Linear Regression Model

In [9]:
```
#Linear Regression
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

#this version of data processing uses years 2016 - 2018

# File paths
file1 = '/Users/aydaozisik/Downloads/monthly_precipitation.csv'  # (2016 - 2022)
file2 = '/Users/aydaozisik/Downloads/full_year_2018.csv'  # (only 2018)
file3 = '/Users/aydaozisik/Downloads/Filtered_Fire_Incidents.csv'  # (2013 - 2020)
file4 = '/Users/aydaozisik/Downloads/full_year_2016.csv'  # (only 2016)
file5 = '/Users/aydaozisik/Downloads/full_year_2017.csv'  # (only 2017)


rain = pd.read_csv(file1)
fire_incidents = pd.read_csv(file3)
noaa_18 = pd.read_csv(file2)
noaa_16 = pd.read_csv(file4)
noaa_17 = pd.read_csv(file5)

#filter rain and fire datasets to include only the years 2016-2018
rain['DATE'] = pd.to_datetime(rain['DATE'], format='%Y-%m')
rain = rain[rain['DATE'].dt.year.between(2016, 2018)]  # Filter for 2016-2018


#remove rows where both LATITUDE and LONGITUDE = 0
fire_incidents = fire_incidents[~((fire_incidents['LATITUDE'] == 0) & (fire_incidents['LONGITUDE'] == 0))]

fire_incidents['DATE'] = pd.to_datetime(fire_incidents['DATE'], format='%Y-%m')  # Adjust format if necessary
fire_incidents = fire_incidents[fire_incidents['DATE'].dt.year.between(2016, 2018)]  # Filter for 2016-2018

rain['LATITUDE'] = rain['LATITUDE'].round(2)
rain['LONGITUDE'] = rain['LONGITUDE'].round(2)

fire_incidents['LATITUDE'] = fire_incidents['LATITUDE'].round(2)
fire_incidents['LONGITUDE'] = fire_incidents['LONGITUDE'].round(2)

noaa_16['LATITUDE'] = noaa_16['LATITUDE'].round(2)
noaa_16['LONGITUDE'] = noaa_16['LONGITUDE'].round(2)

noaa_17['LATITUDE'] = noaa_17['LATITUDE'].round(2)
noaa_17['LONGITUDE'] = noaa_17['LONGITUDE'].round(2)

noaa_18['LATITUDE'] = noaa_18['LATITUDE'].round(2)
noaa_18['LONGITUDE'] = noaa_18['LONGITUDE'].round(2)

#merge rain and fire datasets
merged_rain_fire = pd.merge(rain, fire_incidents, on=['LATITUDE', 'LONGITUDE'], how='inner')

#merged rain and fire data with the multiple NOAA datasets for respective years (2016-2018)
merged_all = pd.merge(merged_rain_fire, noaa_16, on=['LATITUDE', 'LONGITUDE'], how='left', suffixes=('', '_16'))
merged_all = pd.merge(merged_all, noaa_17, on=['LATITUDE', 'LONGITUDE'], how='left', suffixes=('', '_17'))
merged_all = pd.merge(merged_all, noaa_18, on=['LATITUDE', 'LONGITUDE'], how='left', suffixes=('', '_18'))


merged_all = merged_all.drop(columns=['STATION', 'NAME', 'ELEVATION'])


X = merged_all[['TotalPrecipitation', 'PRCP', 'TAVG']]  # Independent variables
y = merged_all['AcresBurned']  # Dependent variable (target)

# 7. Handle missing values (fill with 0 or another imputation method)
X = X.fillna(0)  # Fill missing feature values with 0
y = y.fillna(0)  # Fill missing target values with 0

# 8. Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 9. Create the linear regression model and train it
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# 10. Predict on the test set
y_pred = model.predict(X_test)

# 11. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the results
print("Mean Squared Error:", mse)
print("R²:", r2)
```

```
Mean Squared Error: 110.46847596370779
R²: 0.002402376360975622
```

The second filtering method only uses the year 2018 from all three data sets:

In [10]:

```
# Load the datasets
full_year = pd.read_csv('/Users/aydaozisik/Downloads/full_year_2018.csv')
fire_occurrences = pd.read_csv('/Users/aydaozisik/Downloads/Filtered_Fire_Incidents.csv')
rain = pd.read_csv('/Users/aydaozisik/Downloads/monthly_precipitation.csv')

#covnert to DATE format, although this was already done in csv files
full_year['DATE'] = pd.to_datetime(full_year['DATE'], format='%Y-%m')  # Convert full_year DATE column
fire_occurrences['DATE'] = pd.to_datetime(fire_occurrences['DATE'], format='%Y-%m')
rain['DATE'] = pd.to_datetime(rain['DATE'], format='%Y-%m')

# Filter fire_occurrences and rain datasets for 2018
fire_occurrences = fire_occurrences[fire_occurrences['DATE'].dt.year == 2018]
rain = rain[rain['DATE'].dt.year == 2018]

# Round LATITUDE and LONGITUDE to 1 decimal place for both datasets
full_year['LATITUDE'] = full_year['LATITUDE'].round(2)
full_year['LONGITUDE'] = full_year['LONGITUDE'].round(2)

fire_occurrences['LATITUDE'] = fire_occurrences['LATITUDE'].round(2)
fire_occurrences['LONGITUDE'] = fire_occurrences['LONGITUDE'].round(2)

rain['LATITUDE'] = rain['LATITUDE'].round(2)
rain['LONGITUDE'] = rain['LONGITUDE'].round(2)

#filter columns
rain = rain[['LATITUDE', 'LONGITUDE', 'DATE', 'TotalPrecipitation']]
full_year = full_year[['LATITUDE', 'LONGITUDE', 'DATE', 'EVAP', 'AWND', 'TAVG', 'PRCP', 'WDMV']]
fire_occurrences = fire_occurrences[['LATITUDE', 'LONGITUDE', 'DATE', 'AcresBurned']]

#merge full_year with fire_occurrences on LATITUDE, LONGITUDE, and DATE
merged_data = pd.merge(
    full_year,
    fire_occurrences,
    on=['LATITUDE', 'LONGITUDE', 'DATE'],
    how='left'
)

#merge rain data with LATITUDE, LONGITUDE, and DATE
merged_data = pd.merge(
    merged_data,
    rain,
    on=['LATITUDE', 'LONGITUDE', 'DATE'],
    how='left'
)

# fill missing values in the dataset with 0
# this most like creates immense inaccuracy?
merged_data_v2 = merged_data.fillna(0)

# predicting AcresBurned
x = merged_data_v2[['TAVG', 'PRCP', 'AWND', 'WDMV','EVAP', 'TotalPrecipitation']]  # Independent variables
y = merged_data_v2['AcresBurned']  # Dependent variable (continuous: AcresBurned)



#split into training and testing
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

#train linear model
model = LinearRegression()
model.fit(X_train, y_train)

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

```
#find mean squared error and R²
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R²: {r2:.4f}")
```

```
Mean Squared Error: 18.7016
R²: -0.0005
```

In [11]:
```
merged_data_v2
```

Out[11]:

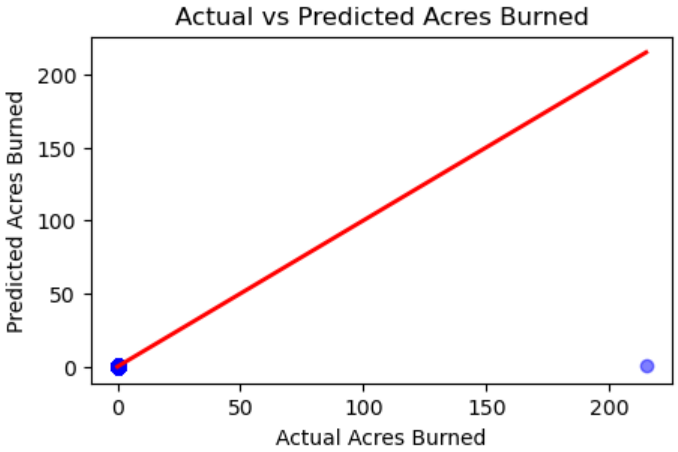
	LATITUDE	LONGITUDE	DATE	EVAP	AWND	TAVG	PRCP	WDMV	AcresBurned	TotalPrecipitation
0	39.02	-122.41	2018-01-01	0.0	0.0	49.3	0.00	0.0	0.0	0.0
1	39.02	-122.41	2018-02-01	0.0	0.0	51.3	0.00	0.0	0.0	0.0
2	39.02	-122.41	2018-03-01	0.0	0.0	49.4	0.00	0.0	0.0	0.0
3	39.02	-122.41	2018-04-01	0.0	0.0	56.7	0.00	0.0	0.0	0.0
4	41.87	-120.16	2018-01-01	0.0	0.0	38.4	1.63	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
12353	39.74	-121.49	2018-09-01	0.0	0.0	73.4	0.00	0.0	0.0	0.0
12354	39.74	-121.49	2018-10-01	0.0	0.0	64.5	0.00	0.0	0.0	0.0
12355	39.74	-121.49	2018-12-01	0.0	0.0	44.9	0.00	0.0	0.0	0.0
12356	34.05	-118.94	2018-09-01	0.0	0.0	66.6	0.00	0.0	0.0	0.0
12357	34.05	-118.94	2018-10-01	0.0	0.0	66.5	0.00	0.0	0.0	0.0

12358 rows × 10 columns

In [13]:
```
# I dont think VIF function is necessary since we are missing a lot of values
#add constant term for the intercept
X = sm.add_constant(X)

