In [3]:
```
import pandas as pd
import seaborn as sns
from sklearn.ensemble import IsolationForest
# import warning.ignore
```

In [4]:
```
file_path = r'C:\Users\hp\Downloads\PropsThreeFinal.csv'

# Read the CSV file into a pandas DataFrame
df = pd.read_csv(file_path)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7240 entries, 0 to 7239
Data columns (total 13 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Peptide                      7240 non-null   object 
 1   LogP                         7240 non-null   float64
 2   Molecular Weight             7240 non-null   float64
 3   Number Of Atoms              7240 non-null   int64  
 4   QED SCORE                    7240 non-null   float64
 5   IsoElectric Point            7240 non-null   float64
 6   MolarExtinction Coefficient  7240 non-null   int64  
 7   HydrogenBond                 7240 non-null   int64  
 8   Affinity                     7240 non-null   float64
 9   BBB_Prediction               7240 non-null   int64  
 10  Hydrophobicity               7240 non-null   float64
 11  Lipophilicity                7240 non-null   float64
 12  Toxicity                     7240 non-null   float64
dtypes: float64(8), int64(4), object(1)
memory usage: 735.4+ KB
```

In [9]:
```
df= df.dropna()
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Peptide', 'LogP', 'Molecular Weight', 'Number Of Atoms', 'QED SCOR
E',
       'IsoElectric Point', 'MolarExtinction Coefficient', 'HydrogenBond',
       'Affinity', 'BBB_Prediction', 'Hydrophobicity', 'Lipophilicity',
       'Toxicity'],
      dtype='object')
```

In [ ]:
```

```

In [10]:
```
anomaly_inputs = ['HydrogenBond','Affinity']
```

In [11]:
```
model_IF = IsolationForest(contamination = 0.1, random_state = 42)
```

In [12]:
```
model_IF.fit(df[anomaly_inputs])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
X does not have valid feature names, but IsolationForest was fitted with fea
ture names
  warnings.warn(
```

Out[12]:
```
IsolationForest(contamination=0.1, random_state=42)
```

In [14]:
```
df['anomaly_scores'] = model_IF.decision_function(df[anomaly_inputs])
df['anomaly'] = model_IF.predict(df[anomaly_inputs])
```

In [15]:
```
df.loc[:,['HydrogenBond','Affinity','anomaly_scores','anomaly']]
```

Out[15]:

	HydrogenBond	Affinity	anomaly_scores	anomaly
0	8	-6.9	0.118940	1
1	14	-7.9	0.139712	1
2	14	-6.9	0.127333	1
3	10	-7.0	0.139400	1
4	8	-6.7	0.109279	1
...	...	...	...	...
7235	5	-6.8	0.033581	1
7236	8	-7.3	0.126487	1
7237	16	-8.5	0.098697	1
7238	20	-8.6	0.069946	1
7239	6	-7.3	0.070374	1

