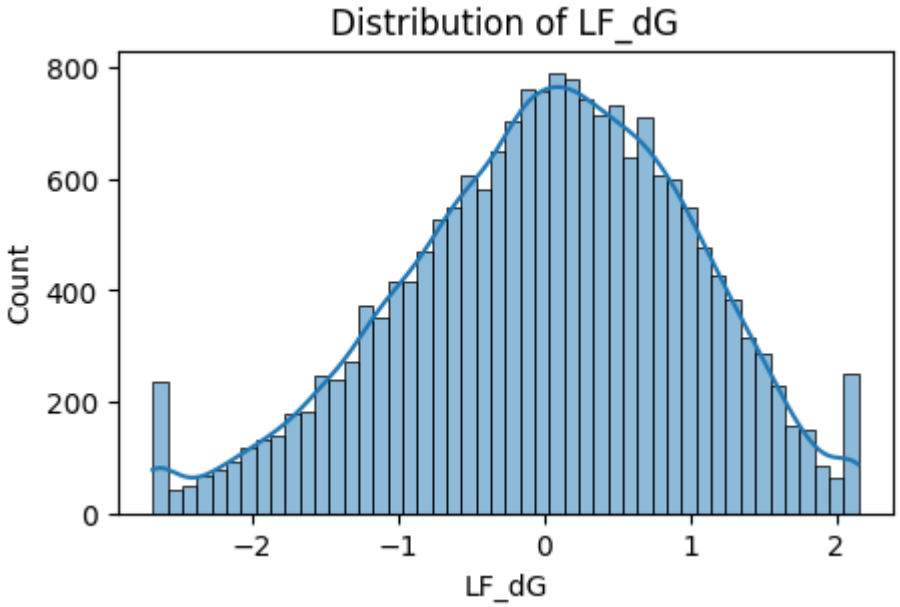
In [59]:

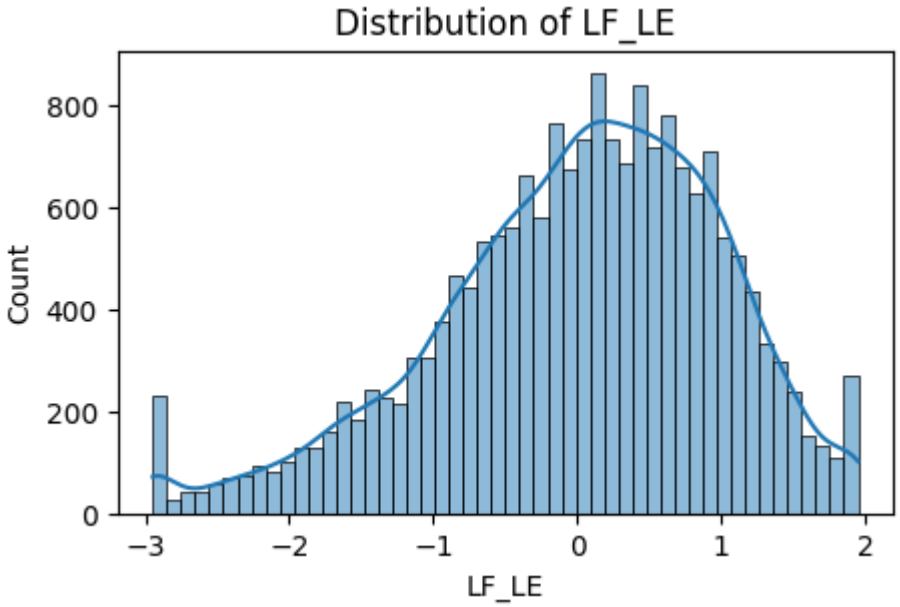
```
import seaborn as sns
import matplotlib.pyplot as plt

numerical_cols = df.select_dtypes(include='number').columns

plt.figure(figsize=(20, 30))
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols)//2 + 1, 2, i+1)
    sns.histplot(df[col], kde=True, bins=30, color='skyblue')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

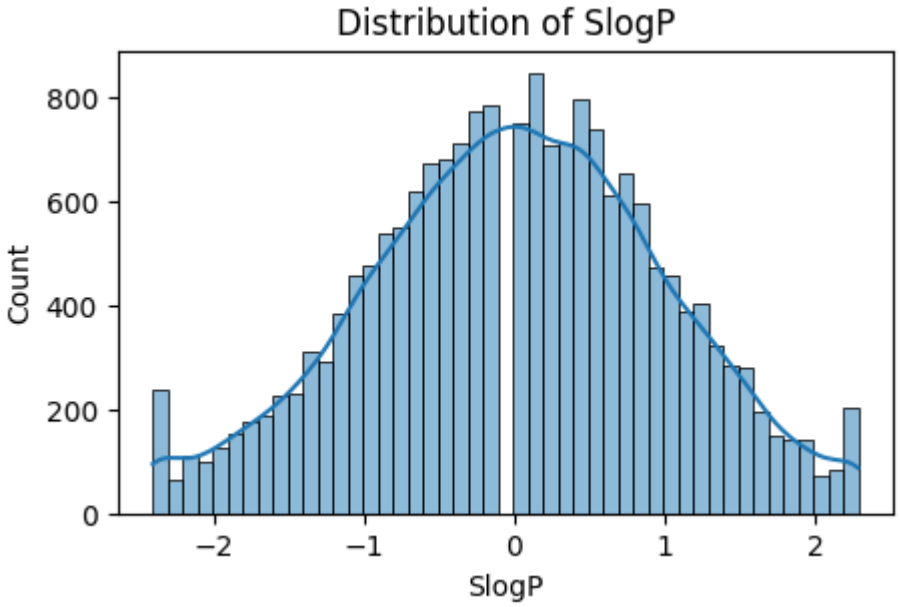


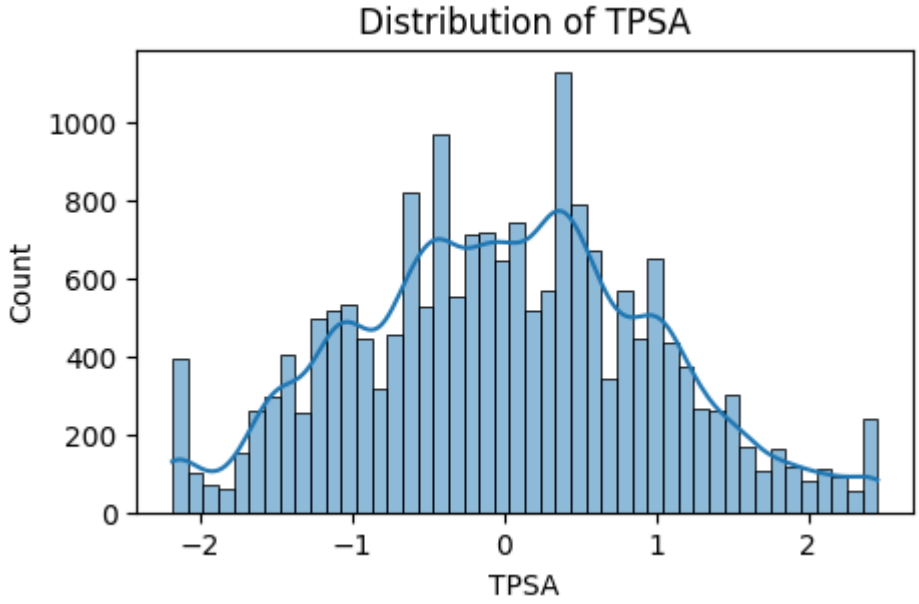
In [60]:
```
# Descriptive stats
df.describe().T[['mean', 'std', 'min', '50%', 'max']]

