Bullying Prediction Data Analysis (Code Shown)

Author: Jeremy Klassen

Introduction

As members of the global community, it is vital that we come together with genuine passion to understand the issue of bullying. Learning more about bullying is not just a matter of necessity; it's our shared responsibility to ensure the well-being of our youth. By gaining deeper insights into this complex problem, we can empower young individuals, strengthen intervention strategies, and create communities that embraces empathy and respect.

To this end, I wrote this jupyter notebook in order to explore a dataset found on Kaggle (link in acknowledgements). Leonardo Martinelli selected questions and data from a 2018 Questionairre that 56,000 Argentinian students responded to. This Questionairre is part of the World Health Organization's Global School-based Student Health Survey (GSHS). The GSHS is designed to assess various mortality and morbitity factors in 13 - 17 year old students around the world.

There are two goals for this project:

```
- Create some machine learning models that can predict if respondents are
being a) cyber bullied b) bullied on school c) bullied off school.
- Identify insights that can help us better understand bullying
```

Sectional overview

[Cleaning the Data] is an important step in the analysis process. This is where the dataset is taken from it's initial state, and transformed into a new state that is usable for the needs of the analyst.
[Statistical Assessments] does a bit of statistical analysis on the dataset. Understanding how each feature in the set relates to others helps in deciding how to better clean the data, and how to implement machine learning models.
[Machine Learning Models] is the section where the machine learning models are developed. Four different models are created for each independent variable in this project (Cyber Bullying, In School Bullying, and Off School Bullying).

[Conclusions] Four major insights were discovered in this project. The details can be found in this section.

[Back to Top]

In [1]:
```
# Public and openly available libraries of code for python developers
# Pandas is the major data manipulation library. Picture Excel but but more powerful
import pandas as pd
# Matplotlib is one of the major libraries for creating charts and graphs
import matplotlib.pyplot as plt
# Seaborn is another chart and graph library. Seaborn uses matplotlib to make cleaner lo
import seaborn as sns
# Numpy is a library that helps make manipulating variables and numbers faster
import numpy as np
# Scipy is a statistical library. In this case we will use it for chi square tests
from scipy.stats import chi2_contingency
```

In [2]:
```
# importing csv file. Saving it to df
# replacing all the empty spaces with 'nan' empty values.
# removing record column
df = pd.read_csv("Bullying_2018.csv", sep=';')
df.replace(r'^\s*$', np.nan, regex=True, inplace=True)
df = df.drop('record', axis=1)
```

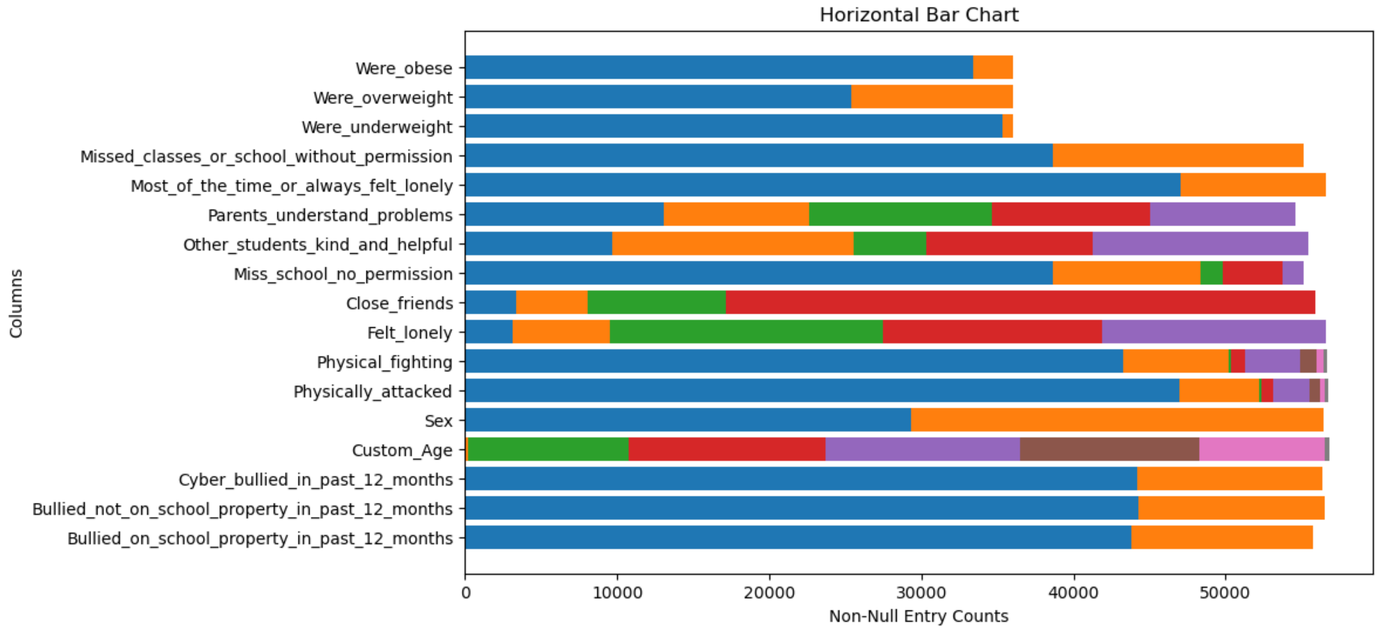
[Back to Top]

Data Cleaning

Lets explore the value counts of each category. If a category is lopsided in it's value counts, then it can affect the training of algorithms.

In [3]:
```
def value_count_graph(df):
    # Calculate non-null entry counts for each column
    non_null_counts = df.notnull().sum()

    # Calculate value counts for each column
    value_counts = df.apply(pd.Series.value_counts)

    # Plot horizontal bar chart with split colors
    plt.figure(figsize=(10, 6))

    # Iterate over each column
    for i, col in enumerate(non_null_counts.index):
        # Get the distinct values and their counts for the column
        values = value_counts[col].dropna()

        # Set the colors for the distinct values
        colors = plt.cm.tab10(range(len(values)))

        # Plot each distinct value as a separate bar segment
        left = 0
        for j, (value, count) in enumerate(values.items()):
            plt.barh(i, count, left=left, color=colors[j])
            left += count

    # Set y-axis labels as column names
    plt.yticks(range(len(non_null_counts)), non_null_counts.index)

    plt.title('Horizontal Bar Chart')
    plt.xlabel('Non-Null Entry Counts')
```