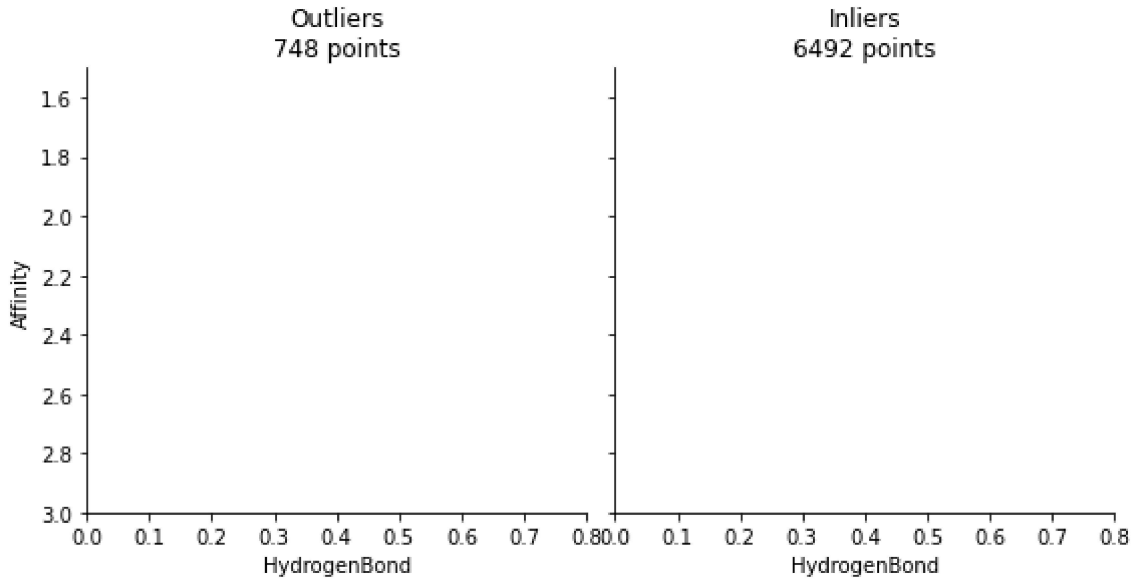
7240 rows × 4 columns

In [39]:
```
def outlier_plot(data, outlier_method_name, x_var, y_var, xaxis_limits=[0, 1],
    print(f'Outlier Method : {outlier_method_name}')
    method = f'{outlier_method_name}_anomaly'

    print(f"Number of anomalous values {len(data[data['anomaly'] == -1])}")
    print(f"Number of non-anomalous values {len(data[data['anomaly'] == 1])}")
    print(f"Total Number of Values: {len(data)}")

    g = sns.FacetGrid(data, col='anomaly', height=4, hue='anomaly', hue_order=
    g.map(sns.scatterplot, x_var, y_var)
#     g.fig.suptitle(f'Outlier Method: {outlier_method_name}', y=1.10, fontwei
#     g.set_axis_labels(x_var, y_var)  # Adding axis labels
    g.set(xlim=xaxis_limits, ylim=yaxis_limits)
    axes = g.axes.flatten()
    axes[0].set_title(f"Outliers\n{len(data[data['anomaly'] == -1])} points")
    axes[1].set_title(f"Inliers\n{len(data[data['anomaly'] == 1])} points")
    return g
```

In [40]:
```
outlier_plot(df,"Isolation Forest","HydrogenBond","Affinity",[0,0.8],[3,1.5])
```

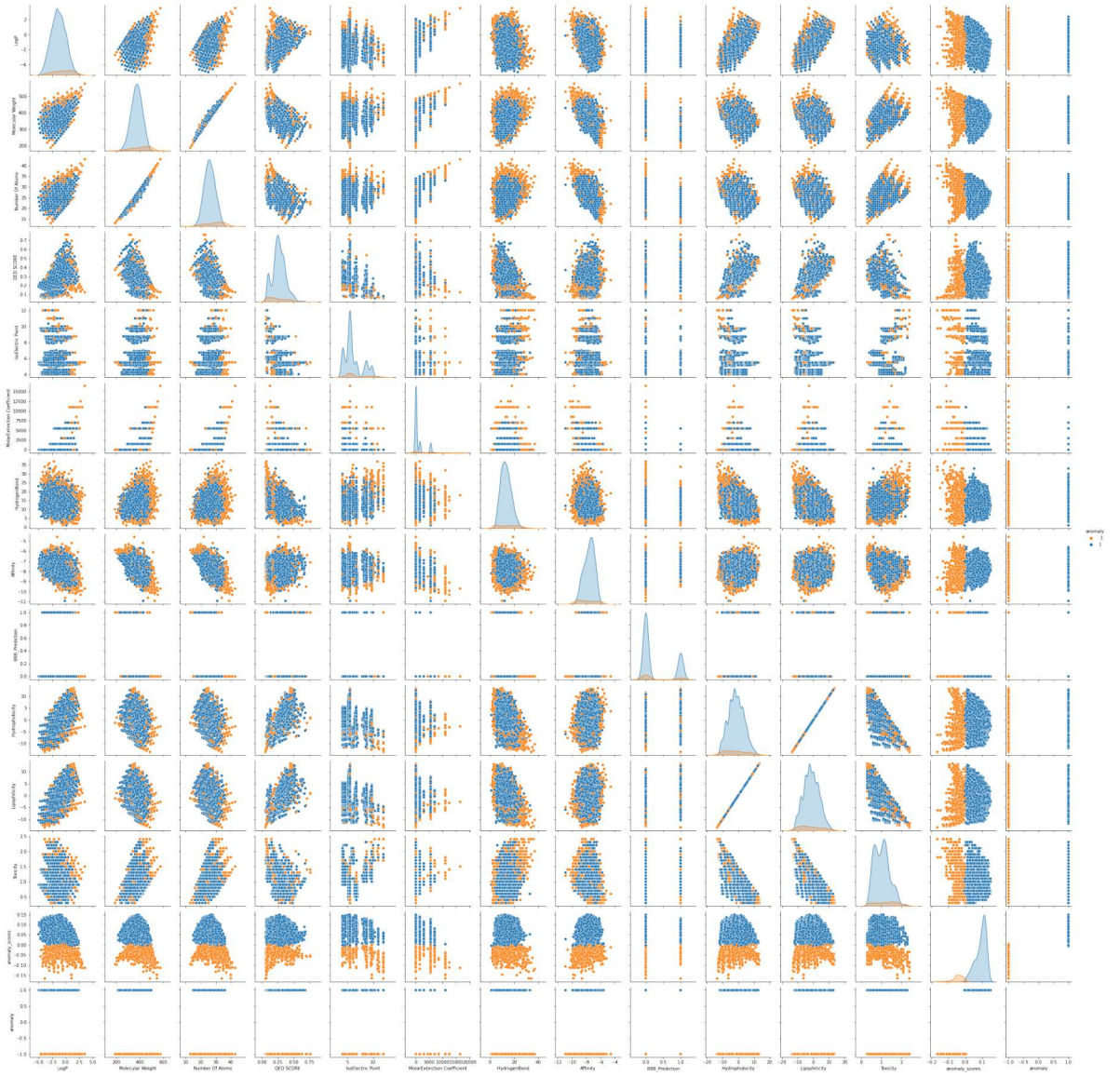
```
Outlier Method : Isolation Forest
Number of anomalous values 748
Number of non-anomalous values 6492
Total Number of Values: 7240
```

