```
!pip install ucimlrepo
```

Show hidden output

Importing Libraries

```
from ucimlrepo import fetch_ucirepo
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import RandomOverSampler
import pandas as pd
import copy
```

fetching dataframe

```
magicGammaMain = fetch_ucirepo(id=159)
```

Creating copy of the dataframe

```
magicGamma = copy.deepcopy(magicGammaMain)
```

encoding target values

```
mgdf = magicGamma.data.original
X = magicGamma.data.features
y = magicGamma.data.targets
mgdf["class"] = mgdf["class"].map(lambda x: 1 if x == "g" else 0)
y = y.map(lambda x: 1 if x == "g" else 0)
```

describing data

```
print("Features shape",X.shape)
print("Target shape",y.shape)
```

```
mgdf.describe()
```

```
Features shape (19020, 10)
Target shape (19020, 1)
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	
count	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	1902
mean	53.250154	22.180966	2.825017	0.380327	0.214657	-4.331745	10.545545	0.249726	27.645707	193.818026	
std	42.364855	18.346056	0.472599	0.182813	0.110511	59.206062	51.000118	20.827439	26.103621	74.731787	
min	4.283500	0.000000	1.941300	0.013100	0.000300	-457.916100	-331.780000	-205.894700	0.000000	1.282600	
25%	24.336000	11.863800	2.477100	0.235800	0.128475	-20.586550	-12.842775	-10.849375	5.547925	142.492250	
50%	37.147700	17.139900	2.739600	0.354150	0.196500	4.013050	15.314100	0.666200	17.679500	191.851450	
75%	70.122175	24.739475	3.101600	0.503700	0.285225	24.063700	35.837800	10.946425	45.883550	240.563825	
max	334.177000	256.382000	5.323300	0.893000	0.675200	575.240700	238.321000	179.851000	90.000000	495.561000	

```
mgdf
```

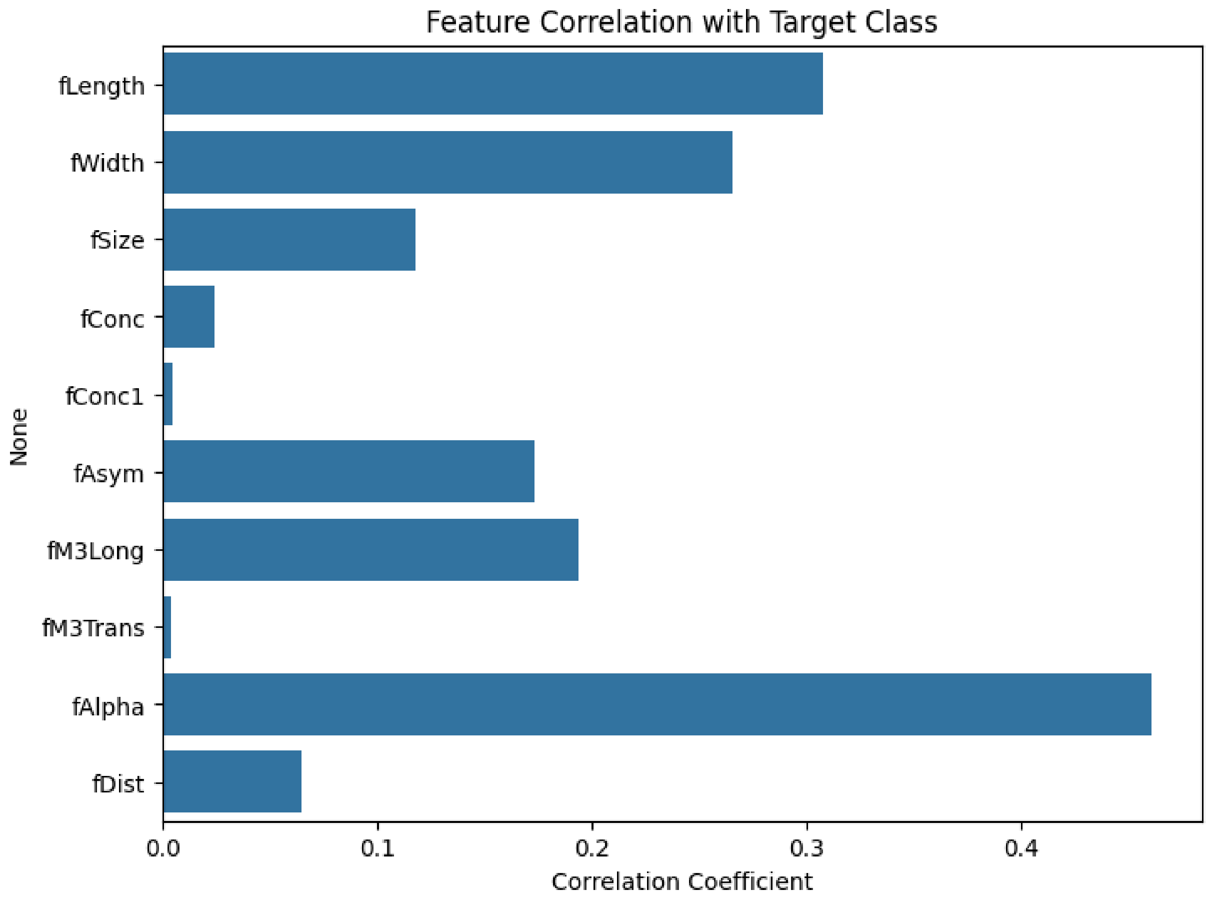
	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	1
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	1
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	1
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	1
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	1
...	...	...	...	...	...	...	...	...	...	...	...
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4229	106.8258	0
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7975	247.4560	0
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2987	256.5166	0
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6874	408.3166	0
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7310	272.3174	0

19020 rows × 11 columns

Next steps: Generate code with mgdf New interactive sheet

finding correlation between features and target values

```
corr = mgdf.corr(numeric_only=True)["class"].abs()
corr = corr.drop("class")

plt.figure(figsize=(8, 6))
sns.barplot(x=corr.values, y=corr.index)
plt.title("Feature Correlation with Target Class")
plt.xlabel("Correlation Coefficient")
plt.show()
```

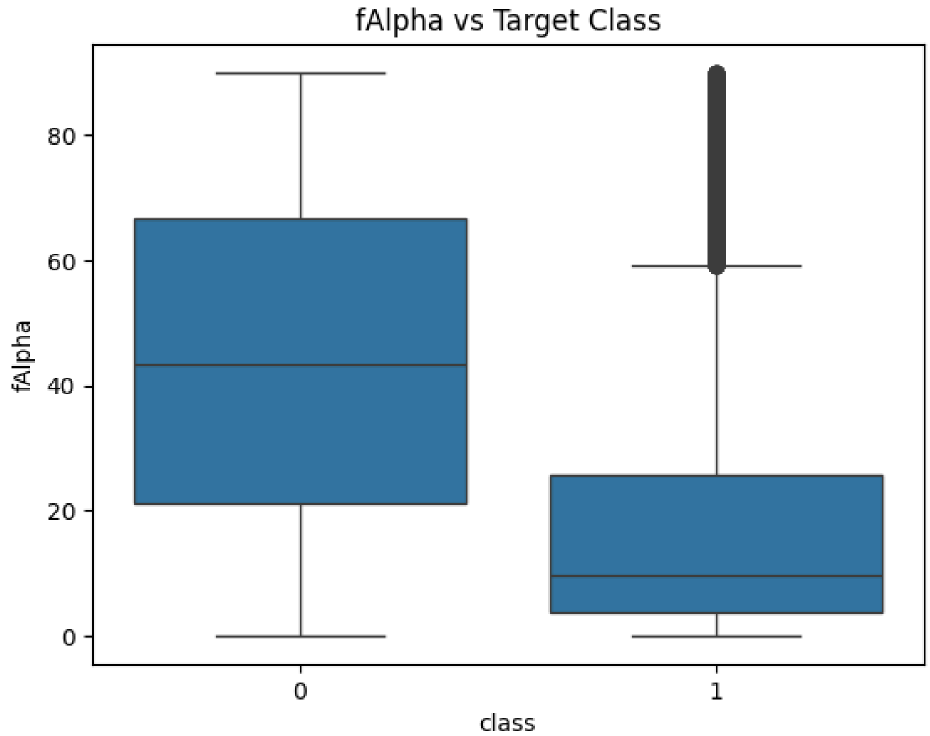


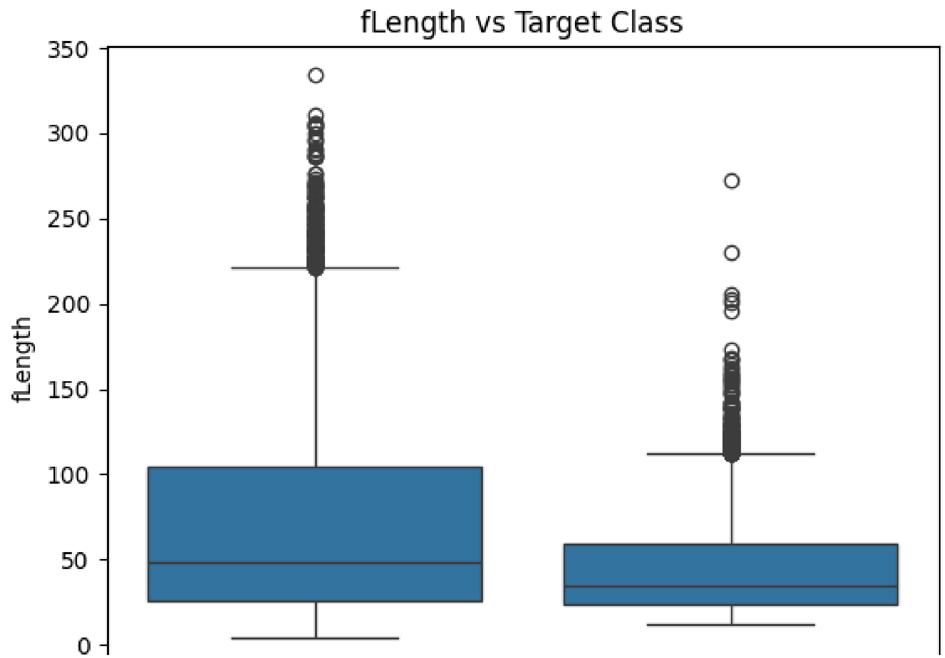
selecting features which have higher correlation that `>=0.02`

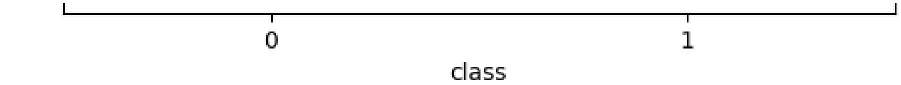
```
selected_features = corr[corr >= 0.02].index.tolist()

print(f"Using {len(selected_features)} features:")
print(selected_features)

X = X[selected_features]
mgdf = mgdf[selected_features + ["class"]]
```

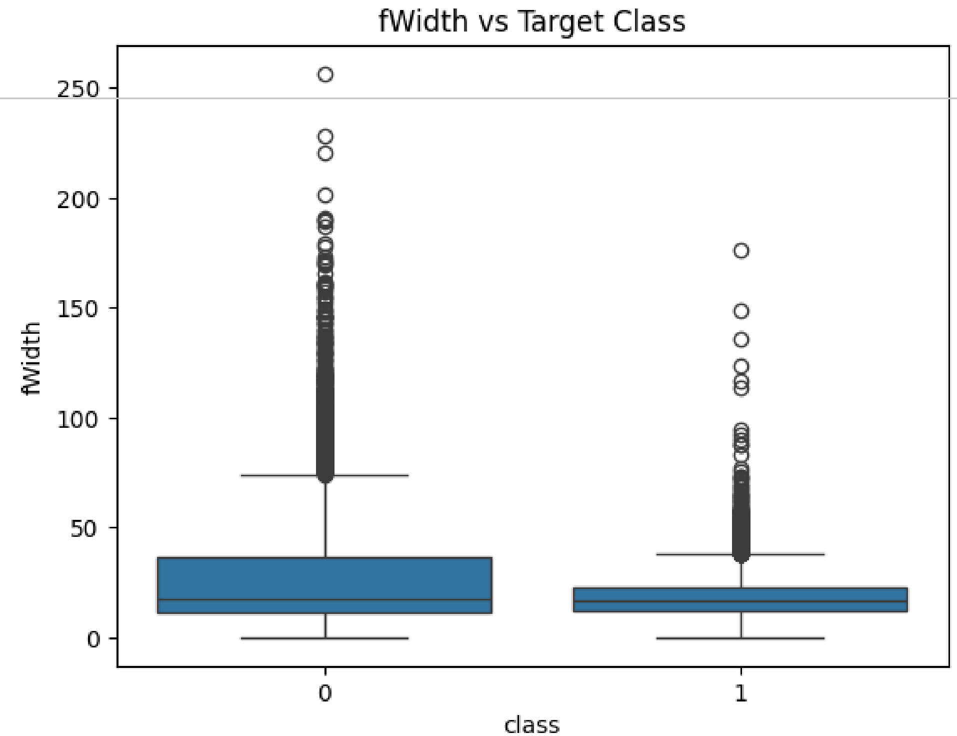
```
Using 8 features:
['fLength', 'fWidth', 'fSize', 'fConc', 'fAsym', 'fM3Long', 'fAlpha', 'fDist']
```

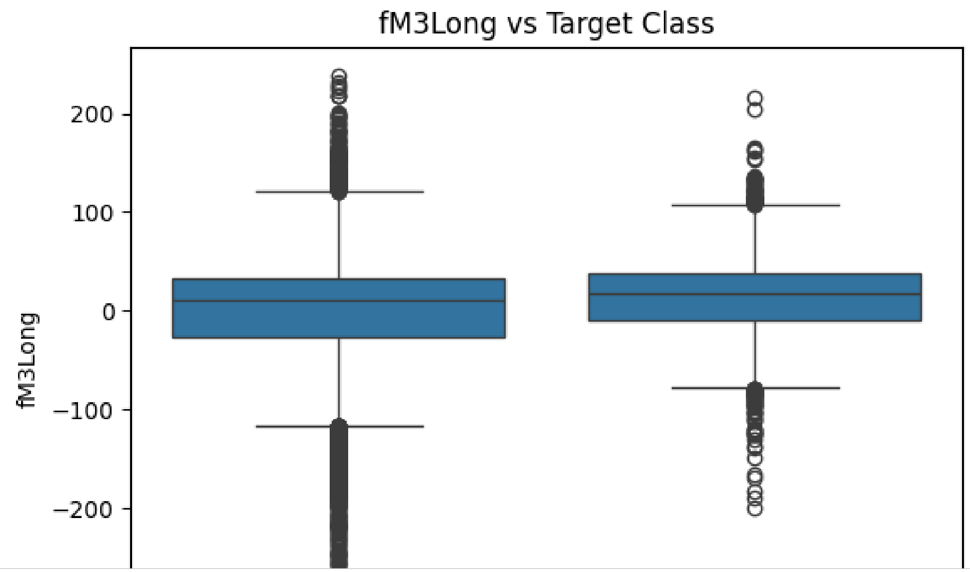
```
top_features = corr[selected_features].sort_values(ascending=False).head(4).index

for tf in top_features:
    sns.boxplot(x=y["class"], y=X[tf])
    plt.title(f"{tf} vs Target Class")
    plt.show()
```

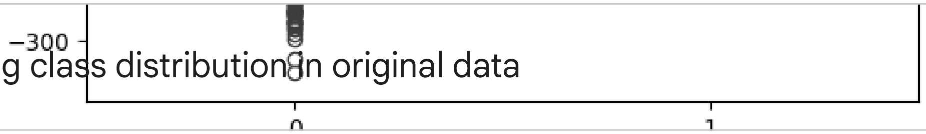

fAlpha vs Target Class
fAlpha
0
20
40
60
80
0
1
class


fLength vs Target Class
fLength
0
50
100
150
200
250
300
350


0
1
class


fWidth vs Target Class
250
200
150
100
50
0
fWidth
0
1
class


fM3Long vs Target Class
200
100
0
−100
−200
fM3Long



showing class distribution in original data

```
print(y.value_counts())

sns.countplot(x=y["class"])
plt.title("Class Distribution (Before Oversampling)")
plt.show()
```

```
class
1        12332
0         6688
Name: count, dtype: int64
```