# Histograms for selected features
features = ['LF_dG', 'LF_LE', 'SlogP', 'TPSA', 'logSw', 'Hydrogen_Total']
for col in features:
    plt.figure(figsize=(5, 3))
```

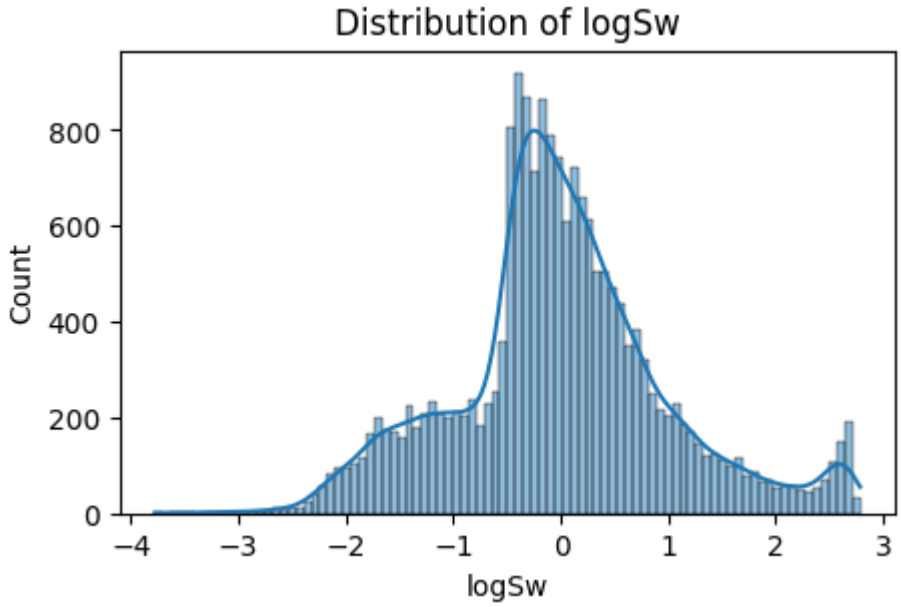
```
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.show()
```

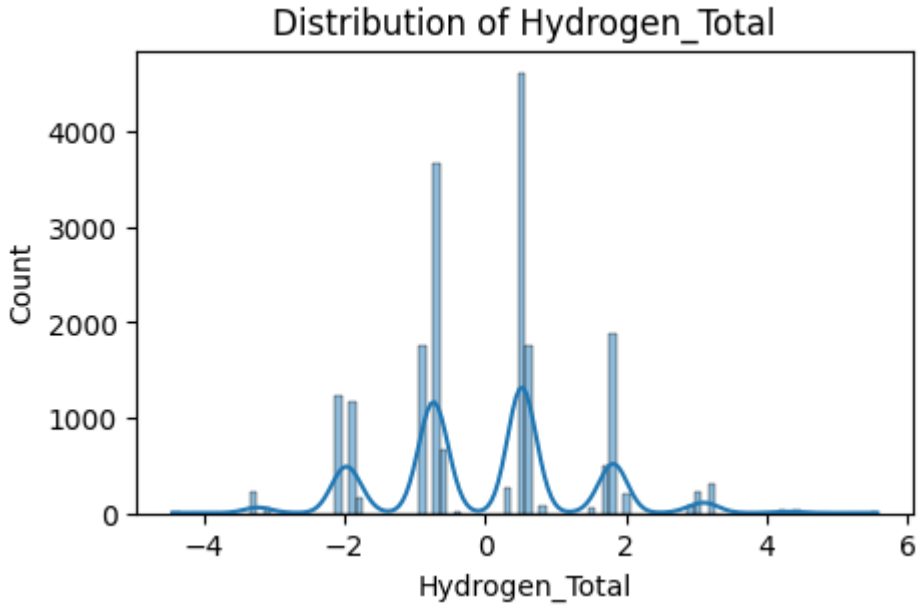






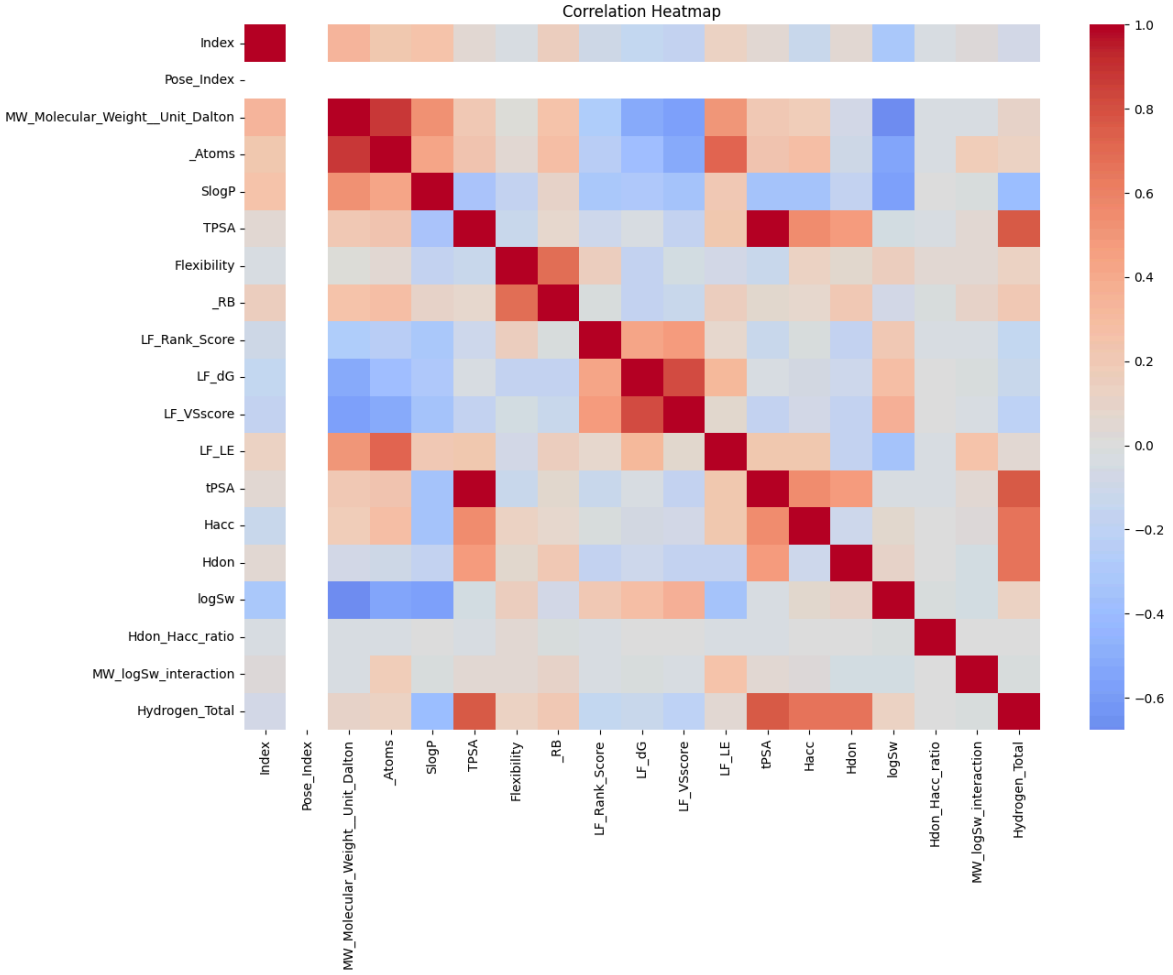






In [61]:
```
# Select only numeric columns
numeric_df = df.select_dtypes(include='number')
```