Out[40]:
```
<seaborn.axisgrid.FacetGrid at 0x1ec851ec400>
```



In [42]:
```
palette = ['#ff7f0e','#1f77b4']
sns.pairplot(df,vars=anomaly_inputs,hue='anomaly',palette = palette)
```

Out[42]:
```
<seaborn.axisgrid.PairGrid at 0x1ec8511a640>
```



In [ ]:
```
# Building an Isolation Forest Model Using Multiple Features
```

In [28]:
```
df.columns
```

Out[28]:
```
Index(['Peptide', 'LogP', 'Molecular Weight', 'Number Of Atoms', 'QED SCOR
E',
       'IsoElectric Point', 'MolarExtinction Coefficient', 'HydrogenBond',
       'Affinity', 'BBB_Prediction', 'Hydrophobicity', 'Lipophilicity',
       'Toxicity', 'anomaly_scores', 'anomaly'],
      dtype='object')
```

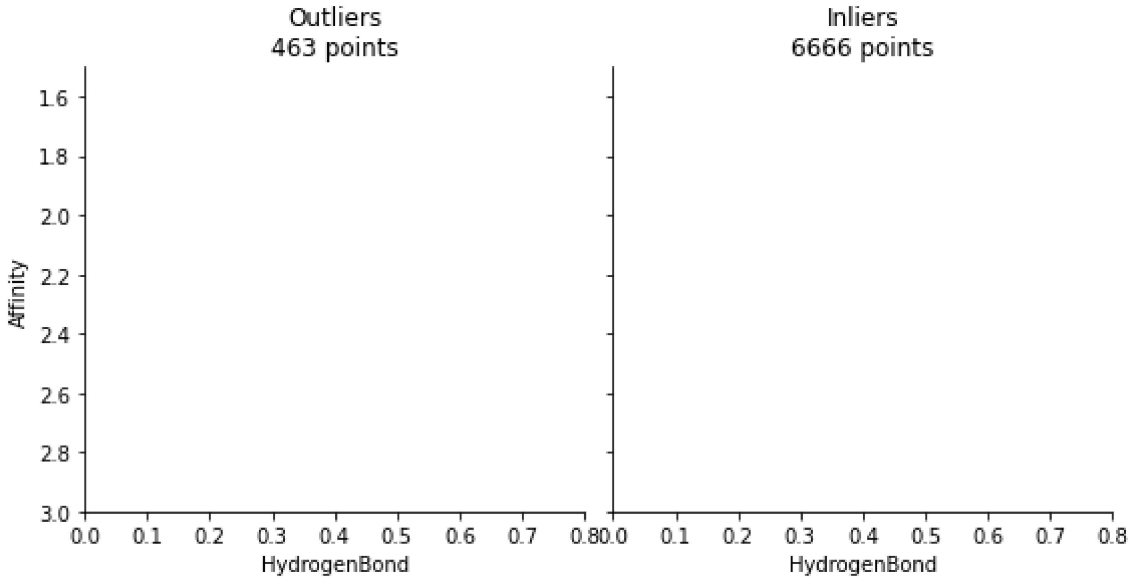
In [31]:
```
anomaly_inputs = [ 'LogP', 'Molecular Weight', 'Number Of Atoms', 'QED SCORE',
       'IsoElectric Point', 'MolarExtinction Coefficient', 'HydrogenBond',
       'Affinity', 'BBB_Prediction', 'Hydrophobicity', 'Lipophilicity',
       'Toxicity', 'anomaly_scores', 'anomaly']
```

