In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Data importing

In [4]:
```
oil_data = pd.read_csv('./oil_spill.csv')
oil_data
```

Out[4]:

	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	targe
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	2850.00	1000.00	763.16	135.46	3.73	0	33243.19	65.74	7.95	
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	
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	
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	
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
932	200	12	92.42	364.42	135	97200	59.42	10.34	884.0	0.17	...	381.84	254.56	84.85	146.97	4.50	0	2593.50	65.85	6.39	
933	201	11	98.82	248.64	159	89100	59.64	10.18	831.0	0.17	...	284.60	180.00	150.00	51.96	1.90	0	4361.25	65.70	6.53	
934	202	14	25.14	428.86	24	113400	60.14	17.94	847.0	0.30	...	402.49	180.00	180.00	0.00	2.24	0	2153.05	65.91	6.12	
935	203	10	96.00	451.30	68	81000	59.90	15.01	831.0	0.25	...	402.49	180.00	90.00	73.48	4.47	0	2421.43	65.97	6.32	
936	204	11	7.73	235.73	135	89100	61.82	12.24	831.0	0.20	...	254.56	254.56	127.28	180.00	2.00	0	3782.68	65.65	6.26	

937 rows × 50 columns

In [5]:
```
oil_data.iloc[:,22:]
```

0	0	0.47	132.78	-0.01	3.78	0.22	3.20	-3.71	-0.18	2.19	...	2850.00	1000.00	763.16	135.46	3.73	0	33243.19	65.74	7.95	1
1	0	0.58	132.78	-0.01	3.78	0.84	7.09	-2.21	0.00	0.00	...	5750.00	11500.00	9593.48	1648.80	0.60	0	51572.04	65.73	6.26	0
2	0	0.46	132.78	-0.01	3.78	0.70	4.79	-3.36	-0.23	1.95	...	1400.00	250.00	150.00	45.13	9.33	1	31692.84	65.81	7.84	1
3	0	0.48	132.78	-0.01	3.78	0.84	6.78	-3.54	-0.33	2.20	...	6041.52	761.58	453.21	144.97	13.33	1	37696.21	65.67	8.07	1
4	0	0.47	132.78	-0.01	3.78	0.02	2.28	-3.44	-0.44	2.19	...	1320.04	710.63	512.54	109.16	2.58	0	29038.17	65.66	7.35	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
932	0	1.06	221.97	0.87	5.07	0.95	3.03	-1.25	-0.53	1.10	...	381.84	254.56	84.85	146.97	4.50	0	2593.50	65.85	6.39	0
933	0	1.07	221.97	0.87	5.07	1.19	3.39	-1.70	-0.64	1.23	...	284.60	180.00	150.00	51.96	1.90	0	4361.25	65.70	6.53	0
934	0	1.08	221.97	0.87	5.07	1.60	5.08	-0.90	-0.55	1.09	...	402.49	180.00	180.00	0.00	2.24	0	2153.05	65.91	6.12	0
935	0	1.07	221.97	0.87	5.07	0.99	2.94	-1.31	-0.36	1.09	...	402.49	180.00	90.00	73.48	4.47	0	2421.43	65.97	6.32	0
936	0	1.11	221.97	0.87	5.07	1.20	3.03	-0.73	-0.50	1.23	...	254.56	254.56	127.28	180.00	2.00	0	3782.68	65.65	6.26	0

937 rows × 28 columns

2. Data Exploration

In [6]:
```
oil_data.shape
# output (rows no., columns no.)
```

Out[6]: (937, 50)

In [7]:
```
oil_data.describe()
```

Out[7]:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_41
count	937.000000	937.000000	937.000000	937.000000	937.000000	9.370000e+02	937.000000	937.000000	937.000000	937.000000	...	937.000000
mean	81.588047	332.842049	698.707086	870.992209	84.121665	7.696964e+05	43.242721	9.127887	3940.712914	0.221003	...	933.928677
std	64.976730	1931.938570	599.965577	522.799325	45.361771	3.831151e+06	12.718404	3.588878	8167.427625	0.090316	...	1001.681331
min	1.000000	10.000000	1.920000	1.000000	0.000000	7.031200e+04	21.240000	0.830000	667.000000	0.020000	...	0.000000
25%	31.000000	20.000000	85.270000	444.200000	54.000000	1.250000e+05	33.650000	6.750000	1371.000000	0.160000	...	450.000000
50%	64.000000	65.000000	704.370000	761.280000	73.000000	1.863000e+05	39.970000	8.200000	2090.000000	0.200000	...	685.420000
75%	124.000000	132.000000	1223.480000	1260.370000	117.000000	3.304680e+05	52.420000	10.760000	3435.000000	0.260000	...	1053.420000
max	352.000000	32389.000000	1893.080000	2724.570000	180.000000	7.131500e+07	82.640000	24.690000	160740.000000	0.740000	...	11949.330000

8 rows × 50 columns

In [8]:
```
oil_data.columns
```

Out[8]:
```
Index(['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
       'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19',
       'f_20', 'f_21', 'f_22', 'f_23', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28',
       'f_29', 'f_30', 'f_31', 'f_32', 'f_33', 'f_34', 'f_35', 'f_36', 'f_37',
       'f_38', 'f_39', 'f_40', 'f_41', 'f_42', 'f_43', 'f_44', 'f_45', 'f_46',
       'f_47', 'f_48', 'f_49', 'target'],
      dtype='object')
```

In [9]:
```
oil_data.target.value_counts()
```

Out[9]:
```
0    896
1     41
Name: target, dtype: int64
```

1. The graph above shows that the majority goes for the value of **Zero** which is **Non-Spill (negative case)**
2. And the minority goes for the value of **One** which is **Oil Spill (positive case)**

3. Data Visualization

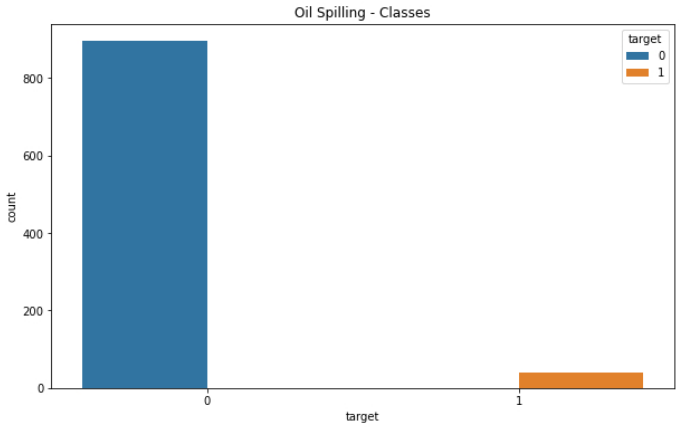
3.1 Target count plot

In [10]:
```
# Set the width and height of the figure
plt.figure(figsize=(10, 6))

# Add title
plt.title("Oil Spilling - Classes")

# Bar chart showing average arrival delay for Spirit Airlines flights by month
sns.countplot(x='target', hue='target', data=oil_data)
```

Out[10]:
```
<AxesSubplot:title={'center':'Oil Spilling - Classes'}, xlabel='target', ylabel='count'>
```



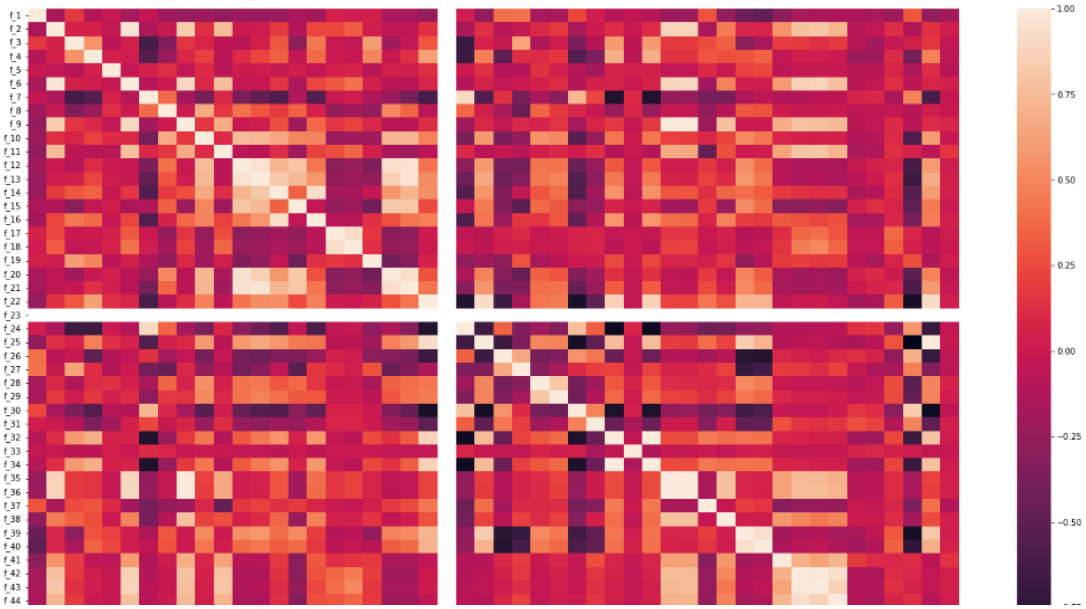
3.2 Find correlation

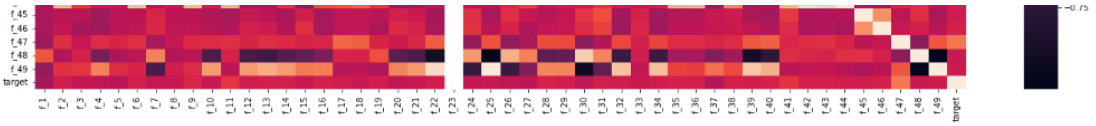
In [11]:
```
# Correlation between different variables
#
corr = oil_data.corr()
#
# Set up the matplotlib plot configuration
#
f, ax = plt.subplots(figsize=(25, 15))

# Draw the heatmap

sns.heatmap(corr)
```

Out[11]:
```
<AxesSubplot:>
```





the correlation graph above shows that `f_47` , `f_41` , and `f_11` have a positive correlation with the `target`

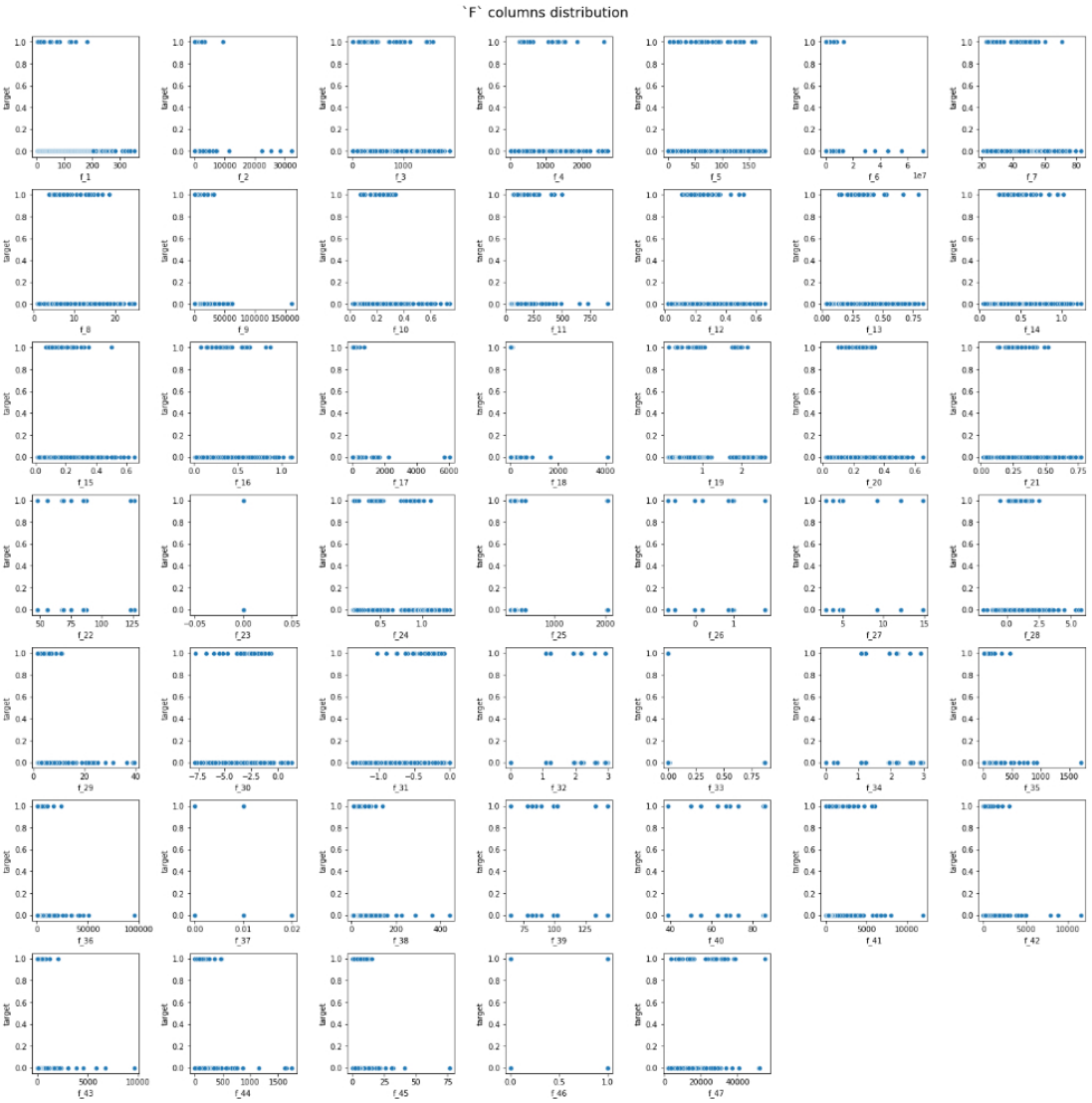
the graph also shows that `f_23` has the same value which I have to drop it since it'll not affect the data

In [12]:
```
oil_data.columns
```

Out[12]:
```
Index(['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
       'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19',
       'f_20', 'f_21', 'f_22', 'f_23', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28',
       'f_29', 'f_30', 'f_31', 'f_32', 'f_33', 'f_34', 'f_35', 'f_36', 'f_37',
       'f_38', 'f_39', 'f_40', 'f_41', 'f_42', 'f_43', 'f_44', 'f_45', 'f_46',
       'f_47', 'f_48', 'f_49', 'target'],
      dtype='object')
```

In [15]:
```
def visualize_scatterplot(df, cols,target):
    plt.figure(figsize=(20,20))
    plt.subplots_adjust(hspace=0.2)
    plt.suptitle("`F` columns distribution",
                 fontsize=18, y=1)
    for i, col in enumerate(cols):
        plt.subplot(7, 7, i+1)
        # plt.title(f'{col}')
        x = col
        sns.scatterplot(x=x, y=target, data=df)
    plt.tight_layout()
    plt.show()
```

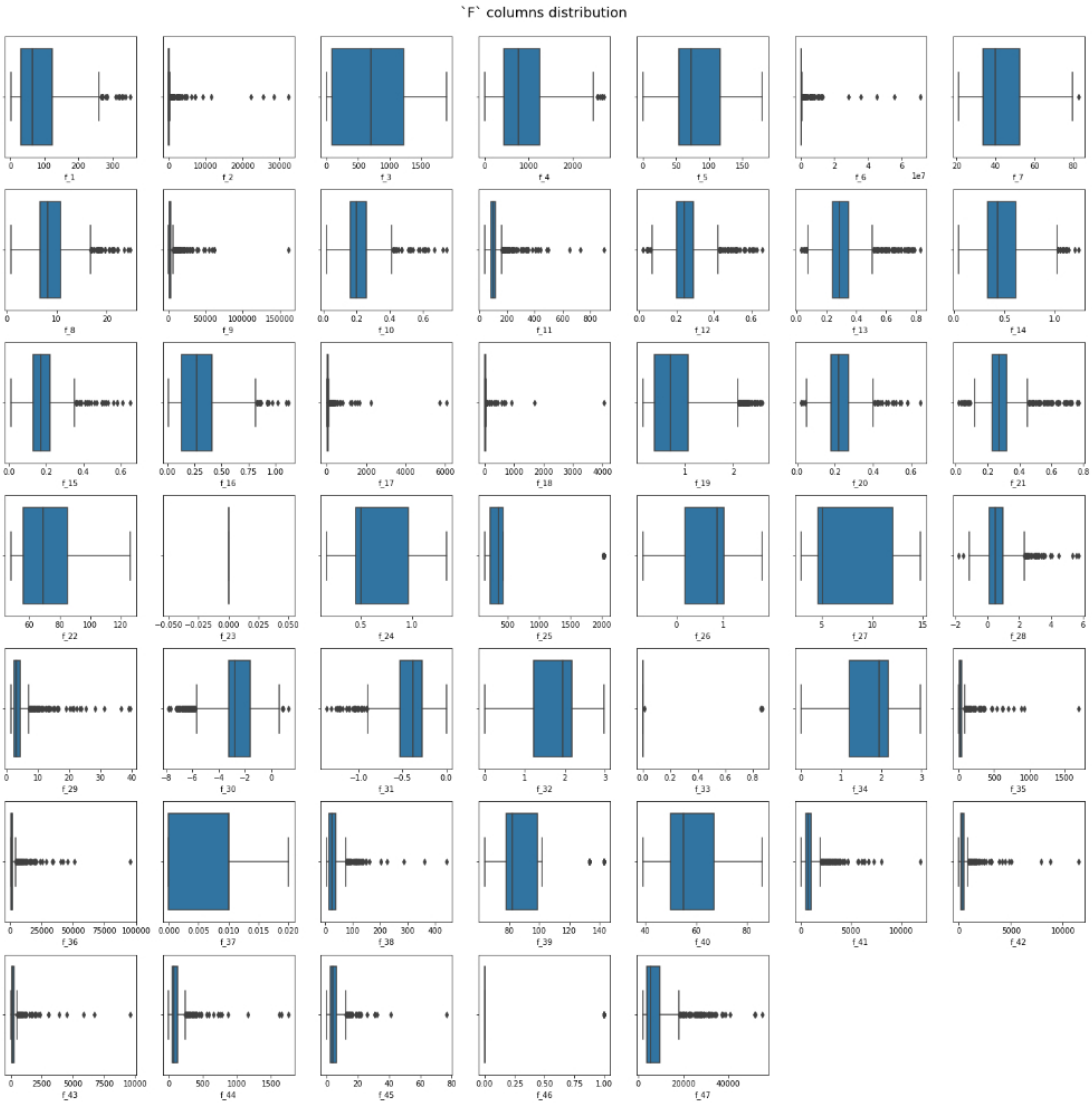
In [19]:
```
cols = oil_data.columns[:47]
visualize_scatterplot(oil_data, cols, oil_data['target'])
```



In [20]:
```
def visualize_boxplot(df, cols):
    plt.figure(figsize=(20, 20))
    plt.subplots_adjust(hspace=0.2)
    plt.suptitle("`F` columns distribution",
                 fontsize=18, y=1)
    for i, col in enumerate(cols):
```

```
        plt.subplot(7, 7, i+1)
        # plt.title(f'{col}')
        x = col
        sns.boxplot(x=x, data=df)
    plt.tight_layout()
    plt.show()
```

In [21]:
```
cols = oil_data.columns[:47]
visualize_boxplot(oil_data, cols)
```



- The graph above shows that the F columns

3. Data preprocessing

3.1 Assess missing values

In [22]:
```
oil_data.isnull().sum()
```

Out[22]:
```
f_1     0
f_2     0
f_3     0
f_4     0
f_5     0
f_6     0
f_7     0
f_8     0
f_9     0
f_10    0
f_11    0
f_12    0
f_13    0
f_14    0
f_15    0
f_16    0
f_17    0
f_18    0
f_19    0
f_20    0
f_21    0
f_22    0
```

```
f_23      0
f_24      0
f_25      0
f_26      0
f_27      0
f_28      0
f_29      0
f_30      0
f_31      0
f_32      0
f_33      0
f_34      0
f_35      0
f_36      0
f_37      0
f_38      0
f_39      0
f_40      0
f_41      0
f_42      0
f_43      0
f_44      0
f_45      0
f_46      0
f_47      0
f_48      0
f_49      0
target    0
dtype: int64
```

- As the above output shows there are no missing values

In [23]:
```
oil_data.columns
```

Out[23]:
```
Index(['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
       'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19',
       'f_20', 'f_21', 'f_22', 'f_23', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28',
       'f_29', 'f_30', 'f_31', 'f_32', 'f_33', 'f_34', 'f_35', 'f_36', 'f_37',
       'f_38', 'f_39', 'f_40', 'f_41', 'f_42', 'f_43', 'f_44', 'f_45', 'f_46',
       'f_47', 'f_48', 'f_49', 'target'],
      dtype='object')
```

3.3 Drop the column `f_23`

In [24]:
```
oil_data['f_23'].value_counts()
```

Out[24]:
```
0    937
Name: f_23, dtype: int64
```

In [25]:
```
oil_data.drop('f_23', axis=1, inplace=True)
```

- (0) Non-Spill: negative case, or majority class.
- (1) Oil Spill: positive case, or minority class.

In [26]:
```
oil_data.columns
```

Out[26]:
```
Index(['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
       'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19',
       'f_20', 'f_21', 'f_22', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28', 'f_29',
       'f_30', 'f_31', 'f_32', 'f_33', 'f_34', 'f_35', 'f_36', 'f_37', 'f_38',
       'f_39', 'f_40', 'f_41', 'f_42', 'f_43', 'f_44', 'f_45', 'f_46', 'f_47',
       'f_48', 'f_49', 'target'],
      dtype='object')
```

`f_23` is already dropped

TO DO: add scalling for the data

3.2 Splitting Data

In [27]:
```
y = oil_data.iloc[:,-1]
y
```

Out[27]:
```
0      1
1      0
2      1
3      1
4      0
      ..
932    0
933    0
934    0
935    0
936    0
Name: target, Length: 937, dtype: int64
```

In [28]:
```
X = oil_data.iloc[:,:-1]
X
```

Out[28]:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_40	f_41	f_42	f_43	f_44	f_45	f_46	f_47	f_48	f_49
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	69	2850.00	1000.00	763.16	135.46	3.73	0	33243.19	65.74	7.95
1	2	22325	79.11	841.03	180	55812500	51.11	1.21	61900.0	0.02	...	69	5750.00	11500.00	9593.48	1648.80	0.60	0	51572.04	65.73	6.26
2	3	115	1449.85	608.43	88	287500	40.42	7.34	3340.0	0.18	...	69	1400.00	250.00	150.00	45.13	9.33	1	31692.84	65.81	7.84

3	4	1201	1562.53	295.65	66	3002500	42.40	7.97	18030.0	0.19	...	69	6041.52	761.58	453.21	144.97	13.33	1	37696.21	65.67	8.07
4	5	312	950.27	440.86	37	780000	41.43	7.03	3350.0	0.17	...	69	1320.04	710.63	512.54	109.16	2.58	0	29038.17	65.66	7.35
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
932	200	12	92.42	364.42	135	97200	59.42	10.34	884.0	0.17	...	50	381.84	254.56	84.85	146.97	4.50	0	2593.50	65.85	6.39
933	201	11	98.82	248.64	159	89100	59.64	10.18	831.0	0.17	...	50	284.60	180.00	150.00	51.96	1.90	0	4361.25	65.70	6.53
934	202	14	25.14	428.86	24	113400	60.14	17.94	847.0	0.30	...	50	402.49	180.00	180.00	0.00	2.24	0	2153.05	65.91	6.12
935	203	10	96.00	451.30	68	81000	59.90	15.01	831.0	0.25	...	50	402.49	180.00	90.00	73.48	4.47	0	2421.43	65.97	6.32
936	204	11	7.73	235.73	135	89100	61.82	12.24	831.0	0.20	...	50	254.56	254.56	127.28	180.00	2.00	0	3782.68	65.65	6.26

937 rows × 48 columns

In [29]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_x = scaler.fit_transform(X)
```

In [30]:
```
scaled_x
```

Out[30]:
```
array([[-1.24092248,  1.15238979,  1.34643407, ...,  3.68676672,
         0.38873035, -0.05837721],
       [-1.22552414, 11.38954577, -1.03327258, ...,  6.36218073,
         0.38776948, -0.6396642 ],
       [-1.21012579, -0.11281849,  1.2526453 , ...,  3.46046621,
         0.39545643, -0.09621246],
       ...,
       [ 1.85414469, -0.1651255 , -1.12327578, ..., -0.8513792 ,
         0.40506512, -0.68781815],
       [ 1.86954303, -0.16719707, -1.00510593, ..., -0.81220448,
         0.41083033, -0.61902679],
       [ 1.88494138, -0.16667918, -1.15230961, ..., -0.6135064 ,
         0.38008253, -0.6396642 ]])
```

As the scaled dataframe above shown that the features `F` are scaled by `StandardScaler` which means scaling each feature to mean 0 and standard deviation 1.

In [31]:
```
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(
    scaled_x, y, random_state=0, test_size=0.25)
```

In [32]:
```
print(f'train_X shape = {train_X.shape}')
print(f'test_X shape = {test_X.shape}')
print(f'train_y shape = {train_y.shape}')
print(f'test_y shape = {test_y.shape}')
```

```
train_X shape = (702, 48)
test_X shape = (235, 48)
train_y shape = (702,)
test_y shape = (235,)
```

4. Fitting the model to the data

In [33]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
oil_spill_preds = forest_model.predict(test_X)
print('MAE',mean_absolute_error(test_y, oil_spill_preds))
```

```
MAE 0.05965957446808511
```

4.1 Solving the problem of Majority class

4.1.1 Oversampling the Minority class

In [34]:
```
oil_data.shape
```

Out[34]:
```
(937, 49)
```

In [35]:
```
import sys
!conda install --yes --prefix {sys.prefix} imbalanced-learn
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with flexible solve.

PackagesNotFoundError: The following packages are not available from current channels:

  - imbalanced-learn

Current channels:

  - https://repo.anaconda.com/pkgs/main/win-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/win-64
  - https://repo.anaconda.com/pkgs/r/noarch
  - https://repo.anaconda.com/pkgs/msys2/win-64
  - https://repo.anaconda.com/pkgs/msys2/noarch

To search for alternate channels that may provide the conda package you're
```

```
To search for alternate channels that may provide the conda package you're
looking for, navigate to

    https://anaconda.org

and use the search bar at the top of the page.
```

In [36]:
```
# pip install imbalanced-learn
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
print('Before oversampling the minority class, (y) shape =', Counter(y))
# define oversampling strategy
oversample = RandomOverSampler(
    sampling_strategy='not majority')  # or sampling_strategy=0.5
# fit and apply the transform
X_over, y_over = oversample.fit_resample(X, y)
# summarize class distribution
print('After oversampling the minority class, (y) shape =', Counter(y_over))
```

```
Before oversampling the minority class, (y) shape = Counter({0: 896, 1: 41})
After oversampling the minority class, (y) shape = Counter({1: 896, 0: 896})
```

4.1.2 Spliting the new oversampled `target` column to fit the model

In [37]:
```
train_X, test_X, train_y, test_y = train_test_split(
    X_over, y_over, random_state=0, test_size=0.25)
forest_model.fit(train_X, train_y)
oil_spill_preds = forest_model.predict(test_X)
print('MAE',mean_absolute_error(test_y, oil_spill_preds))
```

```
MAE 0.025848214285714287
```

4.1.3 Try using another strategy for oversampling like haveing the half of majority class

In [38]:
```
print('Before oversampling the minority class, (y) shape =', Counter(y))
# define oversampling strategy
oversample = RandomOverSampler(
    sampling_strategy=0.5)
# fit and apply the transform
X_over, y_over = oversample.fit_resample(X, y)
# summarize class distribution
print('After oversampling the minority class, (y) shape =', Counter(y_over))
```

```
Before oversampling the minority class, (y) shape = Counter({0: 896, 1: 41})
After oversampling the minority class, (y) shape = Counter({0: 896, 1: 448})
```

4.1.4 Spliting the new oversampled `target` column to fit the model

In [39]:
```
train_X, test_X, train_y, test_y = train_test_split(
    X_over, y_over, random_state=0, test_size=0.25)
forest_model.fit(train_X, train_y)
oil_spill_preds = forest_model.predict(test_X)
print('MAE',mean_absolute_error(test_y, oil_spill_preds))
```

```
MAE 0.029851190476190476
```

- As it is shown above that the final result of MAE which equals 0.029 is the best values, so I will take in my consideration the oversampling using the ratio of 0.5 the majority class