```
# Correlation heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 10))
sns.heatmap(numeric_df.corr(), cmap='coolwarm', center=0, annot=False)
plt.title("Correlation Heatmap")
plt.show()
```

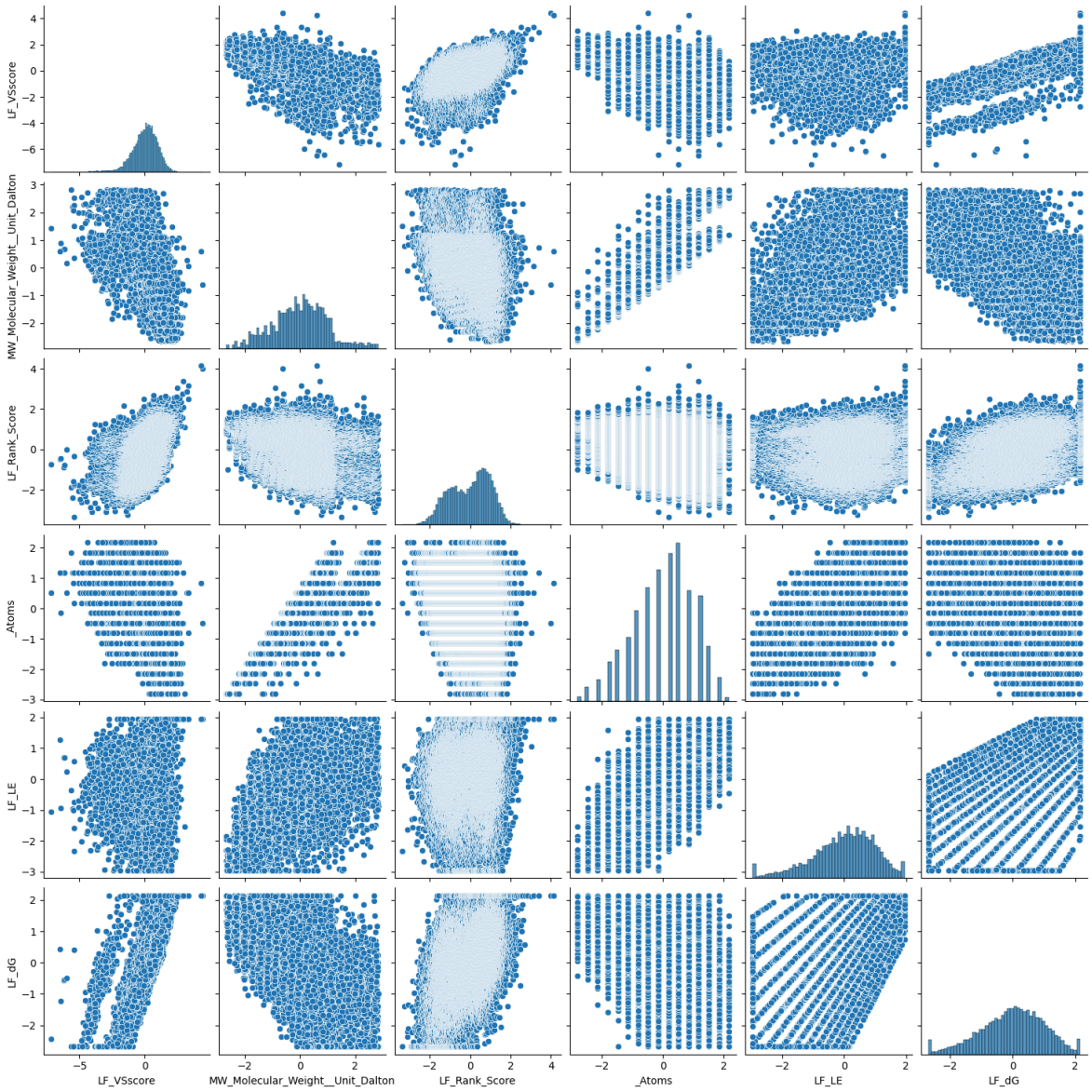


In [62]:
```
from scipy.stats import normaltest