In [86]:
```
model_IF = IsolationForest(contamination=0.1,random_state = 42)
model_IF.fit(df[anomaly_inputs])
df['anomaly_scores'] = model_IF.decision_function(df[anomaly_inputs])
df['anomaly'] = model_IF.predict(df[anomaly_inputs])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
X does not have valid feature names, but IsolationForest was fitted with fea
ture names
  warnings.warn(
```

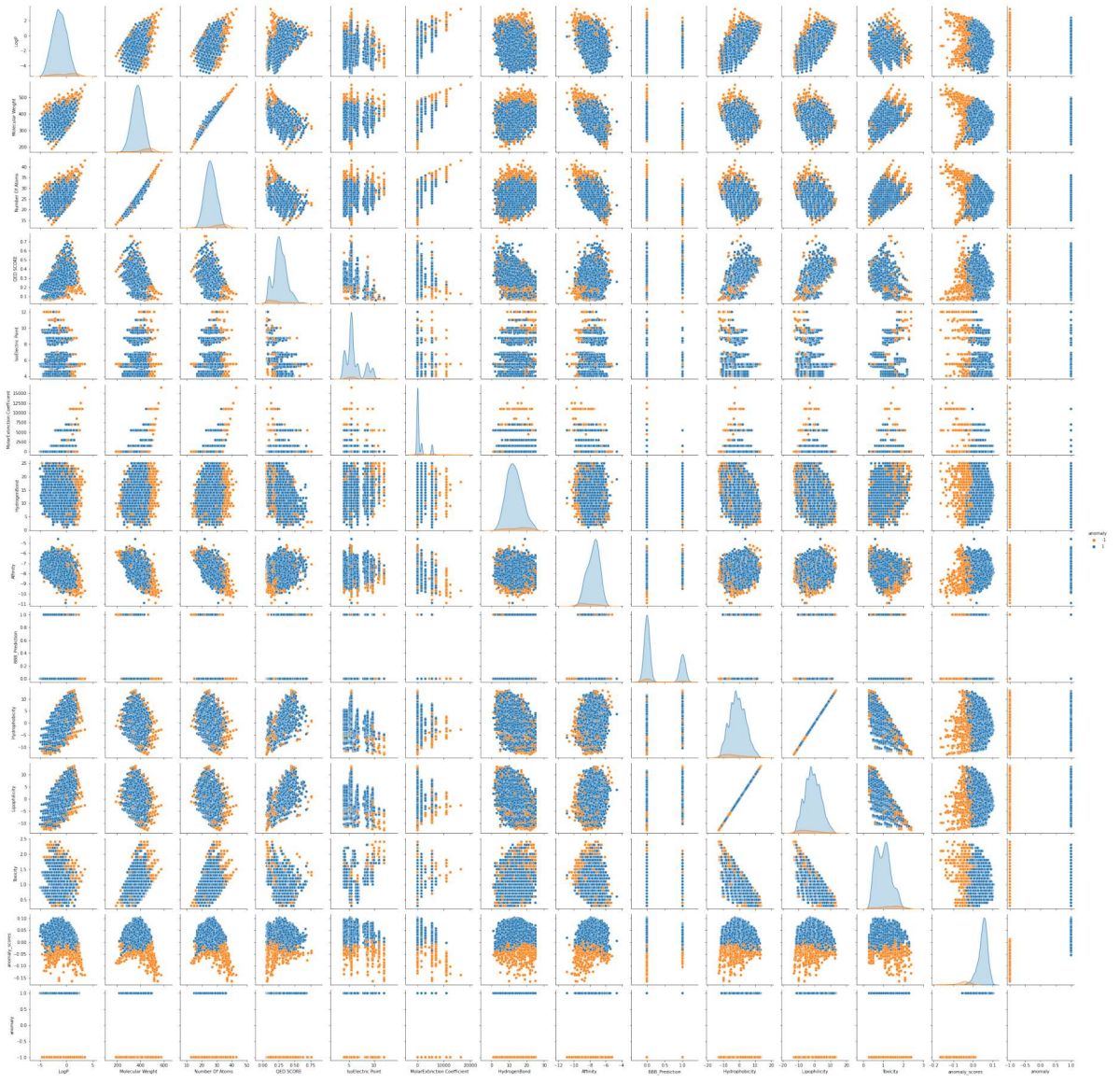
In [87]:
```
outlier_plot(df,'isolation Forest','HydrogenBond','Affinity'
             ,[0,0.8],[3,1.5]);
```

```
Outlier Method : isolation Forest
Number of anomalous values 463
Number of non-anomalous values 6666
Total Number of Values: 7129
```