#OLS model
model = sm.OLS(y, X).fit()

#OLS regression results
print(model.summary())

#VIF (Variance Inflation Factor)
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calculate VIF for each feature in X
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display VIF values
print("\nVIF results:")
print(vif_data)
```

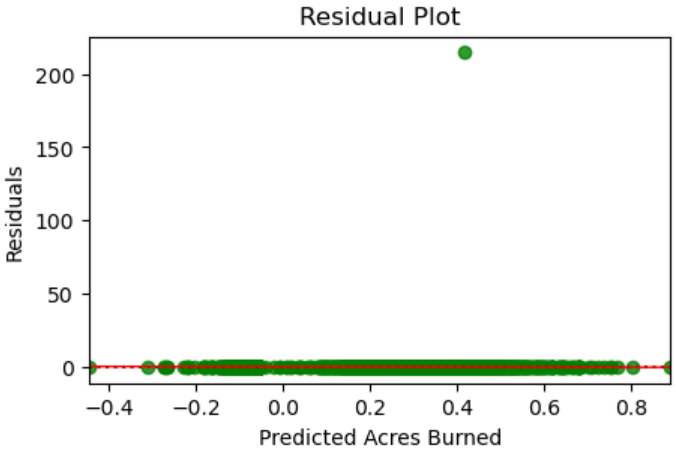
```
                            OLS Regression Results
==============================================================================
Dep. Variable:            AcresBurned   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                 -0.000
Method:                 Least Squares   F-statistic:                    0.9101
Date:                Fri, 13 Dec 2024   Prob (F-statistic):              0.486
Time:                        20:35:01   Log-Likelihood:                -45442.
No. Observations:               12358   AIC:                         9.090e+04
Df Residuals:                   12351   BIC:                         9.095e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const                 -0.0475      0.143     -0.333      0.739      -0.327       0.232
EVAP                  -0.0365      0.162     -0.225      0.822      -0.354       0.281
AWND                   0.0152      0.053      0.286      0.775      -0.089       0.119
TAVG                   0.0061      0.003      2.025      0.043       0.000       0.012
PRCP                  -0.0065      0.036     -0.181      0.856      -0.077       0.064
WDMV               -4.308e-05      0.001     -0.055      0.956      -0.002       0.001
TotalPrecipitation    -0.0492      0.108     -0.457      0.648      -0.260       0.162
==============================================================================
Omnibus:                    47365.895   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):      39190333345.028
Skew:                          89.113   Prob(JB):                         0.00
Kurtosis:                    8725.287   Cond. No.                         277.
==============================================================================

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

VIF results:
              Feature       VIF
0               const  2.750194
1                EVAP  1.783333
2                AWND  1.225906
3                TAVG  1.167263
4                PRCP  1.100303
5                WDMV  1.775397
6  TotalPrecipitation  1.172663
```

In [17]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

#linear regression
plt.figure(figsize=(5, 3))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', lw=2)

# Labels and title
plt.xlabel('Actual Acres Burned')
plt.ylabel('Predicted Acres Burned')
plt.title('Actual vs Predicted Acres Burned')
plt.show()
```



In [16]:
```python
#linear regression
residuals = y_test - y_pred
plt.figure(figsize=(5, 3))
sns.residplot(x=y_pred, y=residuals, lowess=True, color="g", line_kws={'color': 'red', 'lw': 1})
plt.xlabel('Predicted Acres Burned')
```

```
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



In [ ]:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc, classification_report,
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
#Load the datasets.
full_year = pd.read_csv("/Users/jojihun/Desktop/full_year_2018_sorted.csv")
fire_occurrences = pd.read_csv("/Users/jojihun/Desktop/Fire_Occurrences_2018_sorted

#Remove time from 'DATE'column and 'DISCOVERYDATETIME'column to merge two datasets
full_year['DATE'] = full_year['DATE'].apply(lambda x: x.split(' ')[0])
fire_occurrences['DATE'] = fire_occurrences['DISCOVERYDATETIME'].apply(lambda x: x.

#Make a new column 'YEAR_MONTH' on both datasets
full_year['YEAR_MONTH'] = full_year['DATE'].apply(lambda x: '-'.join(x.split('-')[:
fire_occurrences['YEAR_MONTH'] = fire_occurrences['DATE'].apply(lambda x: '-'.join(

#Since the locations on the both datasets are represented as very detailed latitude
#I will round latitude and longitude values to one decimal point for matching.
full_year['LATITUDE'] = full_year['LATITUDE'].round(1)
full_year['LONGITUDE'] = full_year['LONGITUDE'].round(1)
fire_occurrences['X'] = fire_occurrences['X'].round(1)
fire_occurrences['Y'] = fire_occurrences['Y'].round(1)

#Select relevant columns from the datasets for merging.
full_year = full_year[['LATITUDE', 'LONGITUDE', 'YEAR_MONTH', 'EVAP', 'AWND', 'TAVG
fire_occurrences = fire_occurrences[['X', 'Y', 'YEAR_MONTH', 'TOTALACRES']]

#Merge the datasets on latitude/longitude and YEAR_MONTH.
merged_data = pd.merge(
    full_year,
    fire_occurrences,
    left_on=['LATITUDE', 'LONGITUDE', 'YEAR_MONTH'],
    right_on=['Y', 'X', 'YEAR_MONTH'],
    how='left'
)

#Create a binary column 'FIRE_OCCURRED' to indicate if a fire occurred or not.
merged_data['FIRE_OCCURRED'] = (merged_data['TOTALACRES'] > 0).astype(int)

#Drop unnecessary columns 'X' and 'Y' after merging.
merged_data = merged_data.drop(columns=['X', 'Y'])

#Display the distribution of the target variable('FIRE_OCCURRED').
merged_data['FIRE_OCCURRED'].value_counts()

#Fill the remaining missing values with 0 to model the dataset later.
merged_data_v1 = merged_data.fillna(0)
```

```
merged_data_v1
```