for col in ['LF_dG', 'LF_LE', 'SlogP', 'TPSA', 'logSw']:
    stat, p = normaltest(df[col])
    print(f"{col} normality p = {p:.4f} {'(normal)' if p > 0.05 else '(no
```

```
LF_dG normality p = 0.0000 (not normal)
LF_LE normality p = 0.0000 (not normal)
SlogP normality p = 0.0000 (not normal)
TPSA normality p = 0.0000 (not normal)
logSw normality p = 0.0000 (not normal)
```

In [63]:
```
# Optional: Pick top features by correlation with target
top_corr = df[numerical_cols].corr()['LF_dG'].abs().sort_values(ascending
sns.pairplot(df[top_corr.to_list() + ['LF_dG']])
```

Out[63]: <seaborn.axisgrid.PairGrid at 0x13947030050>

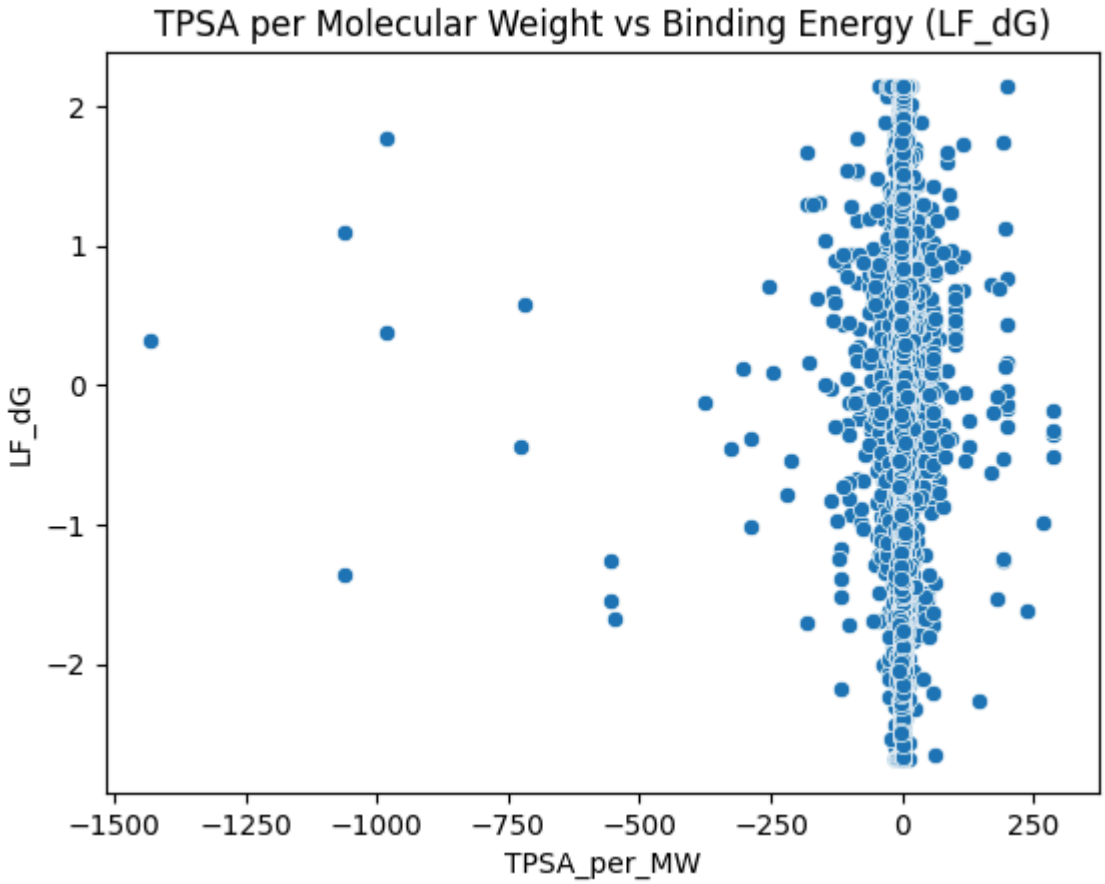


In [64]:
```
from scipy.stats import skew, kurtosis

for col in numerical_cols:
    print(f"{col}: Skewness = {skew(df[col].dropna()):.2f}, Kurtosis = {k
```