In [88]:
```
palette = ['#ff7f0e','#1f77b4']
sns.pairplot(df,vars=anomaly_inputs,hue='anomaly',palette = palette)
```

Out[88]: <seaborn.axisgrid.PairGrid at 0x1eca9257af0>



In [62]:
```
# data = df.drop(df[df['anomaly']] == -1)
data = df[df['anomaly']==1]
data.shape
data
```

Out[62]:

ecular Weight	Number Of Atoms	QED SCORE	IsoElectric Point	MolarExtinction Coefficient	HydrogenBond	Affinity	BBB_Prediction	H
88.304	20	0.335919	5.570017	0	11	-7.0	1	
48.429	23	0.093413	8.294712	0	13	-6.9	0	
74.398	26	0.102647	6.262713	0	14	-8.1	0	
58.443	25	0.135830	9.795020	0	16	-7.6	0	
76.483	25	0.120622	9.795020	0	14	-7.4	0	
...	...	...	...	...	...	...	...	
67.402	26	0.324784	5.494364	1490	16	-7.6	0	
81.429	27	0.331093	5.494364	1490	15	-7.5	0	
30.385	23	0.346990	5.494989	0	10	-7.3	0	
45.396	24	0.358185	4.598695	0	9	-7.5	1	
44.412	24	0.338917	5.494989	0	9	-7.2	0	

In [63]:
```
data.columns
```

Out[63]:
```
Index(['Peptide', 'LogP', 'Molecular Weight', 'Number Of Atoms', 'QED SCOR
E',
       'IsoElectric Point', 'MolarExtinction Coefficient', 'HydrogenBond',
       'Affinity', 'BBB_Prediction', 'Hydrophobicity', 'Lipophilicity',
       'Toxicity', 'anomaly_scores', 'anomaly'],
      dtype='object')
```