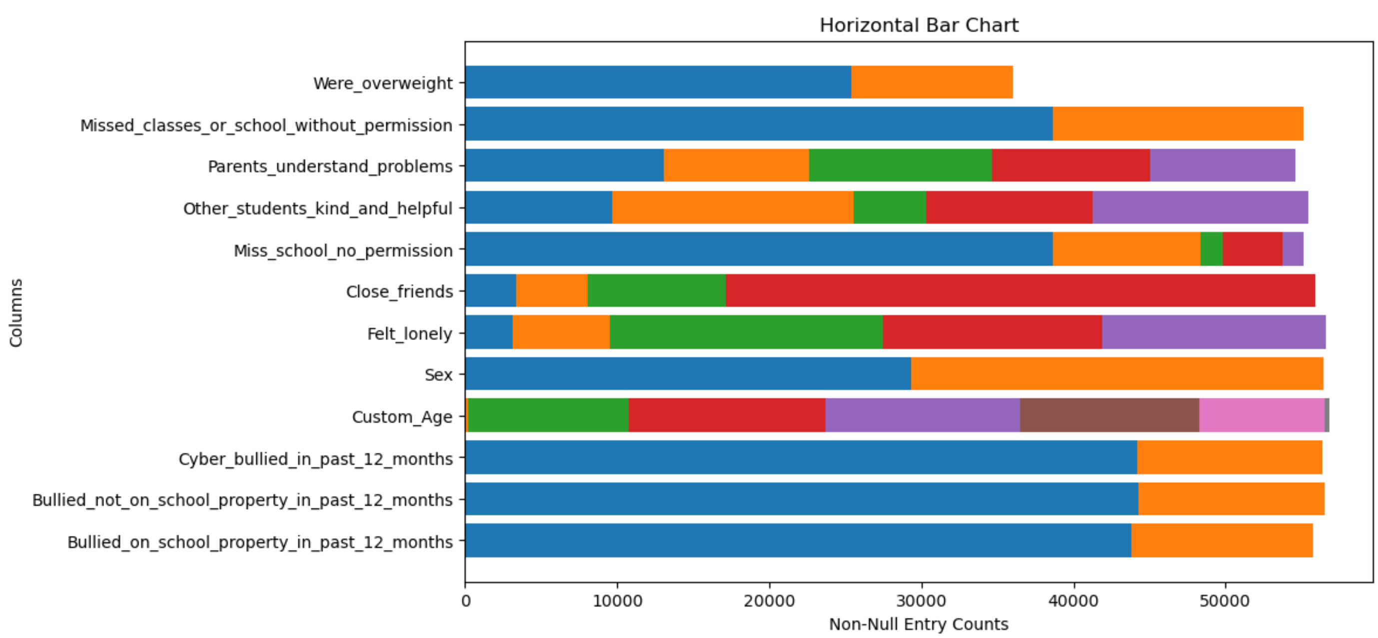
```
    plt.ylabel('Columns')

    # plt.tight_layout()
    plt.show()

value_count_graph(df)
```



Dropping Categories.

We need to keep the 3 categories of bullying since those will be our dependent variables. Were_obese, Were_underweight, Most_of_the_time_or_always_felt_lonely, Physical_fighting, and Physically_attacked will all be removed though due to their lopsided value counts.

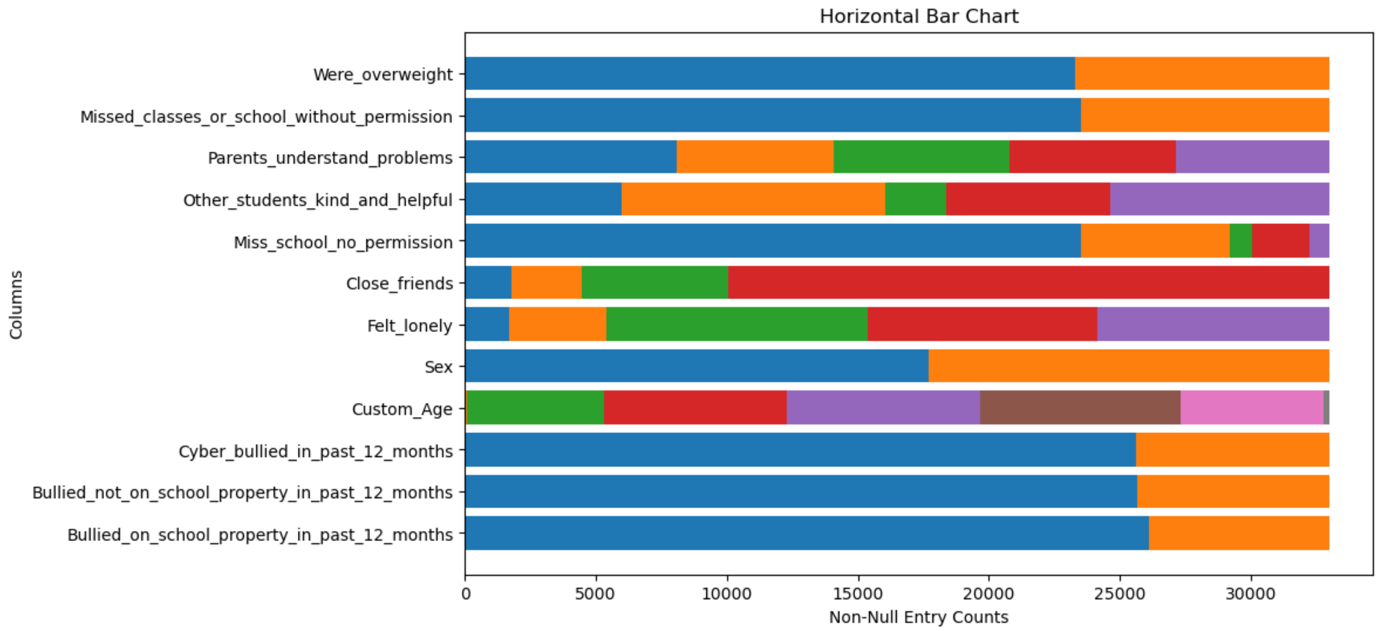
In [4]:
```
df = df.drop([
    'Were_obese',
    'Were_underweight',
    'Most_of_the_time_or_always_felt_lonely',
    'Physical_fighting',
    'Physically_attacked'], axis=1)

value_count_graph(df)
```



In [5]:
```
print(f"Number of rows before dropping NaN's: {len(df)}")
df.dropna(inplace=True)
print(f"Number of rows after dropping NaN's: {len(df)}")
```

```
Number of rows before dropping NaN's: 56981
Number of rows after dropping NaN's: 33031
```

In [6]:
```
value_count_graph(df)
```



Comparing this distribution to the previous distribution they appear to be fairly similar. With 33,000 rows there is still substantial amounts of data. There is no apparent need at this point to impute missing data at this point.

