Oil Spill

이 데이터셋은 해양 위성 이미지를 기반으로 개발되었습니다. 일부 이미지는 기름 유출을 포함하고 있고, 일부는 포함하지 않습니다. 이미지는 섹션으로 나뉘고, 컴퓨터 비전 알고리즘을 사용하여 이미지 섹션 또는 패치의 내용을 설명하는 특징 벡터로 처리되었습니다. 이 작업의 목표는 위성 이미지 패치의 내용을 설명하는 벡터를 기반으로 해당 패치가 기름 유출(예: 해양에서의 불법 또는 사고로 인한 기름 방출)을 포함하는지 여부를 예측하는 것입니다.

이 데이터셋에는 두 가지 클래스가 있으며, 주어진 해양 패치의 특징을 사용하여 유출과 비유출을 구별하는 것이 목표입니다.

- 비유출: 음성 사례 또는 다수 클래스 (0)
- 기름 유출: 양성 사례 또는 소수 클래스 (1)

In [38]:
```
from IPython.display import Image

Image('/Users/youngjinseo/Desktop/파이썬/oil_spill_in_a_marina_photograph.png')
```

Out[38]:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

from sklearn.pipeline import Pipeline
```

In [2]:
```
df = pd.read_csv("/Users/youngjinseo/Desktop/파이썬/oil_spill.csv")
```

In [3]:
```
df.head(8)
```

Out[3]:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_41	f_42	f_43	f_44	f_45	f_46	f_47	f_48	f_49	target
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	2650.00	1000.00	763.16	135.46	3.73	0	33243.19	65.74	7.95	1
1	2	22325	79.11	841.03	180	55812500	51.11	1.21	61900.0	0.02	...	5750.00	11500.00	9593.48	1648.80	0.60	0	51572.04	65.73	6.26	0
2	3	115	1449.85	608.43	88	287500	40.42	7.34	3340.0	0.18	...	1400.00	250.00	150.00	45.13	9.33	1	31692.84	65.81	7.84	1
3	4	1201	1562.53	295.65	66	3002500	42.40	7.97	18030.0	0.19	...	6041.52	761.58	453.21	144.97	13.33	1	37696.21	65.67	8.07	1
4	5	312	950.27	440.86	37	780000	41.43	7.03	3350.0	0.17	...	1320.04	710.63	512.54	109.16	2.58	0	29038.17	65.66	7.35	0
5	6	54	1438.13	544.91	82	135000	44.67	6.92	1570.0	0.15	...	608.28	200.00	150.00	52.22	4.06	0	30967.25	65.77	7.85	1
6	7	116	1446.29	580.94	97	290000	41.53	6.24	3660.0	0.15	...	1060.66	403.11	164.58	114.82	6.44	0	31258.37	65.79	7.85	1
7	8	57	28.68	715.39	141	142500	51.67	0.83	1810.0	0.02	...	500.00	360.56	165.71	132.47	3.02	0	51985.06	65.67	6.25	0

8 rows × 50 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 937 entries, 0 to 936
Data columns (total 50 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   f_1     937 non-null    int64
 1   f_2     937 non-null    int64
 2   f_3     937 non-null    float64
 3   f_4     937 non-null    float64
 4   f_5     937 non-null    int64
 5   f_6     937 non-null    int64
 6   f_7     937 non-null    float64
 7   f_8     937 non-null    float64
 8   f_9     937 non-null    float64
 9   f_10    937 non-null    float64
 10  f_11    937 non-null    float64
 11  f_12    937 non-null    float64
 12  f_13    937 non-null    float64
 13  f_14    937 non-null    float64
 14  f_15    937 non-null    float64
 15  f_16    937 non-null    float64
 16  f_17    937 non-null    float64
 17  f_18    937 non-null    float64
 18  f_19    937 non-null    float64
 19  f_20    937 non-null    float64
 20  f_21    937 non-null    float64
 21  f_22    937 non-null    float64
 22  f_23    937 non-null    int64
 23  f_24    937 non-null    float64
 24  f_25    937 non-null    float64
 25  f_26    937 non-null    float64
 26  f_27    937 non-null    float64
 27  f_28    937 non-null    float64
 28  f_29    937 non-null    float64
 29  f_30    937 non-null    float64
 30  f_31    937 non-null    float64
 31  f_32    937 non-null    float64
 32  f_33    937 non-null    float64
 33  f_34    937 non-null    float64
 34  f_35    937 non-null    int64
 35  f_36    937 non-null    int64
 36  f_37    937 non-null    float64
 37  f_38    937 non-null    float64
 38  f_39    937 non-null    int64
 39  f_40    937 non-null    int64
 40  f_41    937 non-null    float64
 41  f_42    937 non-null    float64
 42  f_43    937 non-null    float64
 43  f_44    937 non-null    float64
 44  f_45    937 non-null    float64
 45  f_46    937 non-null    int64
 46  f_47    937 non-null    float64
 47  f_48    937 non-null    float64
 48  f_49    937 non-null    float64
 49  target  937 non-null    int64
dtypes: float64(39), int64(11)
memory usage: 366.1 KB
```

In [5]:
```
df = df.drop('f_1',axis = 1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 937 entries, 0 to 936
Data columns (total 49 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   f_2     937 non-null    int64
 1   f_3     937 non-null    float64
 2   f_4     937 non-null    float64
 3   f_5     937 non-null    int64
 4   f_6     937 non-null    int64
 5   f_7     937 non-null    float64
 6   f_8     937 non-null    float64
 7   f_9     937 non-null    float64
 8   f_10    937 non-null    float64
 9   f_11    937 non-null    float64
 10  f_12    937 non-null    float64
 11  f_13    937 non-null    float64
 12  f_14    937 non-null    float64
 13  f_15    937 non-null    float64
 14  f_16    937 non-null    float64
 15  f_17    937 non-null    float64
 16  f_18    937 non-null    float64
 17  f_19    937 non-null    float64
 18  f_20    937 non-null    float64
 19  f_21    937 non-null    float64
 20  f_22    937 non-null    float64
 21  f_23    937 non-null    int64
 22  f_24    937 non-null    float64
 23  f_25    937 non-null    float64
 24  f_26    937 non-null    float64
 25  f_27    937 non-null    float64
 26  f_28    937 non-null    float64
 27  f_29    937 non-null    float64
 28  f_30    937 non-null    float64
 29  f_31    937 non-null    float64
 30  f_32    937 non-null    float64
 31  f_33    937 non-null    float64
 32  f_34    937 non-null    float64
 33  f_35    937 non-null    int64
 34  f_36    937 non-null    int64
 35  f_37    937 non-null    float64
 36  f_38    937 non-null    float64
 37  f_39    937 non-null    int64
 38  f_40    937 non-null    int64
 39  f_41    937 non-null    float64
 40  f_42    937 non-null    float64
 41  f_43    937 non-null    float64
 42  f_44    937 non-null    float64
 43  f_45    937 non-null    float64
 44  f_46    937 non-null    int64
 45  f_47    937 non-null    float64
 46  f_48    937 non-null    float64
 47  f_49    937 non-null    float64
 48  target  937 non-null    int64
dtypes: float64(39), int64(10)
memory usage: 358.8 KB
```

In [8]:
```
#x,y 나누기

x = df.drop('target',axis = 1)
y = df['target']
```

In [9]:
```
#train,test

xtrain,xtest,ytrain,ytest = train_test_split(x,y,stratify = y, test_size = 0.2, random_state =42)
```

데이터 스케일링

In [10]:
```
#데이터 스케일링

scaler = StandardScaler()
xtrainscale = scaler.fit_transform(xtrain)
xtestscale = scaler.fit_transform(xtest)
```

cross_val_score

In [15]:
```
#모델 학습
rf = RandomForestClassifier()

rf.fit(xtrainscale,ytrain)
print(rf.score(xtestscale,ytest))
```

```
0.9627659574468085
```

In [21]:
```
#StratifiedKFold 설정
strkf = StratifiedKFold(n_splits=5, shuffle=True, random_state=50)
scores = cross_val_score(rf,xtrainscale,ytrain,cv =strkf)

print(np.mean(scores))
```

```
0.9652975391498881
```

하이퍼파라미터

In [22]:
```
#Grid Search
#최적 하이퍼파라미터:
params = {'max_depth': range(5, 20, 2),
          'min_samples_split':range(2, 100, 10),
          'n_estimators':range(50,100,50)}

gs = GridSearchCV(RandomForestClassifier(random_state = 42),params, n_jobs = -1)
gs.fit(xtrainscale,ytrain)

print(gs.best_params_)
```

```
{'max_depth': 11, 'min_samples_split': 2, 'n_estimators': 50}
```

In [24]:
```
# 결과 출력
#최고 교차 검증 정수:
print("최고 교차 검증 정수:", gs.best_score_)
```

```
최고 교차 검증 정수: 0.9639731543624416
```

In [26]:
```
# 테스트 데이터로 평가
best_model = gs.best_estimator_
y_pred = best_model.predict(xtestscale)
print("테스트 정확도:", accuracy_score(ytest, y_pred))
```

```
테스트 정확도: 0.9627659574468085
```

파이프라인

In [27]:
```
#pipeline
pipe = Pipeline([('std', StandardScaler()), ('randomforest',RandomForestClassifier())])
pipe.fit(xtrain,ytrain)

ypred = pipe.predict(xtest)

print(accuracy_score(ytest,ypred))
```

```
0.9627659574468085
```