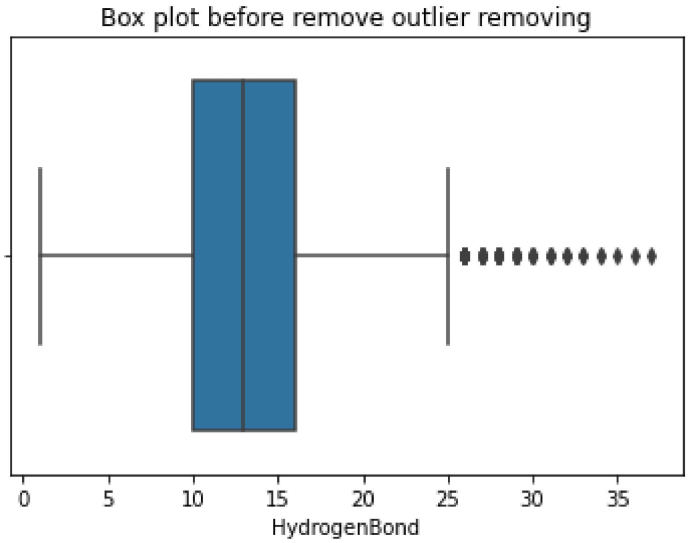
In [ ]:
```
import
```

In [ ]:
```

```

In [47]:
```
import numpy as np
```

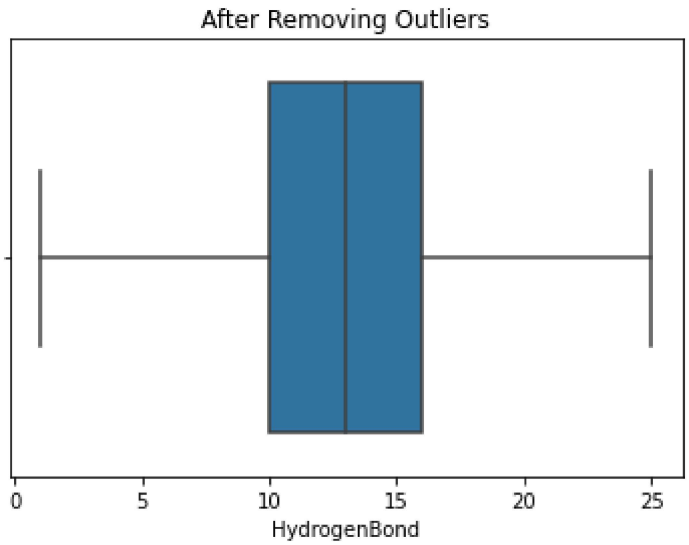
In [48]:
```
from matplotlib import pyplot as plt
sns.boxplot(df['HydrogenBond'])
plt.title("Box plot before remove outlier removing")
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [49]:
```
def drop_outliers(df, field_name):
    q1 = np.percentile(df[field_name], 25)
    q3 = np.percentile(df[field_name], 75)
    iqr = 1.5 * (q3 - q1)
    df.drop(df[(df[field_name] > (q3 + iqr)) | (df[field_name] < (q1 - iqr))].

# Assuming 'train' is your DataFrame
drop_outliers(df, 'HydrogenBond')

# Plotting the boxplot after removing outliers
sns.boxplot(df['HydrogenBond'])
plt.title("After Removing Outliers")
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [64]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
```

In [95]:
```
# Assuming 'df' is your DataFrame containing the dataset
X = df.drop(['Peptide', 'anomaly'], axis=1)
y = df['anomaly']  # Target variable 'anomaly'

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc
```

In [96]:
```
# Create a Logistic Regression model
model = LogisticRegression(random_state=42)

# Train the model
model.fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[96]:
```
LogisticRegression(random_state=42)
```

In [97]:
```
# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
print("Accuracy:", accuracy)
```

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

          -1       0.85      0.27      0.41       103
           1       0.95      1.00      0.97      1323

    accuracy                           0.94      1426
   macro avg       0.90      0.63      0.69      1426
weighted avg       0.94      0.94      0.93      1426

Accuracy: 0.9438990182328191
```

In [68]:
```
# import pandas as pd
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LogisticRegression
# from sklearn.metrics import classification_report, accuracy_score

# # Assuming 'df' is your DataFrame containing the dataset
# X = df.drop(['Peptide', 'anomaly'], axis=1)  # Features (excluding 'Peptide'
# y = df['anomaly']  # Target variable 'anomaly'

# # Split the dataset into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# # Create a Logistic Regression model
# model = LogisticRegression(random_state=42)

# # Train the model
# model.fit(X_train, y_train)

# # Make predictions on the test data
# y_pred = model.predict(X_test)

# # Evaluate the model
# accuracy = accuracy_score(y_test, y_pred)
# report = classification_report(y_test, y_pred)
# print("Classification Report:\n", report)
# print("Accuracy:", accuracy)
```

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

          -1       0.72      0.82      0.77       880
           1       0.63      0.49      0.55       546

    accuracy                           0.70      1426
   macro avg       0.68      0.66      0.66      1426
weighted avg       0.69      0.70      0.69      1426

Accuracy: 0.6956521739130435

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

In [ ]:

In [ ]:
```
## next data prediction of the dataset
```

In [102]:
```
future_predictions = model.predict(X_train)
```

In [103]:
```
# Assuming 'future_predictions' contains the predictions for the future data
print("Future Predictions:")
print(future_predictions)
```