[Back to Top]

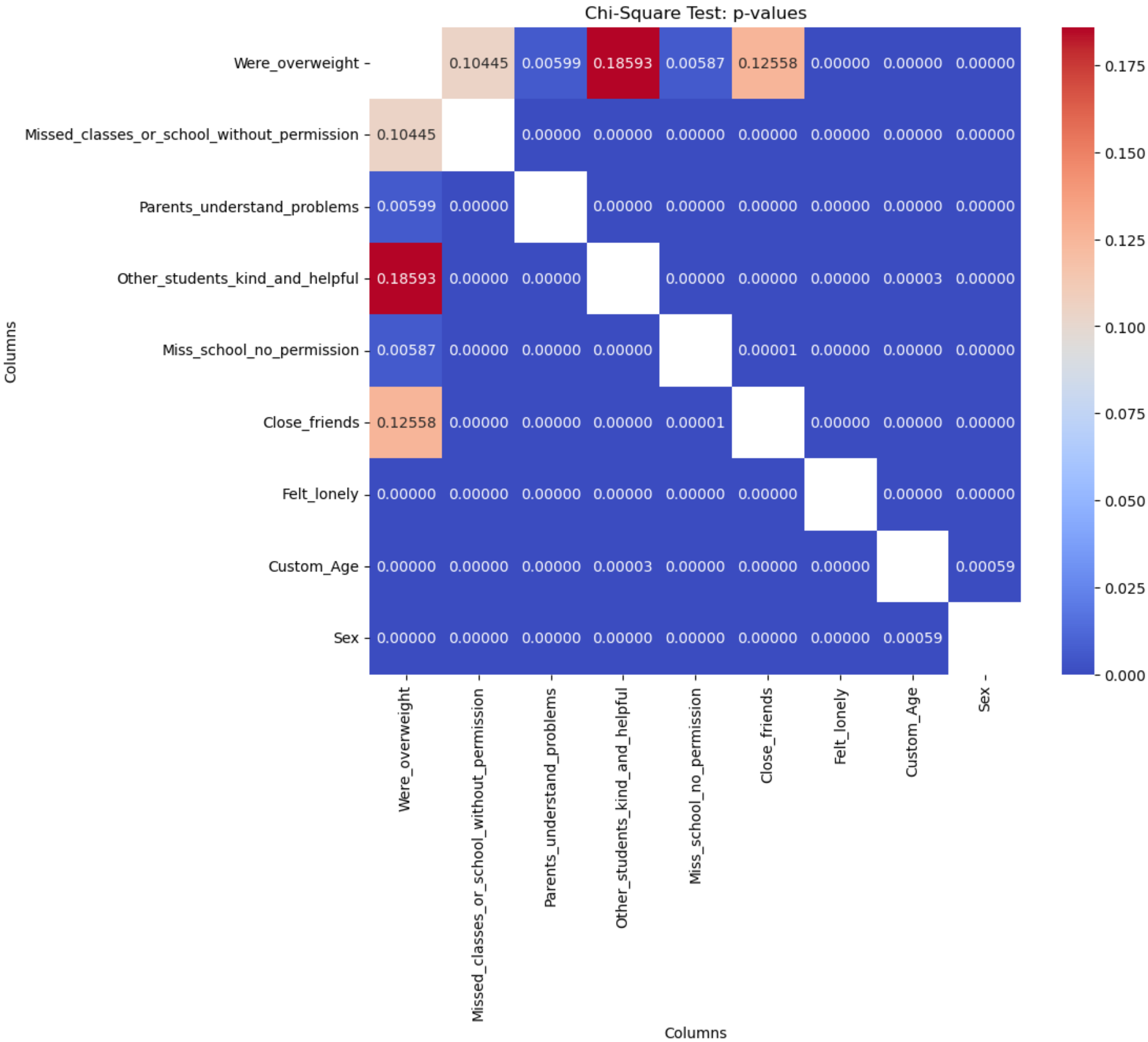
Statistical analysis

If there are categories that are highly correlated with each other, we might not need to include both of them in machine learning algorithms.

All of the columns (features) in this dataset use nominal level data. To find significant correlations we will run Chi Square tests between the features# Public and openly available libraries of code for python developers

In [7]:
```
columns_to_test = [
    'Were_overweight',
    'Missed_classes_or_school_without_permission',
    'Parents_understand_problems',
    'Other_students_kind_and_helpful',
    'Miss_school_no_permission',
    'Close_friends',
    'Felt_lonely',
    'Custom_Age',
    'Sex'
]

# Create an empty DataFrame to store the p-values
p_values_df = pd.DataFrame(columns=columns_to_test, index=columns_to_test)
```

```python
# Iterate over each pair of columns and perform the chi-square test
for i in range(len(columns_to_test)):
    for j in range(i+1, len(columns_to_test)):
        column1 = columns_to_test[i]
        column2 = columns_to_test[j]

        # Create a contingency table for the chi-square test
        contingency_table = pd.crosstab(df[column1], df[column2])

        # Perform the chi-square test
        _, p_value, _, _ = chi2_contingency(contingency_table)

        # Store the p-value in the DataFrame
        p_values_df.loc[column1, column2] = p_value
        p_values_df.loc[column2, column1] = p_value

# Convert the p-values to numeric data type
p_values_df = p_values_df.astype(float)

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(p_values_df, annot=True, cmap='coolwarm', fmt=".5f")

# Add plot labels and title
plt.xlabel('Columns')
plt.ylabel('Columns')
plt.title('Chi-Square Test: p-values')

# Show the plot
plt.show()
```



In [8]:

```
# Assuming your DataFrame is named 'df'
contingency_table = pd.crosstab(df['Felt_lonely'], df['Cyber_bullied_in_past_12_months']

# Perform chi-square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Print the results
print("Tests for Felt_lonely across bullying categories")
print("Cyber Bullying")
print("Chi-square statistic:", chi2)
print("p-value:", p_value)
print("Degrees of freedom:", dof)

# Assuming your DataFrame is named 'df'
contingency_table = pd.crosstab(df['Felt_lonely'], df['Bullied_not_on_school_property_in

# Perform chi-square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Print the results
print()
print("Off School Bullying")
print("Chi-square statistic:", chi2)
print("p-value:", p_value)
print("Degrees of freedom:", dof)
```

```
# Assuming your DataFrame is named 'df'
contingency_table = pd.crosstab(df['Felt_lonely'], df['Bullied_on_school_property_in_pas

# Perform chi-square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Print the results
print()
print("On School Bullying")
print("Chi-square statistic:", chi2)
print("p-value:", p_value)
print("Degrees of freedom:", dof)
```

```
Tests for Felt_lonely across bullying categories
Cyber Bullying
Chi-square statistic: 1827.3355765089782
p-value: 0.0
Degrees of freedom: 4

Off School Bullying
Chi-square statistic: 1537.5274120393294
p-value: 0.0
Degrees of freedom: 4

On School Bullying
Chi-square statistic: 1469.733970977208
p-value: 0.0
Degrees of freedom: 4
```

Chi Square Test Results

Using the Social Sciences standard of a 95% confidence interval, we need a P-value of smaller than 0.05 to for the relationship between the two to be considered statistically significant.

In these results the only tests that didn't result in statistical significance were:

```
-Were_overweight vs Close_friends [X2(3) = 5.7287, p>0.05]
-Were_overweight vs Missed_classes_or_school_without_permission
    [X2(1) = 2.6362, p>0.05]
-Were_overweight vs Other_students_kind_and_helpful [X2(4) = 6.1825,
p>0.05]
```

Were_overweight is the only category with low or non-correlations. It might have to do with how few respondants answered this question. So much correlation in this dataset is unexpected. It is not reasonable at this time to drop any of the categories.

[Back to Top]

Machine Learning Models

Here we will us Machine Learning (AI) to predict if a respondent was a) cyber bullied, b) bullied at school, or c) bullied off school.

To do this four different ML models will be used: Logistic Regression, Random Forest, MLPClassifier (a type of neural network), and XGBoost. Accuracy of predictions will be listed for each model while some will get an AUC score as well. For the MLPClassifiers and XGBoost models bar graphs showing how important each feature is to the model will be displayed.

In [9]:
```
# sklearn (or Scikit-Learn) is the 'tool belt' of machine learning (ML) algorithms. Ther
# many other ML libraries, but none have such a diverse range of options to use.
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier

# imbalanced learn has tools for safely adding and removing data from imbalanced dataset
# This sometimes needs to be done if the dataset doesn't accurately represent the popula
# it is a frame for. Also ML Algorithms have a hard time learning if there is a dispropo
# amount of certain data points.
from imblearn.over_sampling import ADASYN
from imblearn.over_sampling import SMOTE

# xgboost is a machine learning model that creates many small decision trees one by one.
# tree deals with a small part of the problem.
import xgboost as xgb
```

[Back to Top]

Bullying Off School Property

Logistic Regression

In [10]:
```
# Here we convert the data from words to coded numbers. ML models need it this way in or
dforiginal = df.copy()
df = df.apply(lambda x: x.astype('category').cat.codes)

df2 = df.copy()
y = df2['Bullied_not_on_school_property_in_past_12_months']
X = df2.drop(['Bullied_not_on_school_property_in_past_12_months', 'Bullied_on_school_pro


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=19

offSchoolLR = LogisticRegression()
offSchoolLR.fit(X_train, y_train)

y_pred = offSchoolLR.predict(X_test)

offSchoolLRAccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(offSchoolLRAccuracy*100, 2)}%")
```

```
Accuracy: 77.31%
```

Random Forest

In [11]:
```
offSchoolRF = RandomForestClassifier()
offSchoolRF.fit(X_train, y_train)
y_pred = offSchoolRF.predict(X_test)

offSchoolRFAccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(offSchoolRFAccuracy*100, 2)}%")
```

```
Accuracy: 73.95%
```

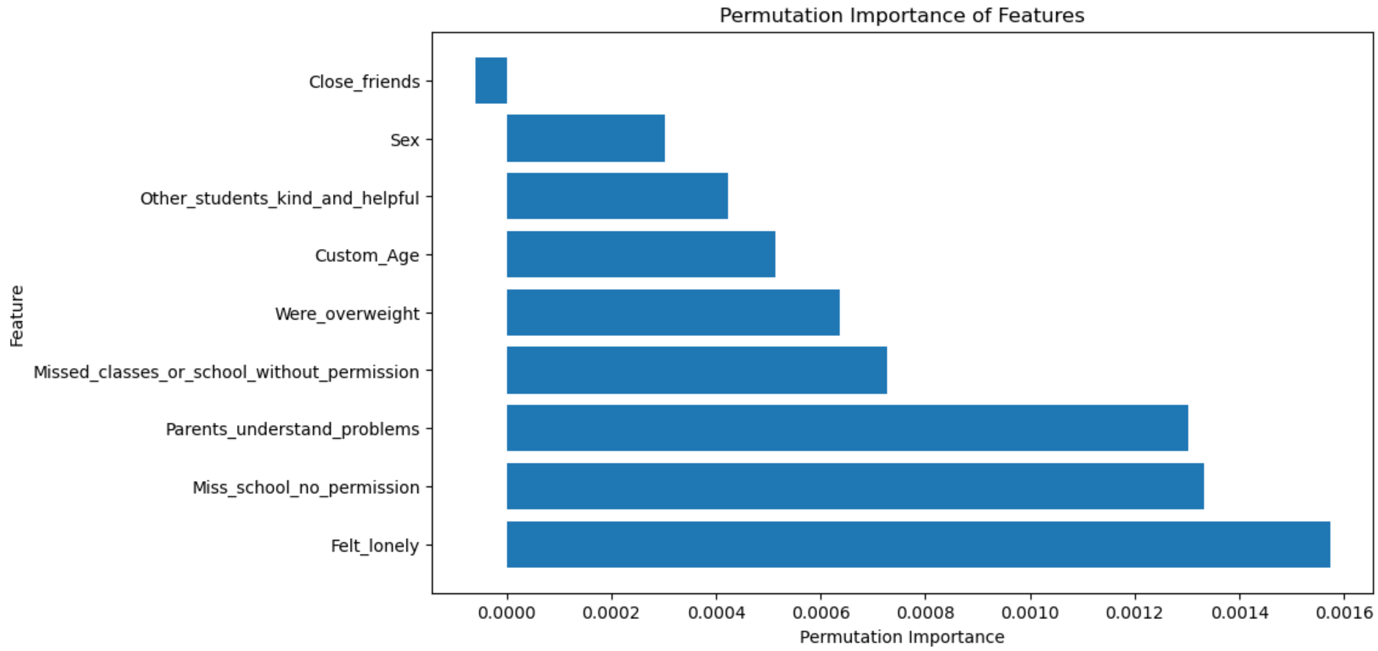
MLPClassifier

MLPClassifier is a Neural Network designed for classification tasks. In this case we are using it to classify a respondent as bullied or not.

In [12]:
```
df2 = df.copy()
y = df2['Bullied_not_on_school_property_in_past_12_months']
X = df2.drop(['Bullied_not_on_school_property_in_past_12_months', 'Bullied_on_school_pro

num_input_features = X.shape[1]
hidden_layer_sizes = (num_input_features, 4, 2)

offSchoolMLP = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes, max_iter=2000, alpha

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=19

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

offSchoolMLPaccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(offSchoolMLPaccuracy*100, 2)}%")

y_pred_prob = offSchoolMLP.predict_proba(X_test)[:, 1]
auc_score = roc_auc_score(y_test, y_pred_prob)
print(f"AUC: {auc_score}")
```

```
Accuracy: 77.57%
AUC: 0.6453930926579301
```

In [13]:
```
from sklearn.inspection import permutation_importance

# Perform permutation importance
result = permutation_importance(offSchoolMLP, X_test, y_test, n_repeats=10, random_state

# Get feature importances and names
importances = result.importances_mean
feature_names = X_test.columns

# Sort feature importances in descending order
sorted_indices = importances.argsort()[::-1]
sorted_importances = importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_importances)), sorted_importances, tick_label=sorted_feature_n
plt.xlabel("Permutation Importance")
plt.ylabel("Feature")
plt.title("Permutation Importance of Features")
plt.show()
```



XGBoost

In [14]:

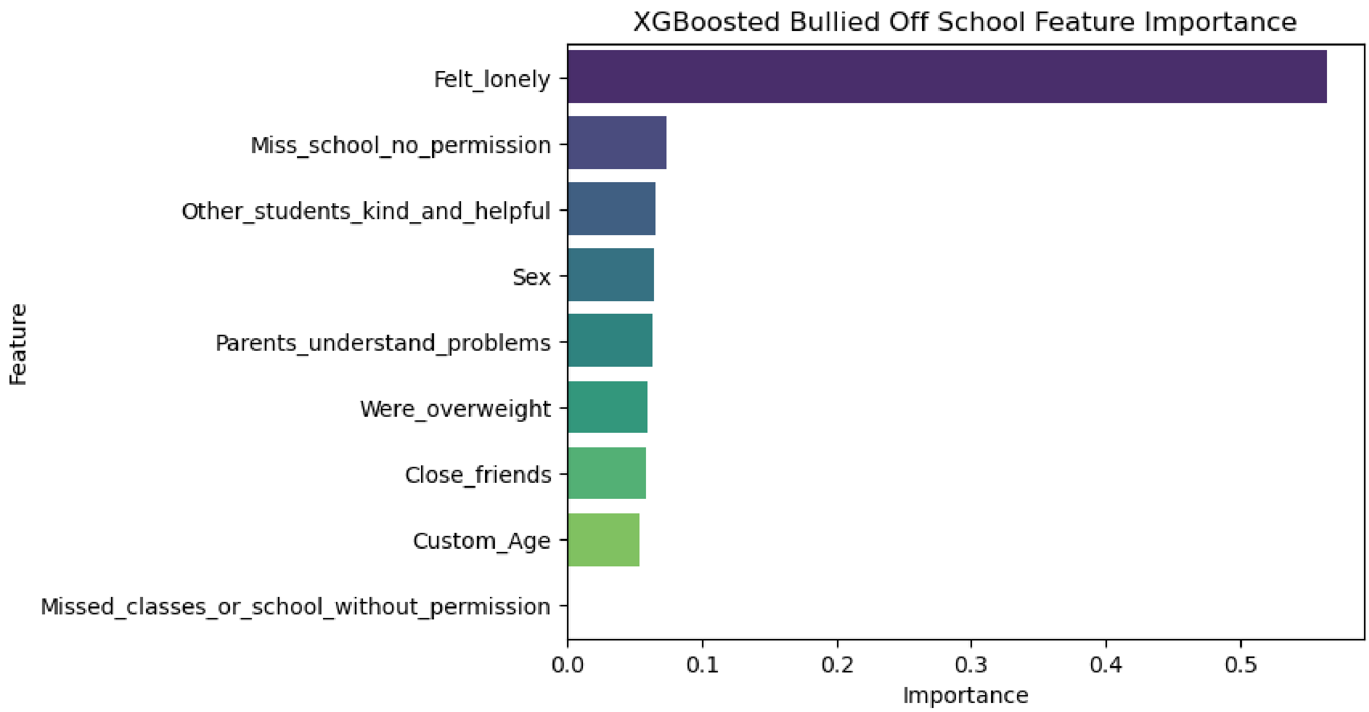
```
df = dforiginal.copy()

df = df.apply(lambda x: x.astype('category').cat.codes)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['Bullied_not_on_school_property_in_past_12_months'])
X = df.drop(['Bullied_not_on_school_property_in_past_12_months', 'Cyber_bullied_in_past_


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

model = xgb.XGBClassifier(n_estimators=3000,reg_lambda=0.01, early_stopping_rounds=50, l

# Train the model
model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose = 2

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(accuracy*100, 2)}%")

features = pd.DataFrame(data=model.feature_importances_, index=model.feature_names_in_)
```

```
[0]	validation_0-logloss:0.67614	validation_1-logloss:0.67594
[25]	validation_0-logloss:0.51810	validation_1-logloss:0.51650
[50]	validation_0-logloss:0.49805	validation_1-logloss:0.49819
[75]	validation_0-logloss:0.49272	validation_1-logloss:0.49581
[100]	validation_0-logloss:0.48897	validation_1-logloss:0.49619
[125]	validation_0-logloss:0.48684	validation_1-logloss:0.49659
[128]	validation_0-logloss:0.48657	validation_1-logloss:0.49665
Accuracy: 78.46%
```

In [15]:

```
# Get the feature importances from the trained XGBoost model
importances = model.feature_importances_

# Create a new DataFrame to hold the features and their importances
features = pd.DataFrame({'Feature': X.columns, 'Importance': importances})

# Sort the DataFrame by the 'Importance' column
features.sort_values('Importance', inplace=True)
```

```
features_r = features[::-1]
sns.barplot(y='Feature', x='Importance', data=features_r, palette='viridis', orient='h')

plt.title("XGBoosted Bullied Off School Feature Importance")
plt.show()
```



[Back to Top]

Bullying On School Property

Logistic regression

In [16]:
```
df = dforiginal.copy()
df = df.apply(lambda x: x.astype('category').cat.codes)

df2 = df.copy()
y = df2['Bullied_on_school_property_in_past_12_months']
X = df2.drop(['Bullied_not_on_school_property_in_past_12_months', 'Bullied_on_school_pro


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=19

offSchoolLR = LogisticRegression()
offSchoolLR.fit(X_train, y_train)

y_pred = offSchoolLR.predict(X_test)

offSchoolLRAccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(offSchoolLRAccuracy*100, 2)}%")
```

```
Accuracy: 79.23%
```

Random Forest

In [17]:
```
onSchoolRF = RandomForestClassifier()
onSchoolRF.fit(X_train, y_train)
y_pred = onSchoolRF.predict(X_test)
```

```
onSchoolRFAccuracy = accuracy_score(y_test, y_pred)
print(f"On School Random Forrest Accuracy: {round(onSchoolRFAccuracy*100,2)}%")
```

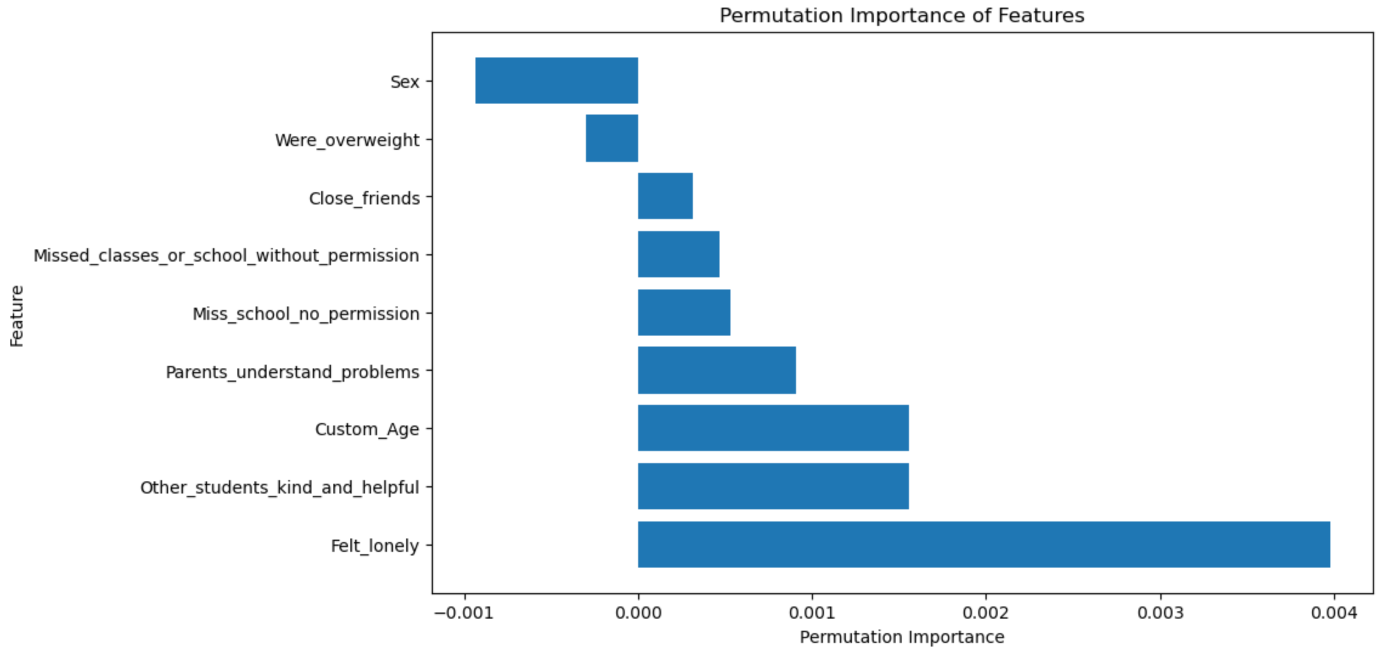
```
On School Random Forrest Accuracy: 76.36%
```

MLPClassifier

In [18]:
```
df2 = df.copy()
y = df2['Bullied_on_school_property_in_past_12_months']
X = df2.drop(['Bullied_not_on_school_property_in_past_12_months', 'Bullied_on_school_pro

num_input_features = X.shape[1]
hidden_layer_sizes = (num_input_features)

onSchoolMLP = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes, max_iter=2000)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=19

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

onSchoolMLPaccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(onSchoolMLPaccuracy*100, 2)}%")

y_pred_prob = onSchoolMLP.predict_proba(X_test)[:, 1]
auc_score = roc_auc_score(y_test, y_pred_prob)
print(f"AUC: {auc_score}")
```

```
Accuracy: 79.2%
AUC: 0.6577262195906809
```

In [19]:
```
# Perform permutation importance
result = permutation_importance(onSchoolMLP, X_test, y_test, n_repeats=10, random_state=

# Get feature importances and names
importances = result.importances_mean
feature_names = X_test.columns

# Sort feature importances in descending order
sorted_indices = importances.argsort()[::-1]
sorted_importances = importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_importances)), sorted_importances, tick_label=sorted_feature_n
plt.xlabel("Permutation Importance")
plt.ylabel("Feature")
plt.title("Permutation Importance of Features")
plt.show()
```



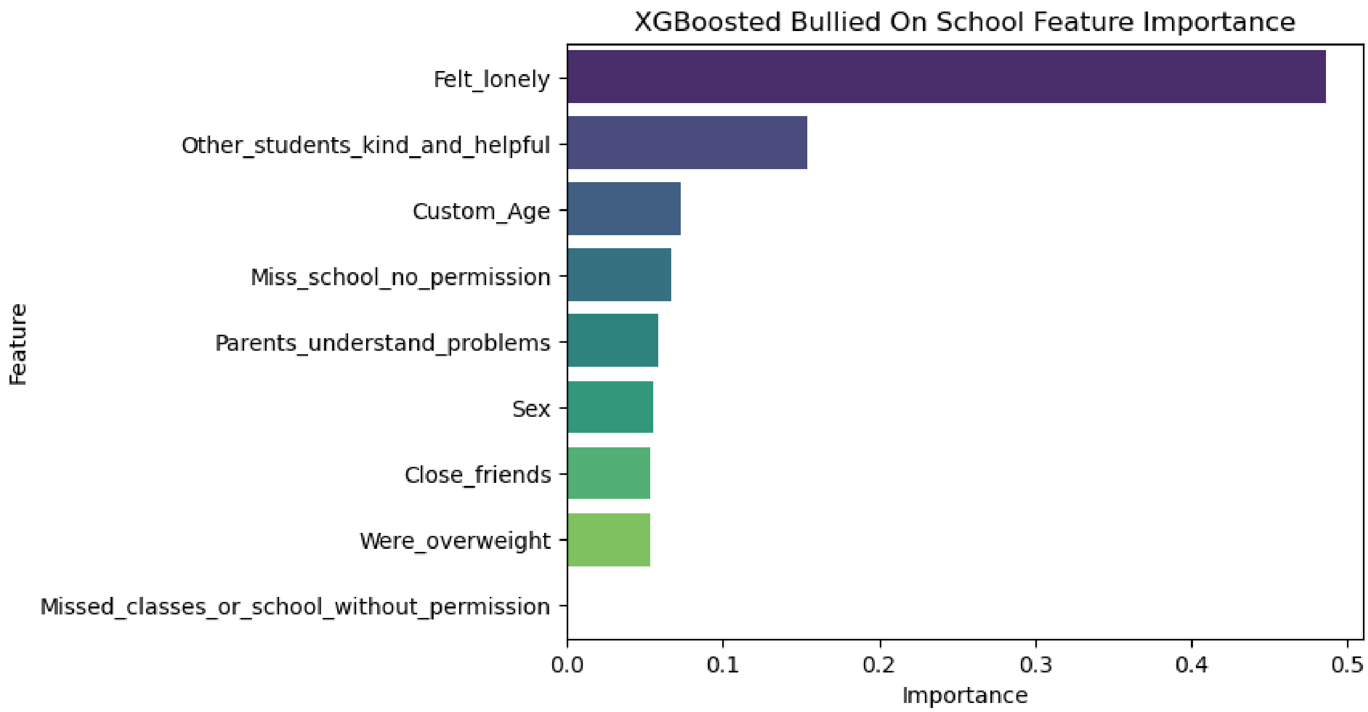
XGBoost

In [20]:
```
df = dforiginal.copy()

df = df.apply(lambda x: x.astype('category').cat.codes)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['Bullied_on_school_property_in_past_12_months'])
X = df.drop(['Bullied_not_on_school_property_in_past_12_months', 'Cyber_bullied_in_past_

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Create an XGBoost classifier
model = xgb.XGBClassifier(n_estimators=1000, early_stopping_rounds=50, learning_rate=0.0

# Train the model
model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose = 2

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(accuracy*100, 2)}%")

features = pd.DataFrame(data=model.feature_importances_, index=model.feature_names_in_)
```

```
[0]	validation_0-logloss:0.68365	validation_1-logloss:0.68379
[25]	validation_0-logloss:0.54683	validation_1-logloss:0.55009
[50]	validation_0-logloss:0.49995	validation_1-logloss:0.50589
[75]	validation_0-logloss:0.48249	validation_1-logloss:0.49081
[100]	validation_0-logloss:0.47483	validation_1-logloss:0.48616
[125]	validation_0-logloss:0.47082	validation_1-logloss:0.48512
[150]	validation_0-logloss:0.46836	validation_1-logloss:0.48532
[175]	validation_0-logloss:0.46658	validation_1-logloss:0.48599
[180]	validation_0-logloss:0.46643	validation_1-logloss:0.48605
Accuracy: 79.13%
```

In [21]:
```
# Get the feature importances from the trained XGBoost model
importances = model.feature_importances_

# Create a new DataFrame to hold the features and their importances
features = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
```

```
# Sort the DataFrame by the 'Importance' column
features.sort_values('Importance', inplace=True)

features_r = features[::-1]
sns.barplot(y='Feature', x='Importance', data=features_r, palette='viridis', orient='h')

plt.title("XGBoosted Bullied On School Feature Importance")
plt.show()
```



[Back to Top]

Cyber Bullying

Logistic Regression

In [22]:

```
df = dforiginal.copy()
df = df.apply(lambda x: x.astype('category').cat.codes)

df2 = df.copy()
y = df2['Cyber_bullied_in_past_12_months']
X = df2.drop(['Bullied_not_on_school_property_in_past_12_months', 'Bullied_on_school_pro

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=19

offSchoolLR = LogisticRegression()
offSchoolLR.fit(X_train, y_train)

y_pred = offSchoolLR.predict(X_test)

offSchoolLRAccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(offSchoolLRAccuracy*100,2)}%")
```

```
Accuracy: 77.57%
```

Random Forest

In [23]:

```
cyberRF = RandomForestClassifier()
cyberRF.fit(X_train, y_train)
y_pred = onSchoolRF.predict(X_test)
```

```
cyberRFAccuracy = accuracy_score(y_test, y_pred)
print(f"On School Random Forrest Accuracy: {round(cyberRFAccuracy*100,2)}%")
```

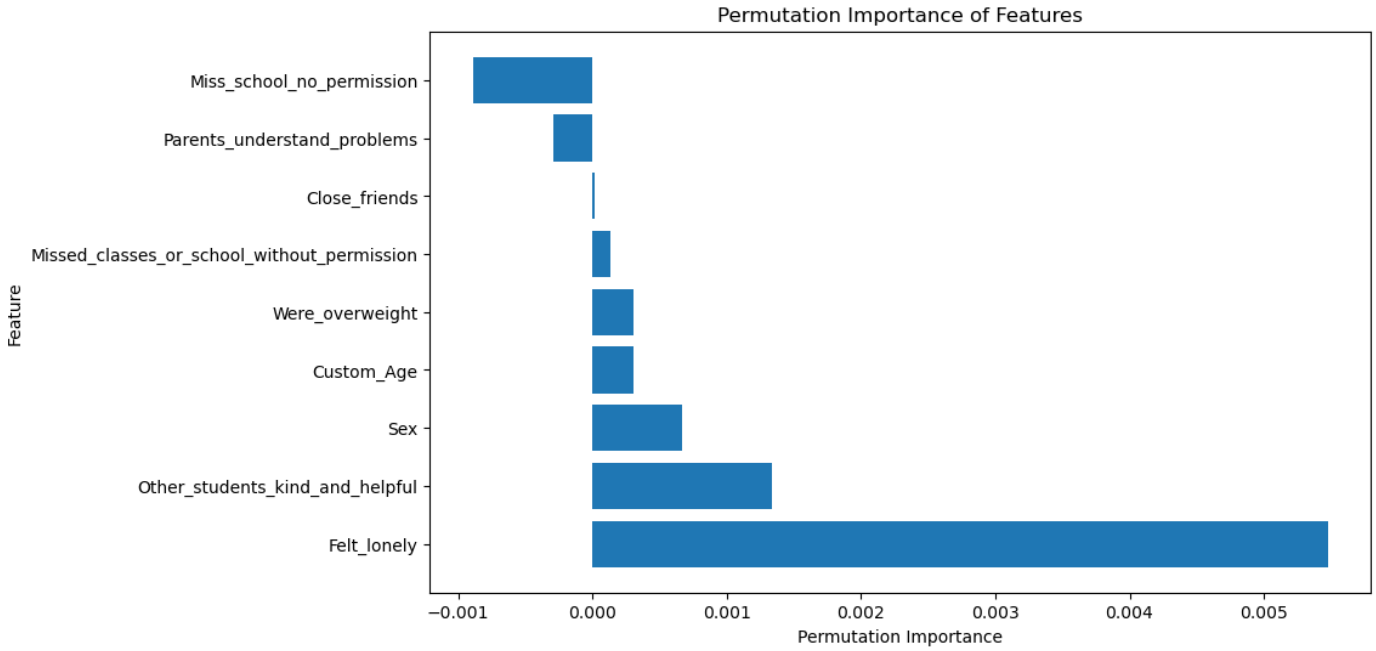
```
On School Random Forrest Accuracy: 73.86%
```

MLPClassifier

In [24]:
```
df2 = df.copy()
y = df2['Cyber_bullied_in_past_12_months']
X = df2.drop(['Bullied_not_on_school_property_in_past_12_months', 'Bullied_on_school_pro

num_input_features = X.shape[1]
hidden_layer_sizes = (num_input_features)

cyberMLP = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes, max_iter=2000)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=19

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

cyberMLPaccuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(cyberMLPaccuracy*100, 2)}%")

y_pred_prob = cyberMLP.predict_proba(X_test)[:, 1]
auc_score = roc_auc_score(y_test, y_pred_prob)
print(f"AUC: {auc_score}")
```

```
Accuracy: 77.34%
AUC: 0.6653357032355749
```

In [25]:
```
# Perform permutation importance
result = permutation_importance(cyberMLP, X_test, y_test, n_repeats=10, random_state=42)

# Get feature importances and names
importances = result.importances_mean
feature_names = X_test.columns

# Sort feature importances in descending order
sorted_indices = importances.argsort()[::-1]
sorted_importances = importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_importances)), sorted_importances, tick_label=sorted_feature_n
plt.xlabel("Permutation Importance")
plt.ylabel("Feature")
plt.title("Permutation Importance of Features")
plt.show()
```