Out[2]:

	LATITUDE	LONGITUDE	YEAR_MONTH	EVAP	AWND	TAVG	PRCP	WDMV	TOTA
0	39.0	-122.4	2018-01	0.0	0.0	49.3	0.00	0.0	
1	37.6	-119.0	2018-01	0.0	0.0	34.0	1.93	0.0	
2	38.4	-123.0	2018-01	0.0	0.0	49.6	11.18	0.0	
3	39.3	-121.6	2018-01	0.0	0.0	0.0	4.57	0.0	
4	41.6	-122.9	2018-01	0.0	0.0	39.6	0.00	0.0	
...	...	...	...	...	...	...	...	...	
12436	35.7	-118.5	2018-12	0.0	0.0	45.0	0.00	0.0	
12437	36.4	-121.6	2018-12	0.0	0.0	0.0	3.49	0.0	
12438	41.0	-122.0	2018-12	0.0	0.0	0.0	8.64	0.0	
12439	41.8	-122.0	2018-12	0.0	0.0	33.2	0.00	0.0	
12440	38.4	-122.0	2018-12	0.0	5.6	48.8	0.93	0.0	

12441 rows × 10 columns

In [3]:

```
#Boosting

#Evaporation (EVAP)
#Average Wind Speed (AWND)
#Average Temperature (TAVG)
#Precipitation (PRCP)
#Wind Movement (WDMV)

#Load and preprocess the dataset.
x = merged_data_v1[['EVAP', 'AWND', 'TAVG', 'PRCP', 'WDMV']]
y = merged_data_v1['FIRE_OCCURRED']

#Split the dataset into training and testing sets(80:20).
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_sta

#Standardize the training and test features using mean and standard deviation of th
mean_train = x_train.mean()
std_train = x_train.std()
x_train_sc = (x_train - mean_train) / std_train
x_test_sc = (x_test - mean_train) / std_train

#Use KFold Cross-Validation for model evaluation and define model hyperparameters.
kf = KFold(n_splits = 10, shuffle = True, random_state = 2018)
n_estimators = 100
max_leaf_nodes = 10

#Store indices for each fold in cross-validation.
```

```
kfold_index = []
for index in kf.split(x_train_sc):
    kfold_index.append(index)

#CV loop to compute CV accuracy scores.
cv_scores = []
for train_index, val_index in kf.split(x_train_sc):
    x_kf_train, x_kf_val = x_train_sc.iloc[train_index], x_train_sc.iloc[val_index]
    y_kf_train, y_kf_val = y_train.iloc[train_index], y_train.iloc[val_index]

    #Train Gradient Boosting Classifier.
    gbm = GradientBoostingClassifier(n_estimators = n_estimators, max_leaf_nodes =
    gbm.fit(x_kf_train, y_kf_train)

    #Compute cv accuracy scores.
    y_val_pred = gbm.predict(x_kf_val)
    val_accuracy = accuracy_score(y_kf_val, y_val_pred)
    cv_scores.append(val_accuracy)

#Mean CV Accuracy result
mean_cv_accuracy = np.mean(cv_scores)
print("Mean CV Accuracy:", mean_cv_accuracy)

#Train the final model on the entire training set.
final_gbm = GradientBoostingClassifier(n_estimators = n_estimators, max_leaf_nodes
final_gbm.fit(x_train_sc, y_train)

#Gradient Boosting Classifier Test Accuracy result
y_pred_test = final_gbm.predict(x_test_sc)
gbm_model = accuracy_score(y_test, y_pred_test)
print("GBC Test Accuracy:", gbm_model)
print(classification_report(y_test, y_pred_test))
```

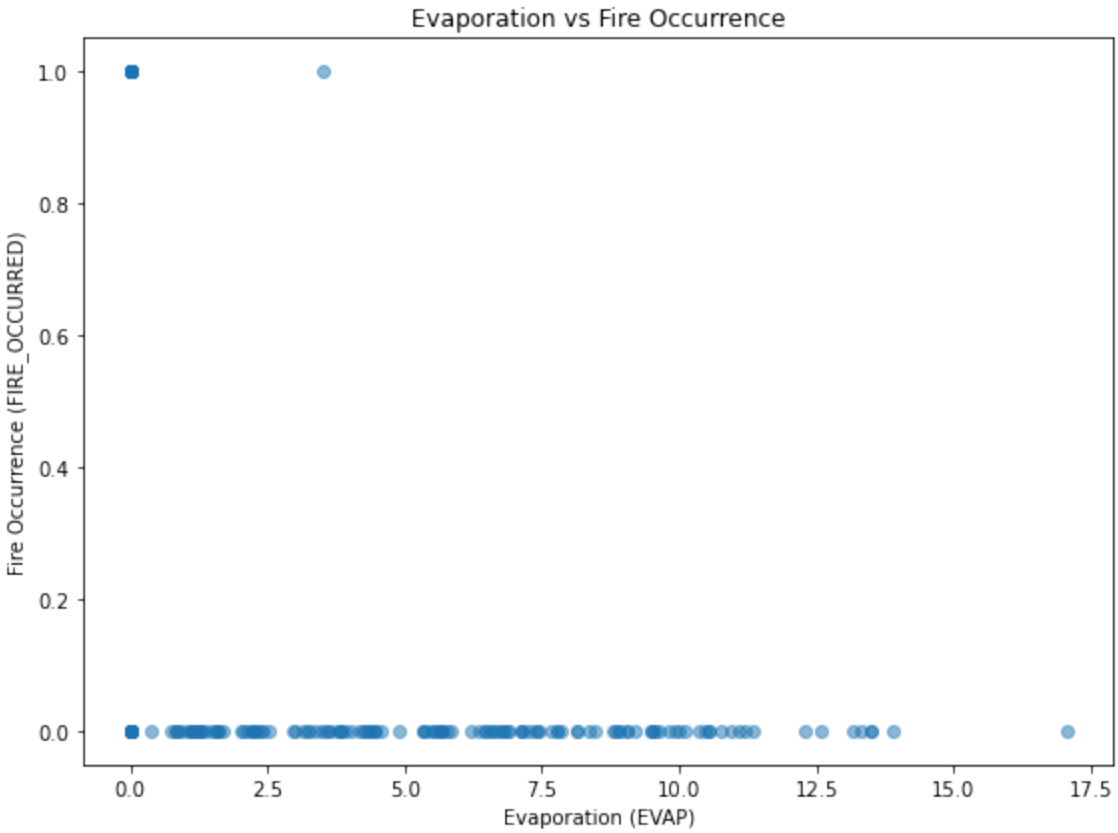
```
Mean CV Accuracy: 0.967945651954552
GBC Test Accuracy: 0.9682603455202893
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      2412
           1       0.33      0.03      0.05        77

    accuracy                           0.97      2489
   macro avg       0.65      0.51      0.52      2489
weighted avg       0.95      0.97      0.95      2489
```

In [4]:
```
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
print(f"\nMean Squared Error (MSE): {mse:.4f}")
print(f"R² Score: {r2:.4f}")
```

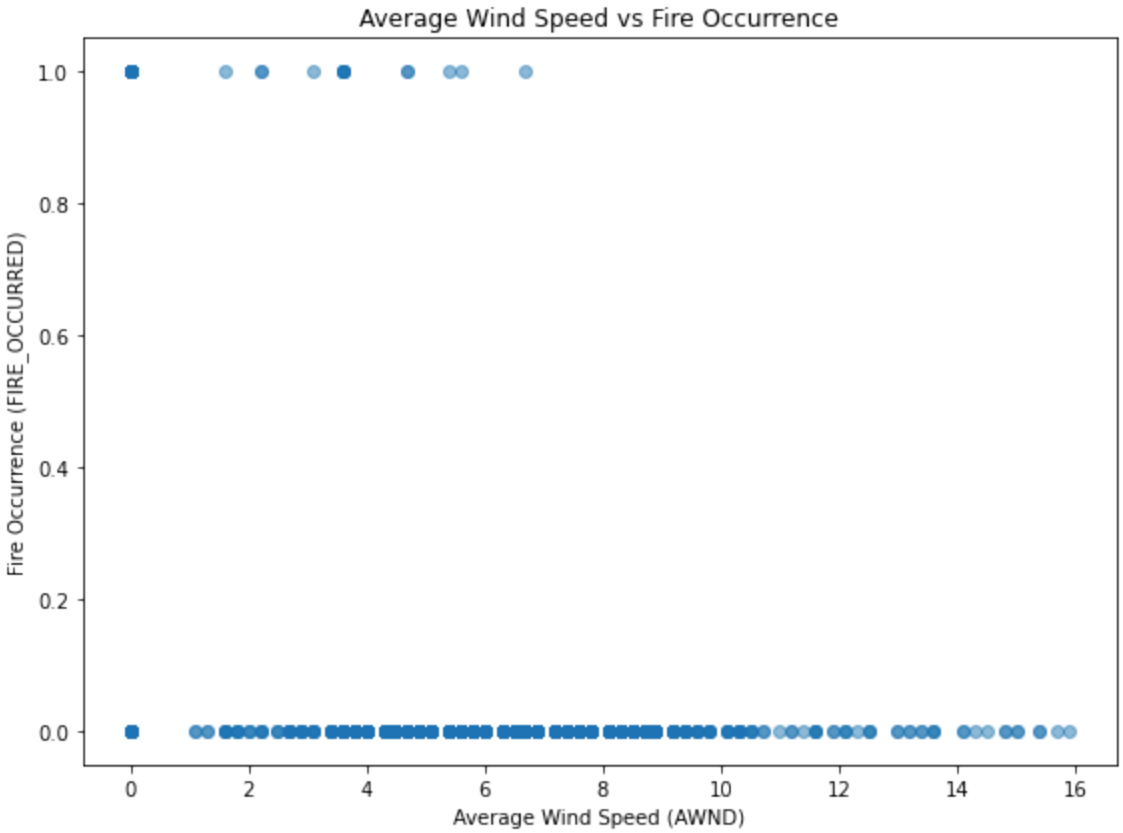
```
Mean Squared Error (MSE): 0.0317
R² Score: -0.0587
```

In [5]:
```
plt.figure(figsize=(8, 6))
plt.scatter(merged_data_v1['EVAP'], merged_data_v1['FIRE_OCCURRED'], alpha=0.5)
plt.xlabel('Evaporation (EVAP)')
plt.ylabel('Fire Occurrence (FIRE_OCCURRED)')
```

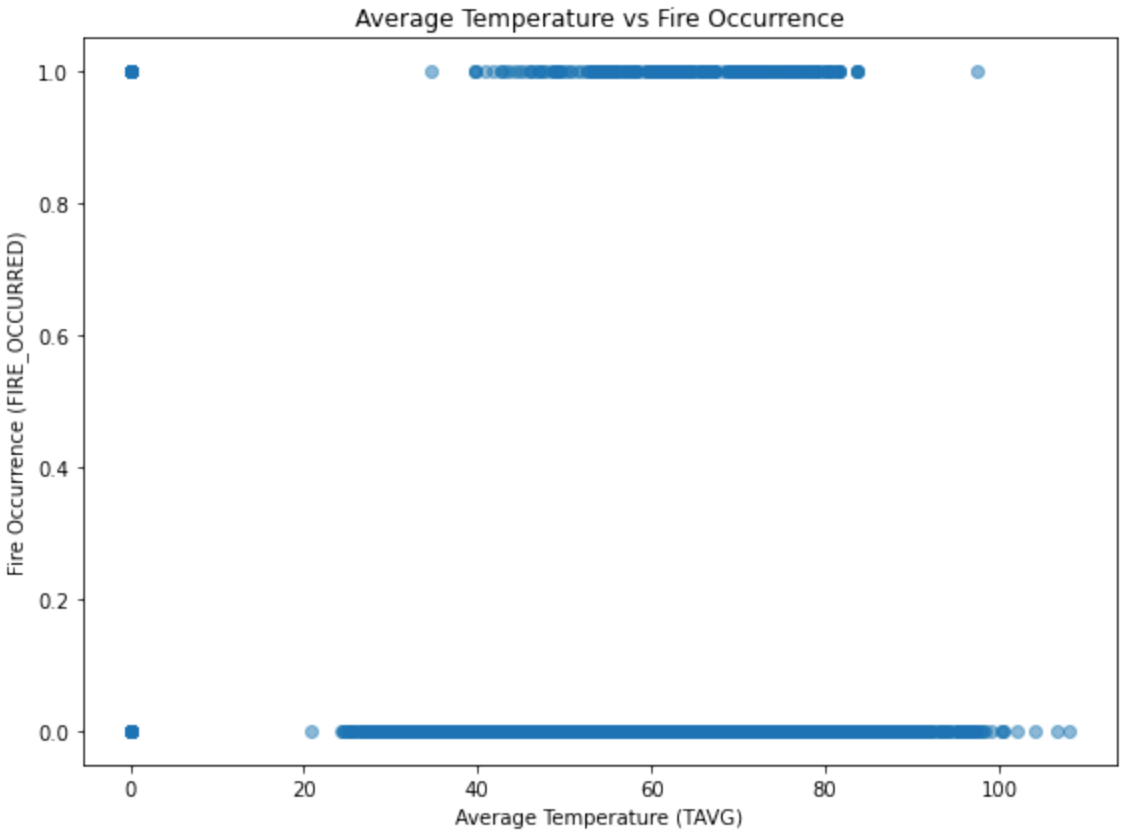
```
plt.title('Evaporation vs Fire Occurrence')
plt.tight_layout()
plt.savefig('Evaporation_vs_Fire_Occurrence.png')
plt.show()
```



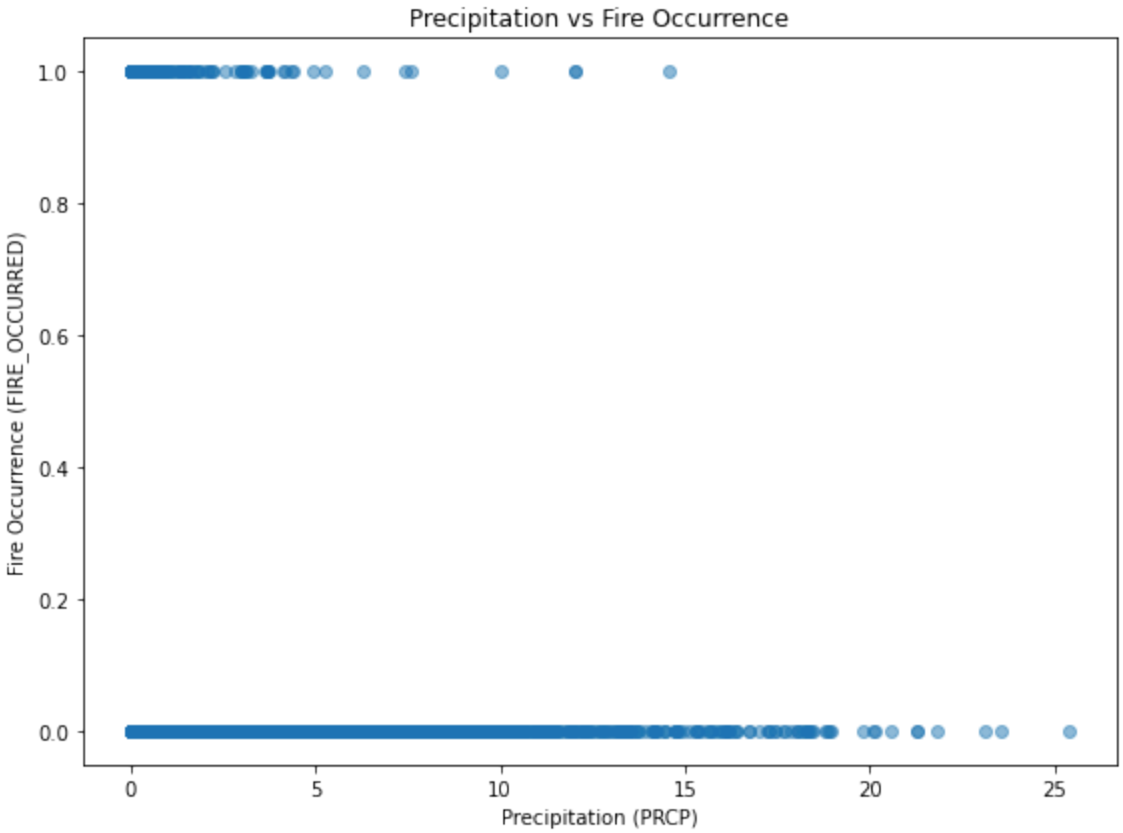
In [6]:
```
plt.figure(figsize=(8, 6))
plt.scatter(merged_data_v1['AWND'], merged_data_v1['FIRE_OCCURRED'], alpha=0.5)
plt.xlabel('Average Wind Speed (AWND)')
plt.ylabel('Fire Occurrence (FIRE_OCCURRED)')
plt.title('Average Wind Speed vs Fire Occurrence')
plt.tight_layout()
plt.savefig('Average_Wind_Speed_vs_Fire_Occurrence.png')
plt.show()
```



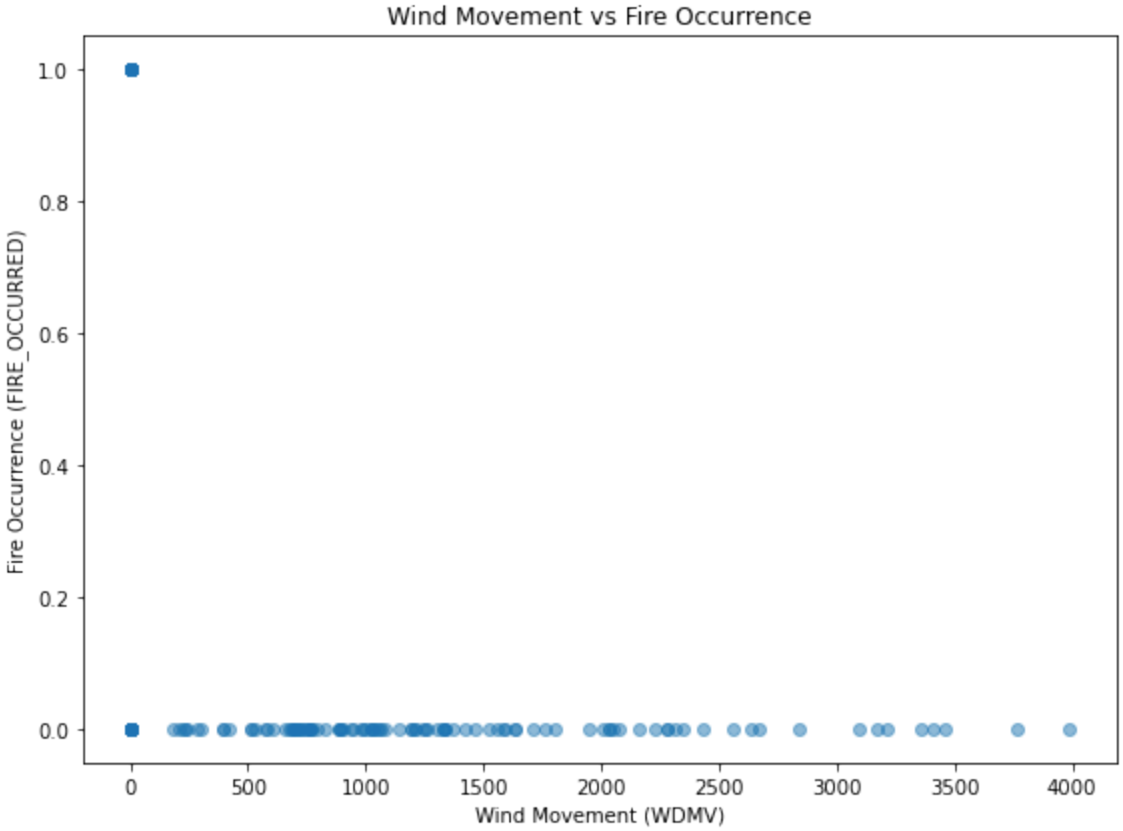
In [7]:
```
plt.figure(figsize=(8, 6))
plt.scatter(merged_data_v1['TAVG'], merged_data_v1['FIRE_OCCURRED'], alpha=0.5)
plt.xlabel('Average Temperature (TAVG)')
plt.ylabel('Fire Occurrence (FIRE_OCCURRED)')
plt.title('Average Temperature vs Fire Occurrence')
plt.tight_layout()
plt.savefig('Average_Temperature_vs_Fire_Occurrence.png')
plt.show()
```



In [8]:
```
plt.figure(figsize=(8, 6))
plt.scatter(merged_data_v1['PRCP'], merged_data_v1['FIRE_OCCURRED'], alpha=0.5)
plt.xlabel('Precipitation (PRCP)')
plt.ylabel('Fire Occurrence (FIRE_OCCURRED)')
plt.title('Precipitation vs Fire Occurrence')
plt.tight_layout()
plt.savefig('Precipitation_vs_Fire_Occurrence.png')
plt.show()
```



In [9]:
```
plt.figure(figsize=(8, 6))
plt.scatter(merged_data_v1['WDMV'], merged_data_v1['FIRE_OCCURRED'], alpha=0.5)
plt.xlabel('Wind Movement (WDMV)')
plt.ylabel('Fire Occurrence (FIRE_OCCURRED)')
plt.title('Wind Movement vs Fire Occurrence')
plt.tight_layout()
plt.savefig('Wind Movement_vs_Fire_Occurrence.png')
plt.show()
```



In [10]:
```
feature_importances = final_gbm.feature_importances_

feature_names = x.columns
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
})

print("\nFeature importances:")
print(feature_importances)
plt.figure(figsize=(8, 6))
plt.bar(importance_df['Feature'], importance_df['Importance'], alpha=0.7)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.show()
```

```
Feature importances:
[2.10732360e-02 1.08745034e-01 6.39046997e-01 2.30678991e-01
 4.55741831e-04]
```