```
Future Predictions:
[1 1 1 ... 1 1 1]
```

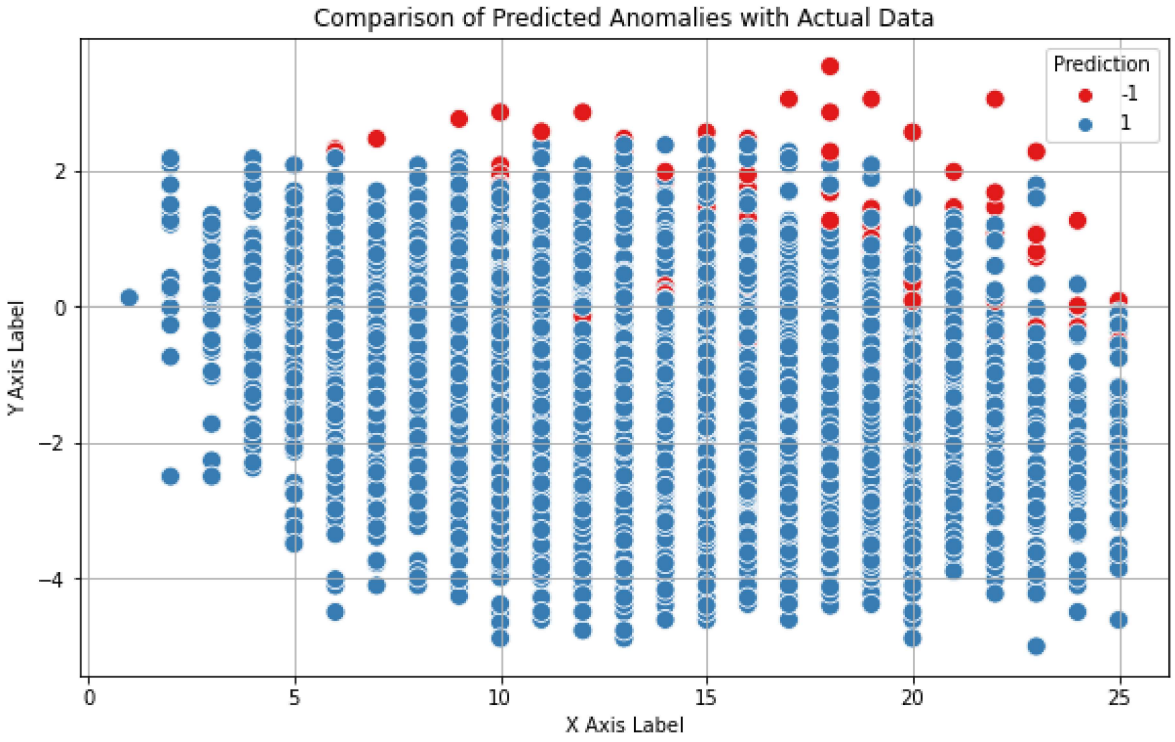
In [104]:
```
# Get probability predictions (confidence scores)
probability_predictions = model.predict_proba(X_train)
```

In [105]:
```
probability_predictions
```

Out[105]:
```
array([[0.14207774, 0.85792226],
       [0.07173814, 0.92826186],
       [0.12765756, 0.87234244],
       ...,
       [0.00783189, 0.99216811],
       [0.04688666, 0.95311334],
       [0.01169727, 0.98830273]])
```

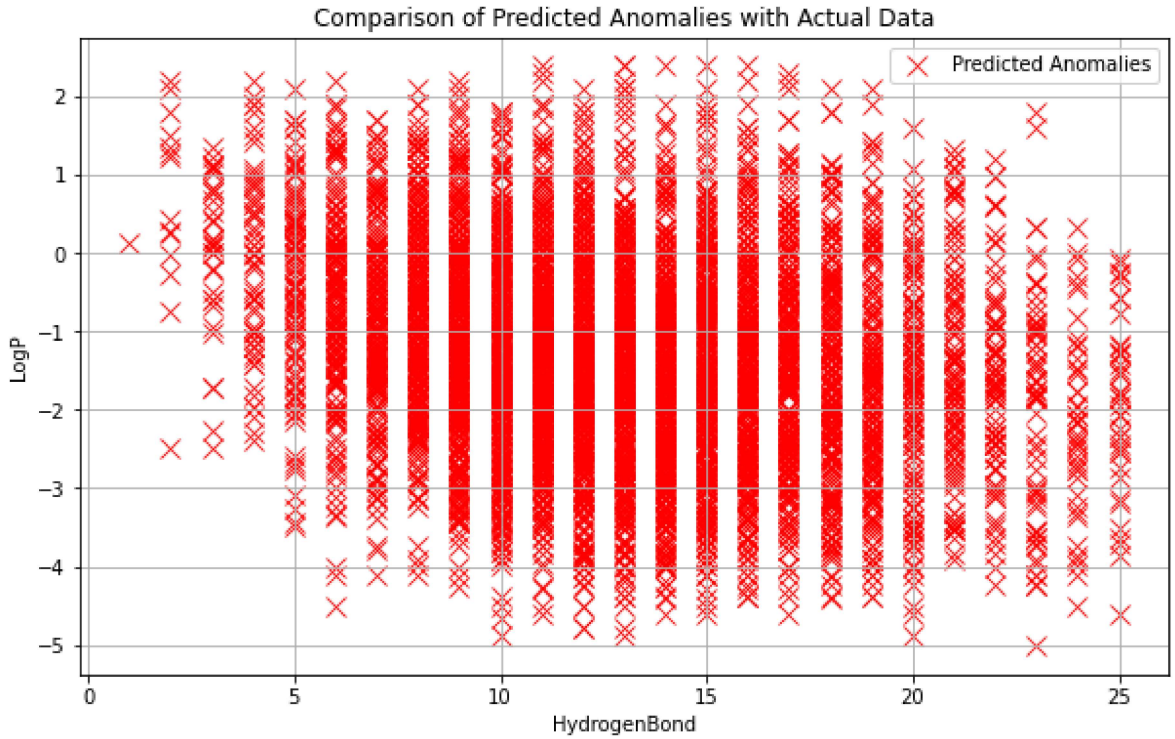
In [109]:

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


X_train['prediction'] = future_predictions

# Plotting the comparison
plt.figure(figsize=(10, 6))
sns.scatterplot(data=X_train, x='HydrogenBond', y='LogP', hue='prediction', pa
plt.title('Comparison of Predicted Anomalies with Actual Data')
plt.xlabel('X Axis Label')
plt.ylabel('Y Axis Label')
plt.legend(title='Prediction')
plt.grid(True)
plt.show()
```