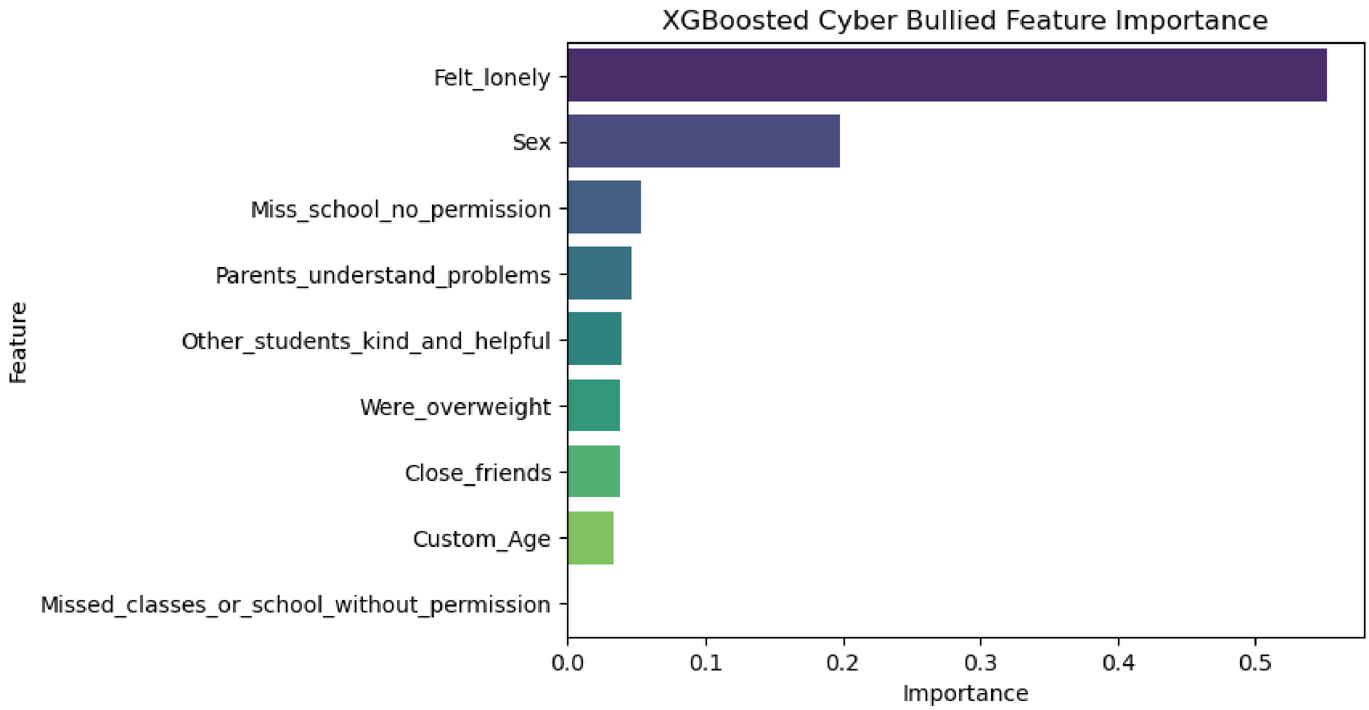
XGBoost

In [26]:
```
df = dforiginal.copy()

df = df.apply(lambda x: x.astype('category').cat.codes)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['Cyber_bullied_in_past_12_months'])
X = df.drop(['Bullied_not_on_school_property_in_past_12_months', 'Cyber_bullied_in_past_

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Create an XGBoost classifier
model = xgb.XGBClassifier(n_estimators=1000, early_stopping_rounds=50, learning_rate=0.0

# Train the model
model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose = 2

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(accuracy*100, 2)}%")

features = pd.DataFrame(data=model.feature_importances_, index=model.feature_names_in_)
```

```
[0]	validation_0-logloss:0.68433	validation_1-logloss:0.68447
[25]	validation_0-logloss:0.55749	validation_1-logloss:0.56101
[50]	validation_0-logloss:0.51413	validation_1-logloss:0.52045
[75]	validation_0-logloss:0.49808	validation_1-logloss:0.50684
[100]	validation_0-logloss:0.49142	validation_1-logloss:0.50262
[125]	validation_0-logloss:0.48792	validation_1-logloss:0.50134
[150]	validation_0-logloss:0.48577	validation_1-logloss:0.50086
[175]	validation_0-logloss:0.48392	validation_1-logloss:0.50067
[200]	validation_0-logloss:0.48233	validation_1-logloss:0.50085
[225]	validation_0-logloss:0.48170	validation_1-logloss:0.50103
Accuracy: 77.71%
```

In [27]:
```
# Get the feature importances from the trained XGBoost model
importances = model.feature_importances_

# Create a new DataFrame to hold the features and their importances
features = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
```

```python
# Sort the DataFrame by the 'Importance' column
features.sort_values('Importance', inplace=True)

features_r = features[::-1]
sns.barplot(y='Feature', x='Importance', data=features_r, palette='viridis', orient='h')

plt.title("XGBoosted Cyber Bullied Feature Importance")
plt.show()
```



[Back to Top]

Conclusions

1 - The features in this dataset are mostly correlated with each other.

In the Statistics section, it is noted that the only statistically insignificant measurements were:

- `Were_overweight` vs `Close_friends` [X2(3) = 4.4463, p > 0.05]
- `Were_overweight` vs `Missed_classes_or_school_without_permission` [X2(1) = 1.45213, p > 0.05]
- `Were_overweight` vs `Other_students_kind_and_helpful` [X2(4) = 6.1825, p>0.05]

This means that, apart from `Feeling_lonely` and a few secondary features, the various machine learning models developed could have weighed the categories differently and still achieved similar accuracy numbers. Experimenting with different `random_state` values suggests this to be true.

2 - All four models used have similar accuracy scores across Off School, On School, and Cyber Bullying predictions.

All the trained models have accuracies between 73-80%. This suggests that the categories chosen by the World Health Organization in their Global Student Health Survey were good predictors of bullying. While there is room for improving model performance through hyperparameter tuning and data cleaning techniques like one-hot encoding, it also suggests that there may be reasons for bullying that these categories simply cannot account for.

3 - Feeling lonely is significantly correlated with bullying in all three types of bullying assessed.

In every model that used weights, `Felt_Lonely` was the category that most strongly influenced its predictions, usually by a large margin. The correlation is also statistically significant when compared to the three bullying categories:

- Cyber Bullying [$X2(4) = 1827.33$, $p < 0.05$]
- Off School Bullying [$X2(4) = 1537.52$, $p < 0.05$]
- On School Bullying [$X2(4) = 1469.73$, $p < 0.05$]

4 - Secondary categories

When examining the weights of MLPClassifier and XGBoost models, certain features show unique contributions to specific types of bullying:

- In Cyber Bullying, `Sex` carries more weight in predictions.
- In On School Bullying, `Other_students_kind_and_helpful` carries more weight in predictions.
- Further exploration is needed for Off School Bullying. There may be a stronger relationship with features `Miss_school_no_permission` and `Close_friends`. Data cleaning techniques like imputation and one-hot encoding might help provide a clearer understanding of these relationships.

[Back to Top]

Acknowledgements

- Bullying in schools dataset by LEONARDO MARTINELLI: I acknowledge the valuable contribution of LEONARDO MARTINELLI for providing the "Bullying in schools" dataset on Kaggle. Your efforts in curating this dataset enable researchers and data scientists to address the critical issue of bullying, and we are grateful for your contribution to the community.

- ChatGPT by OpenAI: I express my sincere gratitude to OpenAI for providing access to their powerful language model, ChatGPT. This remarkable tool has been instrumental in my technical troubleshooting and has significantly assisted me in finding the right wording and solutions to various challenges.

This notebook utilizes the following libraries:

- Seaborn: Used for creating visually appealing graphs. © 2023 Seaborn Development Team. All rights reserved. Used under permission. Seaborn is distributed under the BSD 3-Clause License.

- NumPy: Used for numerical computations and array manipulation. © 2023 NumPy Developers. All rights reserved. Used under permission. NumPy is distributed under the BSD 3-Clause License.

- SciPy: Used for statistical computations and tests. © 2023 SciPy Developers. All rights reserved. Used under permission. SciPy is distributed under the BSD 3-Clause License.

- scikit-learn (sklearn): Used for machine learning tasks. © 2023 scikit-learn Developers. All rights reserved. Used under permission. scikit-learn is distributed under the BSD 3-Clause License.

- imbalanced-learn (imblearn): Used for handling imbalanced datasets. © 2023 imbalanced-learn Developers. All rights reserved. Used under permission. imbalanced-learn is distributed under the MIT License.

- XGBoost: Used for gradient boosting tasks. © 2023 Contributors. All rights reserved. Used under permission. XGBoost is distributed under the Apache License 2.0.

Please consult the respective library documentation for additional licensing details and any specific copyright or usage terms associated with these libraries.

[Back to Top]