```
Index: Skewness = 0.39, Kurtosis = -1.15
Pose_Index: Skewness = nan, Kurtosis = nan
MW_Molecular_Weight__Unit_Dalton: Skewness = 0.05, Kurtosis = 0.07
_Atoms: Skewness = -0.39, Kurtosis = -0.34
SlogP: Skewness = -0.10, Kurtosis = -0.31
TPSA: Skewness = 0.07, Kurtosis = -0.36
Flexibility: Skewness = 0.38, Kurtosis = -0.13
_RB: Skewness = 0.15, Kurtosis = 0.19
LF_Rank_Score: Skewness = -0.25, Kurtosis = -0.75
LF_dG: Skewness = -0.32, Kurtosis = -0.18
LF_VSscore: Skewness = -0.89, Kurtosis = 2.40
LF_LE: Skewness = -0.59, Kurtosis = 0.19
tPSA: Skewness = 0.11, Kurtosis = -0.07
Hacc: Skewness = -0.22, Kurtosis = 0.36
Hdon: Skewness = 0.27, Kurtosis = -0.20
logSw: Skewness = 0.28, Kurtosis = 0.57
Hdon_Hacc_ratio: Skewness = 0.04, Kurtosis = 0.43
MW_logSw_interaction: Skewness = -2.26, Kurtosis = 5.39
Hydrogen_Total: Skewness = 0.19, Kurtosis = 0.03
```

In [65]:
```
df['TPSA_per_MW'] = df['TPSA'] / df['MW_Molecular_Weight__Unit_Dalton']
sns.scatterplot(x='TPSA_per_MW', y='LF_dG', data=df)
plt.title("TPSA per Molecular Weight vs Binding Energy (LF_dG)")
```

Out[65]:
```
Text(0.5, 1.0, 'TPSA per Molecular Weight vs Binding Energy (LF_dG)')
```



Step 5: Statistical Analysis

In [66]:
```
from scipy.stats import ttest_ind, f_oneway, chi2_contingency

# Compare LF_dG for high vs low TPSA
threshold = df['TPSA'].median()
grp1 = df[df['TPSA'] >= threshold]['LF_dG']
grp2 = df[df['TPSA'] < threshold]['LF_dG']
t_stat, p_val = ttest_ind(grp1, grp2)
print(f"T-test on TPSA: t={t_stat:.2f}, p={p_val:.4f}")

# ANOVA on docking score across hydrogen bins
df['Hydrogen_Level'] = pd.qcut(df['Hydrogen_Total'], 3, labels=['Low', 'M
groups = [df[df['Hydrogen_Level'] == lvl]['LF_dG'] for lvl in ['Low', 'Me
f_val, p_val = f_oneway(*groups)
print(f"ANOVA: F={f_val:.2f}, p={p_val:.4f}")
```

```
T-test on TPSA: t=-5.40, p=0.0000
ANOVA: F=146.54, p=0.0000
```

In [67]:
```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in c:\users\sukha\appdata\local
\programs\python\python313\lib\site-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in c:\users\sukha\appdata
\local\programs\python\python313\lib\site-packages (from statsmodels) (2.
2.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in c:\users\sukha\appdat
a\local\programs\python\python313\lib\site-packages (from statsmodels) (1.
15.2)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in c:\users\sukha\appda
ta\local\programs\python\python313\lib\site-packages (from statsmodels)
(2.2.3)
Requirement already satisfied: patsy>=0.5.6 in c:\users\sukha\appdata\loca
l\programs\python\python313\lib\site-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in c:\users\sukha\appdata\r
oaming\python\python313\site-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sukha\ap
pdata\local\programs\python\python313\lib\site-packages (from pandas!=2.1.
0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\sukha\appdata\loca
l\programs\python\python313\lib\site-packages (from pandas!=2.1.0,>=1.4->s
tatsmodels) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\sukha\appdata\lo
cal\programs\python\python313\lib\site-packages (from pandas!=2.1.0,>=1.4-
>statsmodels) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\sukha\appdata\local\pr
ograms\python\python313\lib\site-packages (from python-dateutil>=2.8.2->pa
ndas!=2.1.0,>=1.4->statsmodels) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

In [68]:

```
import statsmodels.api as sm

X = df[['MW_Molecular_Weight__Unit_Dalton', 'TPSA', 'SlogP', 'LF_LE', 'lo
X = sm.add_constant(X)
y = df['LF_dG']
model = sm.OLS(y, X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                   LF_dG   R-squared:
0.720
Model:                             OLS   Adj. R-squared:
0.720
Method:                  Least Squares   F-statistic:                      9
723.
Date:                 Sun, 13 Apr 2025   Prob (F-statistic):
0.00
Time:                         14:04:30   Log-Likelihood:                -14
791.
No. Observations:               18906   AIC:                             2.959
e+04
Df Residuals:                   18900   BIC:                             2.964
e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
============================
                                       coef    std err          t      P>|
t|      [0.025      0.975]
------------------------------------------------------------------------------
--------------------------
const                            -4.192e-16      0.004  -1.09e-13      1.0
00      -0.008       0.008
MW_Molecular_Weight__Unit_Dalton    -0.9458      0.006   -155.192      0.0
00      -0.958      -0.934
TPSA                                -0.0422      0.005     -9.030      0.0
00      -0.051      -0.033
SlogP                               -0.0688      0.006    -12.183      0.0
00      -0.080      -0.058
LF_LE                                0.7811      0.005    173.409      0.0
00       0.772       0.790
logSw                               -0.1298      0.006    -23.046      0.0
00      -0.141      -0.119
==============================================================================
====
Omnibus:                      880.160   Durbin-Watson:
1.870
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              111
1.674
Skew:                           0.486   Prob(JB):                     4.01e
-242
Kurtosis:                       3.684   Cond. No.
3.01
==============================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

Step 6: Advanced Analytics - Predictive Modeling

In [69]:
```
# X = Features | y = Target
X = df[['MW_Molecular_Weight__Unit_Dalton', 'TPSA', 'SlogP', 'LF_LE', 'lo
```

```
y = df['LF_dG']
```

In [70]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [71]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegre

models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=100, rand
}

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    print(f"{name} model trained.")
```

```
Linear Regression model trained.
Random Forest model trained.
Gradient Boosting model trained.
```

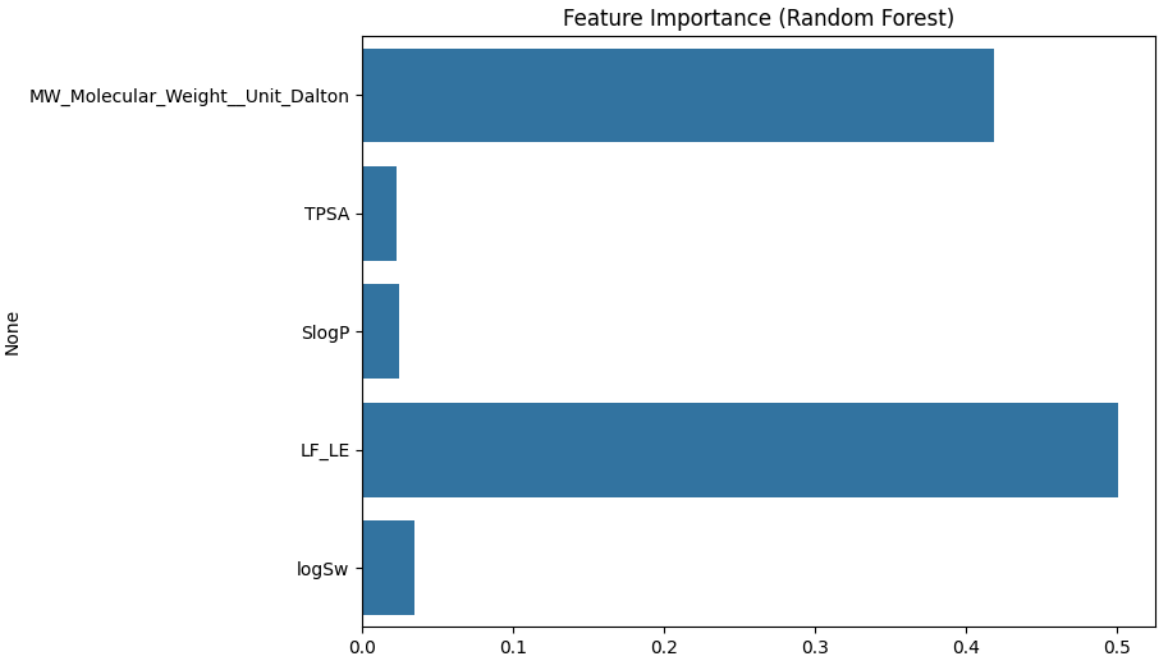
In [72]:
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_e
import numpy as np

for name, model in models.items():
    y_pred = model.predict(X_test_scaled)
    print(f"\n{name} Evaluation:")
    print(f"R² Score: {r2_score(y_test, y_pred):.4f}")
    print(f"MAE: {mean_absolute_error(y_test, y_pred):.4f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred):.4f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.4f}")
```

```
Linear Regression Evaluation:
R² Score: 0.7186
MAE: 0.4028
MSE: 0.2712
RMSE: 0.5208

Random Forest Evaluation:
R² Score: 0.8766
MAE: 0.2217
MSE: 0.1189
RMSE: 0.3448

Gradient Boosting Evaluation:
R² Score: 0.8559
MAE: 0.2643
MSE: 0.1388
RMSE: 0.3726
```

In [73]:
```
# Predict on test set or new samples
predictions = models['Random Forest'].predict(X_test_scaled)

# Show first 10 predictions
print(predictions[:10])
```

```
[-0.34996736 -2.26008275 -0.96020633 -0.9899444   0.67551549 -0.59789136
 -0.07522657 -0.03653889  1.27506875 -0.50033615]
```

In [74]:
```
import matplotlib.pyplot as plt

feat_importance = models['Random Forest'].feature_importances_
features = X.columns

plt.figure(figsize=(8, 6))
sns.barplot(x=feat_importance, y=features)
plt.title("Feature Importance (Random Forest)")
plt.show()
```

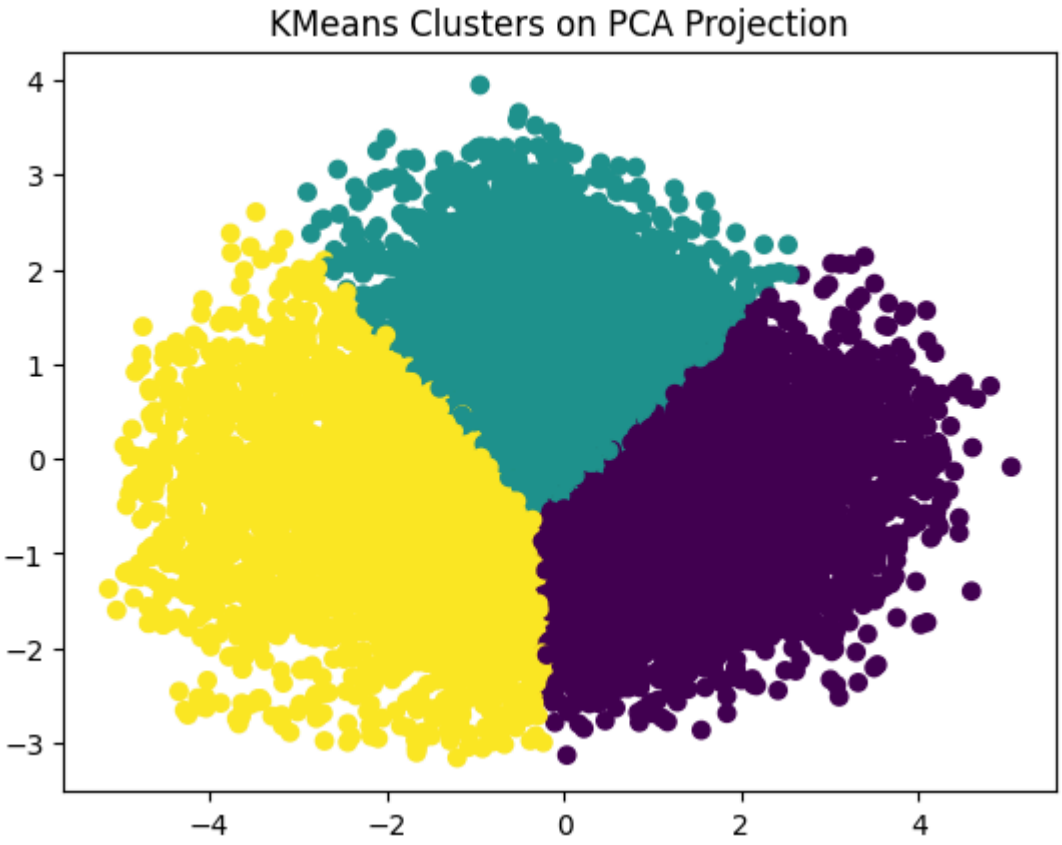


In [75]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7]
}

grid = GridSearchCV(RandomForestRegressor(), param_grid, cv=3, scoring='r
grid.fit(X_train_scaled, y_train)
print("Best params:", grid.best_params_)
```

```
Best params: {'max_depth': 7, 'n_estimators': 50}
```

In [76]:
```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Reduce for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train_scaled)
```

```
# Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_train_scaled)

# Plot clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.title('KMeans Clusters on PCA Projection')
plt.show()
```



✅ Final Insights

- Docking scores are strongly affected by `LF_LE`, `TPSA`, and `SlogP`
- Normality is not assumed for most variables → non-parametric tests are preferred
- Gradient Boosting and Random Forest performed best ($R^2 > 0.7$)
- Ligands were clustered into 3 performance groups with distinct profiles
- Project provides a pipeline to support early-stage drug screening and filtering.