In [110]:

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



# Add the 'prediction' column to the actual data DataFrame (X_train)
X_train['prediction'] = future_predictions

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


sns.scatterplot(data=X_train[X_train['prediction'] == 0], x='HydrogenBond', y=

sns.scatterplot(data=X_train[X_train['prediction'] == 1], x='HydrogenBond', y=

plt.title('Comparison of Predicted Anomalies with Actual Data')
plt.xlabel('HydrogenBond')
plt.ylabel('LogP')
plt.legend()
plt.grid(True)
plt.show()
```



In [113]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

data['PeptideEncoder'] = label_encoder.fit(data['Peptide'])
print(data['PeptideEncoder'])
data.columns
```

```
6       LabelEncoder()
24      LabelEncoder()
25      LabelEncoder()
29      LabelEncoder()
32      LabelEncoder()
             ...
7214    LabelEncoder()
7216    LabelEncoder()
7222    LabelEncoder()
7225    LabelEncoder()
7226    LabelEncoder()
Name: PeptideEncoder, Length: 2735, dtype: object

C:\Users\hp\AppData\Local\Temp\ipykernel_28436\1513803019.py:4: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  data['PeptideEncoder'] = label_encoder.fit(data['Peptide'])
```

Out[113]:
```
Index(['Peptide', 'LogP', 'Molecular Weight', 'Number Of Atoms', 'QED SCOR
E',
       'IsoElectric Point', 'MolarExtinction Coefficient', 'HydrogenBond',
       'Affinity', 'BBB_Prediction', 'Hydrophobicity', 'Lipophilicity',
       'Toxicity', 'anomaly_scores', 'anomaly', 'PeptideEncoder'],
      dtype='object')
```

In [116]:
```
# Assuming you have a trained model named 'model' and new data DataFrame named

# Predict anomalies for new data
peptide = float(data['PeptideEncoder'])
new_data['prediction'] = model.predict(data[['HydrogenBond', 'peptide']])

# Plotting the comparison with different markers for actual and predicted data
plt.figure(figsize=(10, 6))

# Plotting actual data with 'o' markers
sns.scatterplot(data=X_train[X_train['prediction'] == 0], x='HydrogenBond', y=

# Plotting predicted data with 'x' markers
sns.scatterplot(data=X_train[X_train['prediction'] == 1], x='HydrogenBond', y=

plt.title('Comparison of Predicted Anomalies with Actual Data')
plt.xlabel('HydrogenBond')
plt.ylabel('LogP')
plt.legend()
plt.grid(True)
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [116], in <cell line: 4>()
      1 # Assuming you have a trained model named 'model' and new data DataF
rame named 'new_data' containing hydrogen bond data and protein names
      2
      3 # Predict anomalies for new data
----> 4 peptide = float(data['PeptideEncoder'])
      5 new_data['prediction'] = model.predict(data[['HydrogenBond', 'peptid
e']])
      7 # Plotting the comparison with different markers for actual and pred
icted data

File C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py:191, i
n _coerce_method.<locals>.wrapper(self)
    189 if len(self) == 1:
    190     return converter(self.iloc[0])
--> 191 raise TypeError(f"cannot convert the series to {converter}")

TypeError: cannot convert the series to <class 'float'>
```

In [ ]: