There are two classes and the goal is to distinguish between
spill and non-spill using the features for a given ocean patch.
• Non-Spill: negative case, or majority class.
• Oil Spill: positive case, or minority class.
There are a total of 50 Columns in the Dataset , the output column is named as target.

Q.1. Download the Oil Spill Dataset and perform Data cleaning and Data Pre-Processing if Necessary

In [1]:
```
#Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
# Loading the dataset
oil_df=pd.read_csv('oil_spill.csv')
oil_df.head()
```

Out[2]:

	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
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	2850.00	1000.00	763.16	135
1	2	22325	79.11	841.03	180	55812500	51.11	1.21	61900.0	0.02	...	5750.00	11500.00	9593.48	1648
2	3	115	1449.85	608.43	88	287500	40.42	7.34	3340.0	0.18	...	1400.00	250.00	150.00	45
3	4	1201	1562.53	295.65	66	3002500	42.40	7.97	18030.0	0.19	...	6041.52	761.58	453.21	144
4	5	312	950.27	440.86	37	780000	41.43	7.03	3350.0	0.17	...	1320.04	710.63	512.54	109

5 rows × 50 columns

In [3]:
```
# Perform data cleaning and preprocessing
# Example: Remove missing values
nv= oil_df.isnull().sum()
nv= nv[nv>0]
nv
```

Out[3]:
```
Series([], dtype: int64)
```

In [4]:
```
oil_df.dropna(inplace=True)
```

In [5]:
```
oil_df.duplicated().sum()
```

Out[5]:
```
0
```

In [5]:
```
oil_df.head()
```

Out[5]:

	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
0	1	2558	1506.09	456.63	90	6395000	40.88	7.89	29780.0	0.19	...	2850.00	1000.00	763.16	135
1	2	22325	79.11	841.03	180	55812500	51.11	1.21	61900.0	0.02	...	5750.00	11500.00	9593.48	1648
2	3	115	1449.85	608.43	88	287500	40.42	7.34	3340.0	0.18	...	1400.00	250.00	150.00	45
3	4	1201	1562.53	295.65	66	3002500	42.40	7.97	18030.0	0.19	...	6041.52	761.58	453.21	144
4	5	312	950.27	440.86	37	780000	41.43	7.03	3350.0	0.17	...	1320.04	710.63	512.54	109

5 rows × 50 columns

EDA

In [ ]:
```
oil_df.columns
```

In [6]:
```
df=oil_df
```

In [7]:
```
#Separatiing the categorical and continuous columns
def Cat_Cont(df):
    cat_cols,cont_cols=[],[]
    for col in df.columns[:-1]:#we are exluding the target variable
        count=df[col].nunique()
        if count < 10:
            cat_cols.append(col)
        else:
            cont_cols.append(col)
    return (cat_cols,cont_cols)
```

In [8]:
```
cat_col, cont_col = Cat_Cont(df)
print(f"Categorical Columns length is {len(cat_col)} \n :{cat_col}")
print(f"Continuous Columns length is {len(cont_col)} \n {cont_col}")
```

```
Categorical Columns length is 10
 :['f_22', 'f_23', 'f_25', 'f_26', 'f_27', 'f_33', 'f_37', 'f_39', 'f_40', 'f_46']
Continuous Columns length is 39
 ['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_24', 'f_28',
'f_29', 'f_30', 'f_31', 'f_32', 'f_34', 'f_35', 'f_36', 'f_38', 'f_41', 'f_42', 'f_43',
'f_44', 'f_45', 'f_47', 'f_48', 'f_49']
```

In [9]:
```
x=['f_1','f_23','f_33','f_37']
for i in x:
    print(f"{i} columns with unique values : \n {df[i].value_counts()}")
    print("********************************* \n")
```

```
f_1 columns with unique values :
 3      9
5       9
6       9
8       9
9       9
       ..
261     1
266     1
267     1
269     1
204     1
Name: f_1, Length: 238, dtype: int64
*********************************

f_23 columns with unique values :
 0    937
Name: f_23, dtype: int64
*********************************

f_33 columns with unique values :
 0.00    933
0.87      2
0.01      1
0.86      1
Name: f_33, dtype: int64
*********************************

f_37 columns with unique values :
 0.01    580
0.00    324
0.02     33
Name: f_37, dtype: int64
*********************************
```

In [10]:
```
df.drop(['f_1','f_23','f_33','f_37'],axis=1,inplace=True)
df.shape
```

Out[10]:
```
(937, 46)
```

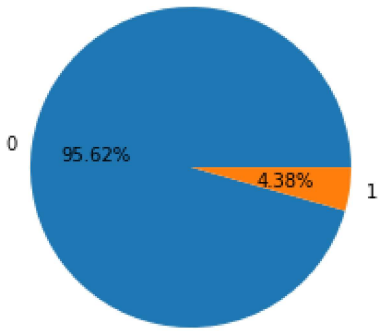
In [11]:
```
#separting categorical and continuos once again
cat_col, cont_col = Cat_Cont(df)
```

In [12]:
```
cat_col, cont_col = Cat_Cont(df)
print(f"Categorical Columns length is {len(cat_col)} \n :{cat_col}")
print(f"Continuous Columns length is {len(cont_col)} \n {cont_col}")
```

```
Categorical Columns length is 7
 :['f_22', 'f_25', 'f_26', 'f_27', 'f_39', 'f_40', 'f_46']
Continuous Columns length is 38
 ['f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10', 'f_11', 'f_12', 'f_1
3', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19', 'f_20', 'f_21', 'f_24', 'f_28', 'f_2
9', 'f_30', 'f_31', 'f_32', 'f_34', 'f_35', 'f_36', 'f_38', 'f_41', 'f_42', 'f_43', 'f_4
4', 'f_45', 'f_47', 'f_48', 'f_49']
```

In [13]:
```
r=df['target'].value_counts()
print(r)
plt.pie(r,labels=r.index,autopct='%.2f%%')
plt.show()
```

```
0    896
1     41
Name: target, dtype: int64
```



In [ ]:
```
95.62% zero as output
4.38% consist as 1 output
Majority of data consist of 0 as output
```

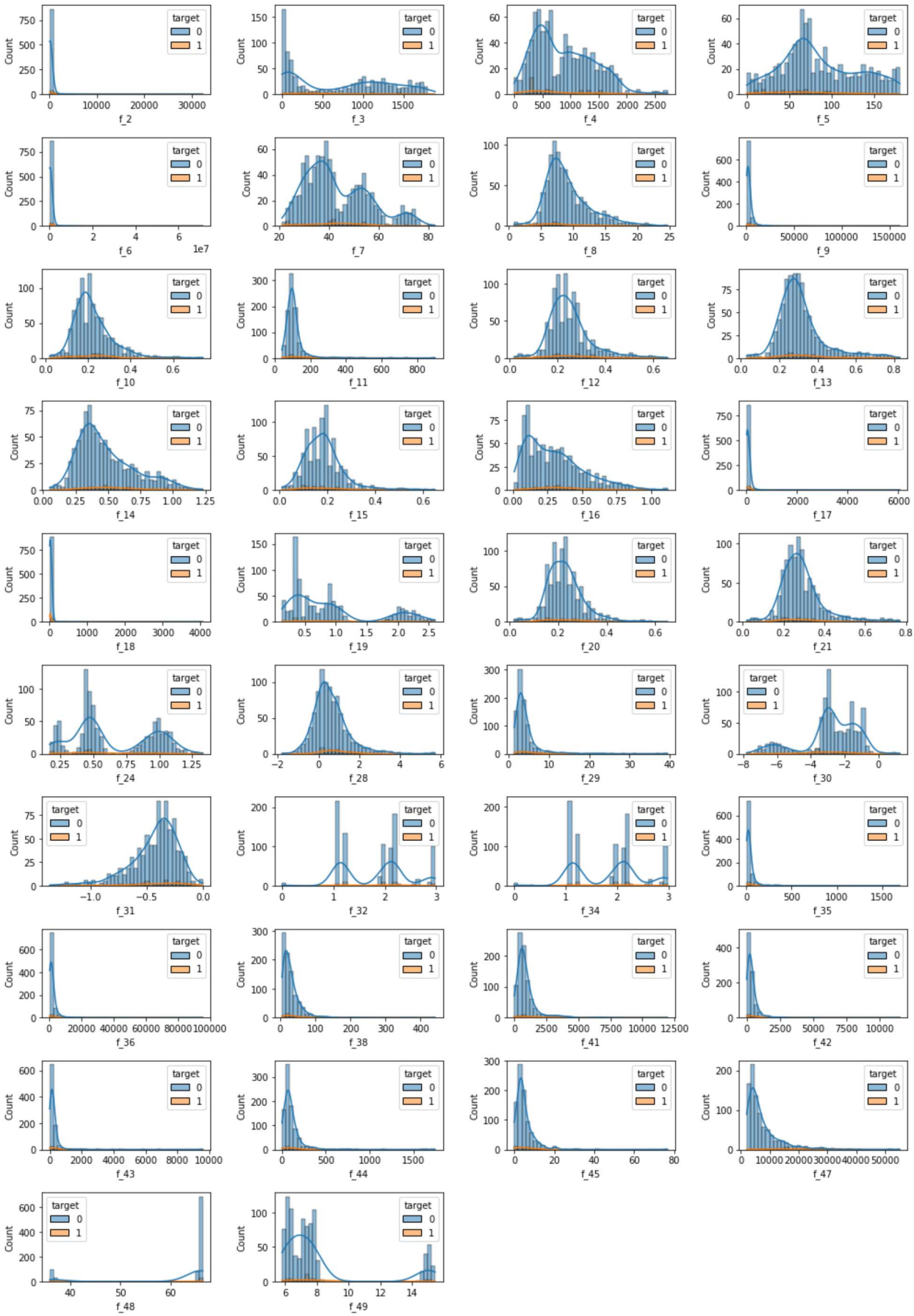
In [14]:
```
plt.figure(figsize=(22, 22))
for i in range(len(cat_col)):
    plt.subplot(4, 3, i + 1)
    ct = pd.crosstab(index=df[cat_col[i]], columns=df['target'])
    ct.plot(kind='bar', stacked=True, ax=plt.gca())
    plt.title(f'Stacked Bar Chart for {cat_col[i]}')
    plt.xlabel(cat_col[i])
    plt.ylabel('Count')
    plt.legend(title='target', labels=['0', '1'])
plt.tight_layout()
plt.show()
```

In [ ]:
```
All categorical features are significantly helps in deciding negative and positive spills
```

In [15]:
```
df2=df.copy()
df_cat=df2[cat_col]
df_cat['out_put']=df2.iloc[:,-1]
```

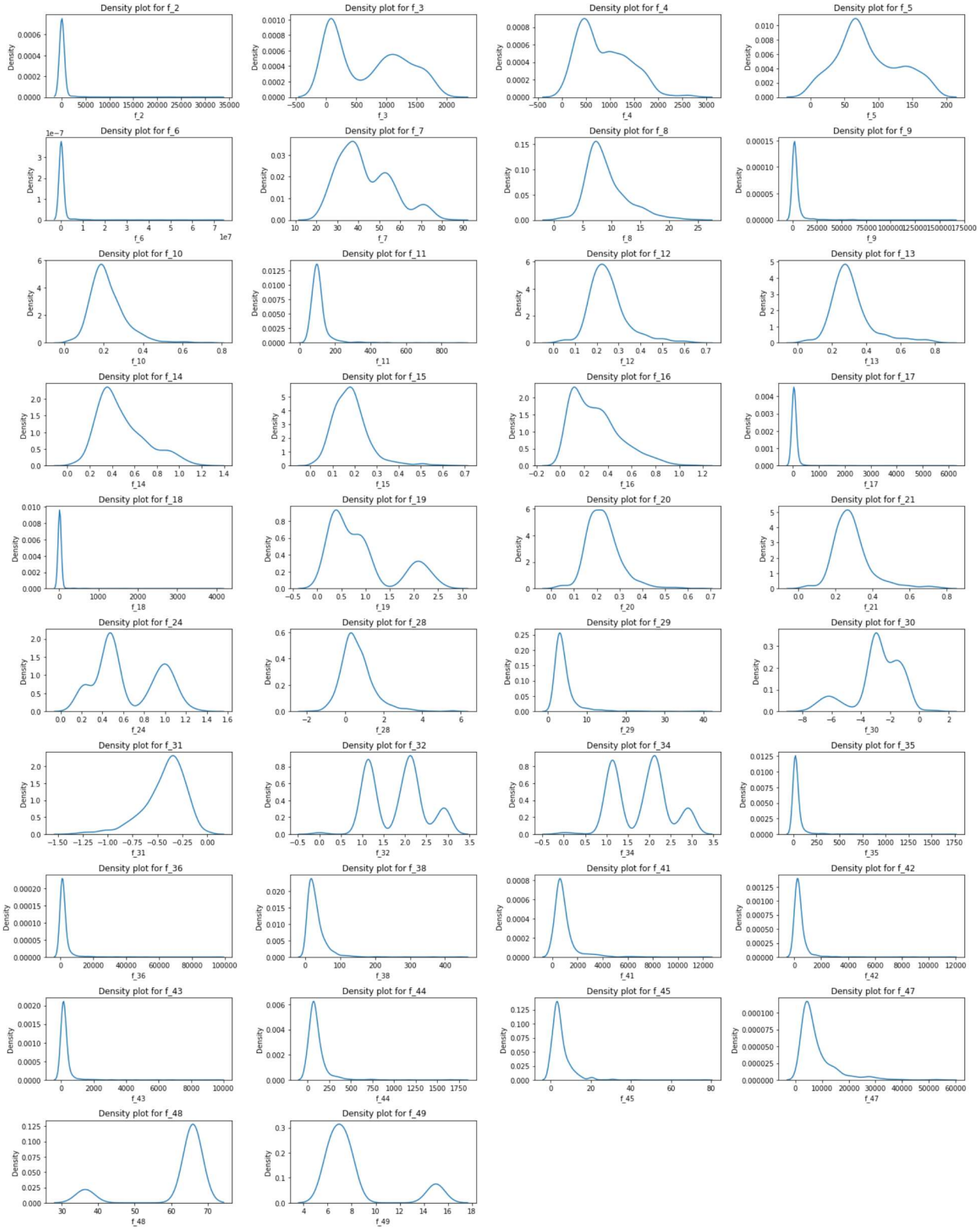
```
C:\Users\USER\AppData\Local\Temp/ipykernel_1976/1590522172.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_cat['out_put']=df2.iloc[:,-1]
```

In [17]:
```
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(14,20))
for i,col in enumerate(cont_col):
    plt.subplot(10,4,i+1)
    sns.histplot(df,x=cont_col[i],bins=40,kde=True,hue="target")
plt.tight_layout()
plt.show()
```


target
0
1
Count
f_2
f_3
f_4
f_5
f_6
1e7
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_24
f_28
f_29
f_30
f_31
f_32
f_34
f_35
f_36
f_38
f_41
f_42
f_43
f_44
f_45
f_47
f_48
f_49

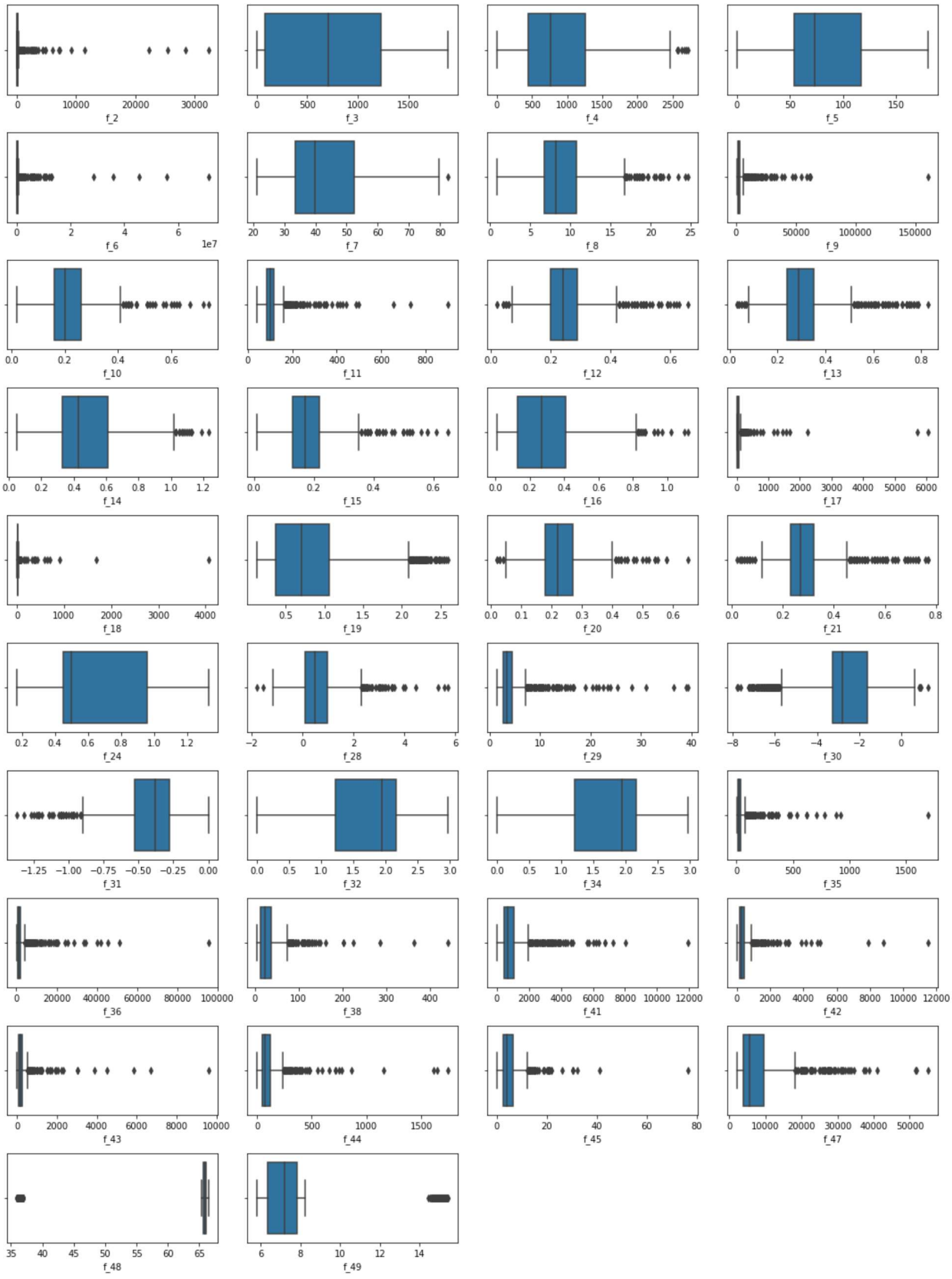
In [ ]:
```
Most of the continuous categorical column do not follow normal distribution.
Almost all the columns have subjected to outliers.
```

In [18]:
```
plt.figure(figsize=(20,25))
for i,col in enumerate(cont_col):
    plt.subplot(10,4,i+1)
    sns.kdeplot(df[col])
    plt.title(f' Density plot for {col}')
plt.tight_layout()
plt.savefig('KDE plot for Continuos Columns')
plt.show()
```



In [ ]:
```
Skewness is positive in majority of columns
```

In [19]:
```
plt.figure(figsize=(15,20))
for i in range(len(cont_col)):
    plt.subplot(10,4,i+1)
    sns.boxplot(df,x=cont_col[i])
plt.tight_layout()
plt.show()
```



In [20]:
```
df[cont_col].describe()
```

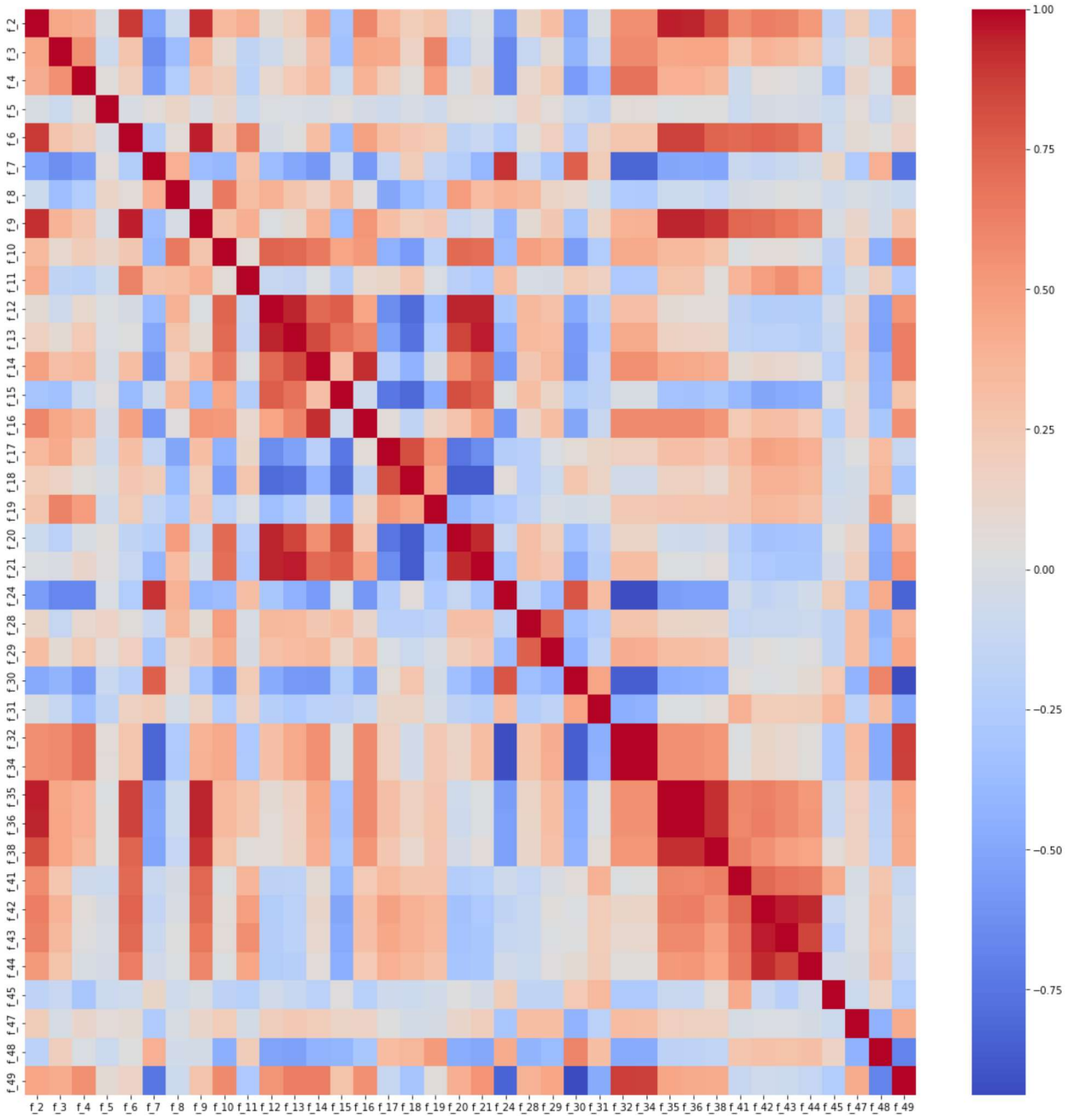
Out[20]:

	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f
count	937.000000	937.000000	937.000000	937.000000	9.370000e+02	937.000000	937.000000	937.0000
mean	332.842049	698.707086	870.992209	84.121665	7.696964e+05	43.242721	9.127887	3940.7129
std	1931.938570	599.965577	522.799325	45.361771	3.831151e+06	12.718404	3.588878	8167.4276
min	10.000000	1.920000	1.000000	0.000000	7.031200e+04	21.240000	0.830000	667.0000
25%	20.000000	85.270000	444.200000	54.000000	1.250000e+05	33.650000	6.750000	1371.0000
50%	65.000000	704.370000	761.280000	73.000000	1.863000e+05	39.970000	8.200000	2090.0000
75%	132.000000	1223.480000	1260.370000	117.000000	3.304680e+05	52.420000	10.760000	3435.0000
max	32389.000000	1893.080000	2724.570000	180.000000	7.131500e+07	82.640000	24.690000	160740.0000

8 rows × 38 columns

In [21]:
```
for i in df[cont_col]:
    q1=df[cont_col].quantile(0.25)
    q3=df[cont_col].quantile(0.75)
    iqr=q3-q1
    bmin=q1-1.5*iqr
    bmax=q3+1.5*iqr
    df[cont_col]=np.where(df[cont_col]>bmax,bmax,df[cont_col])
    df[cont_col]=np.where(df[cont_col]<bmin,bmin,df[cont_col])
```

In [22]:
```
plt.figure(figsize=(20,20))
corr=df[cont_col].corr()

sns.heatmap(corr,cmap='coolwarm')
plt.show()
```



Q2)Use various methods such as Handling null values, One-Hot Encoding, Imputation, and Scaling of Data Pre-Processing where necessary.

In [7]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

## Handling null values

# Option 1: Drop rows with missing values
oil_df.dropna(inplace=True)

# Option 2: Impute missing values
imputer = SimpleImputer(strategy='mean') # You can choose different imputation strategies
imputed_df = imputer.fit_transform(oil_df)

# One-Hot Encoding categorical data
encoder = OneHotEncoder(sparse_output=False, drop='first') # Drop first column to avoid mu
encoded_df = pd.DataFrame(encoder.fit_transform(oil_df[['target']]))
df = pd.concat([oil_df, encoded_df], axis=1)
df.drop(['target'], axis=1, inplace=True)

# Scaling of numerical features
scaler = StandardScaler()
scaler_df = scaler.fit_transform(encoded_df)
```

or

In [8]:
```
# Perform feature scaling
scaler = StandardScaler()
oil_df[['f_1', 'f_2']] = scaler.fit_transform(oil_df[['f_1', 'f_2']])
oil_df[['f_1', 'f_2']]
```

Out[8]:

	f_1	f_2
0	-1.240922	1.152390
1	-1.225524	11.389546
2	-1.210126	-0.112818
3	-1.194727	0.449611
4	-1.179329	-0.010794
...	...	...
932	1.823348	-0.166161
933	1.838746	-0.166679
934	1.854145	-0.165126
935	1.869543	-0.167197
936	1.884941	-0.166679

937 rows × 2 columns

In [9]:
```
oil_df.describe()
```

Out[9]:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f
count	9.370000e+02	9.370000e+02	937.000000	937.000000	937.000000	9.370000e+02	937.000000	937.0000
mean	-1.516633e-17	3.033267e-17	698.707086	870.992209	84.121665	7.696964e+05	43.242721	9.1278
std	1.000534e+00	1.000534e+00	599.965577	522.799325	45.361771	3.831151e+06	12.718404	3.5888
min	-1.240922e+00	-1.671971e-01	1.920000	1.000000	0.000000	7.031200e+04	21.240000	0.8300
25%	-7.789722e-01	-1.620182e-01	85.270000	444.200000	54.000000	1.250000e+05	33.650000	6.7500
50%	-2.708268e-01	-1.387130e-01	704.370000	761.280000	73.000000	1.863000e+05	39.970000	8.2000
75%	6.530738e-01	-1.040143e-01	1223.480000	1260.370000	117.000000	3.304680e+05	52.420000	10.7600
max	4.163896e+00	1.660160e+01	1893.080000	2724.570000	180.000000	7.131500e+07	82.640000	24.6900

8 rows × 50 columns

Q.3. Derive some insights from the dataset.

In [10]:
```
import pandas as pd
# Read the dataset
df = pd.read_csv('oil_spill.csv')
average_values = df.mean(axis=0)
print(average_values)
correlation_coefficients = df.corr()
# Print the correlation coefficients
print(correlation_coefficients)
```

```
f_1           81.588047
f_2          332.842049
f_3          698.707086
f_4          870.992209
f_5           84.121665
f_6       769696.378869
f_7           43.242721
f_8            9.127887
f_9         3940.712914
f_10           0.221003
f_11         109.892743
f_12           0.251441
f_13           0.311217
f_14           0.484664
f_15           0.182892
f_16           0.301932
f_17          77.412561
f_18          31.151921
f_19           0.909039
```

In [11]:

```
dataset = pd.read_csv("oil_spill.csv")
# Explore the dataset
#  1: Get the dimensions of the dataset (number of rows and columns)
num_rows, num_cols = dataset.shape
print("Number of rows:", num_rows)
print("Number of columns:", num_cols)
#   2: Check the summary statistics of numeric columns
numeric_cols = dataset.select_dtypes(include='number').columns
numeric_summary = dataset[numeric_cols].describe()
print("Summary statistics of numeric columns:")
print(numeric_summary)
#   3: Calculate the correlation between features and the target variable
correlation = dataset.corr()['target']
print("Correlation with target variable:")
print(correlation)
#   4: Count the occurrences of each category in a categorical column
categorical_col = 'target'
category_counts = dataset[categorical_col].value_counts()
print("Category counts:")
print(category_counts)
#   5: Visualize the distribution of a numeric column
import matplotlib.pyplot as plt
plt.hist(dataset, bins=20)
plt.xlabel('Numeric Column')
plt.ylabel('Count')
plt.title('Distribution of Numeric Column')
plt.show()
# 6: Explore relationships between features using scatter plots or heatmaps
import seaborn as sns
sns.scatterplot(x='f_1', y='f_2', hue='target', data=dataset)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Scatter plot of Feature 1 vs. Feature 2')
plt.show()
#  7: Grouping and aggregation operations
grouped_data = dataset.groupby('target')['f_1'].mean()
print("Mean of numeric column by category:")
print(grouped_data)
#  8: Identify outliers in numeric columns
numeric_cols = dataset.select_dtypes(include='number').columns
for col in numeric_cols:
 q1 = dataset[col].quantile(0.25)
 q3 = dataset[col].quantile(0.75)
 iqr = q3 - q1
 lower_bound = q1 - 1.5 * iqr
 upper_bound = q3 + 1.5 * iqr
 outliers = dataset[(dataset[col] < lower_bound) | (dataset[col] > upper_bound)]
 print("Outliers in", col)
 print(outliers)
```

```
Number of rows: 937
Number of columns: 50
Summary statistics of numeric columns:
              f_1           f_2          f_3          f_4          f_5  \
count  937.000000    937.000000   937.000000   937.000000  937.000000
mean    81.588047    332.842049   698.707086   870.992209   84.121665
std     64.976730   1931.938570   599.965577   522.799325   45.361771
min      1.000000     10.000000     1.920000     1.000000    0.000000
25%     31.000000     20.000000    85.270000   444.200000   54.000000
50%     64.000000     65.000000   704.370000   761.280000   73.000000
75%    124.000000    132.000000  1223.480000  1260.370000  117.000000
max    352.000000  32389.000000  1893.080000  2724.570000  180.000000

                f_6         f_7         f_8           f_9         f_10  ...  \
count  9.370000e+02  937.000000  937.000000    937.000000  937.000000  ...
mean   7.696964e+05   43.242721    9.127887   3940.712914    0.221003  ...
std    3.831151e+06   12.718404    3.588878   8167.427625    0.090316  ...
min    7.031200e+04   21.240000    0.830000    667.000000    0.020000  ...
25%    1.250000e+05   33.650000    6.750000   1371.000000    0.160000  ...
```

In [12]:
```
 # Split the dataset into features (X) and target (y)
X = oil_df.drop('target', axis=1)
y = oil_df['target']
print("X",X)
print("y",y)

 #Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(oil_df, oil_df["target"],test_size=0.
print("X_train", X_train)
print( "y_train", y_train)
print("X_test", X_test)
print("y_test", y_test)

# Train a random forest classifier
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
# Make predictions on the test set
y_pred = classifier.predict(X_test)
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
X          f_1        f_2      f_3     f_4  f_5       f_6    f_7    f_8  \
0   -1.240922   1.152390  1506.09  456.63   90   6395000  40.88   7.89
1   -1.225524  11.389546    79.11  841.03  180  55812500  51.11   1.21
2   -1.210126  -0.112818  1449.85  608.43   88    287500  40.42   7.34
3   -1.194727   0.449611  1562.53  295.65   66   3002500  42.40   7.97
4   -1.179329  -0.010794   950.27  440.86   37    780000  41.43   7.03
..        ...        ...      ...     ...  ...       ...    ...    ...
932  1.823348  -0.166161    92.42  364.42  135     97200  59.42  10.34
933  1.838746  -0.166679    98.82  248.64  159     89100  59.64  10.18
934  1.854145  -0.165126    25.14  428.86   24    113400  60.14  17.94
935  1.869543  -0.167197    96.00  451.30   68     81000  59.90  15.01
936  1.884941  -0.166679     7.73  235.73  135     89100  61.82  12.24

         f_9  f_10  ...  f_40     f_41      f_42     f_43     f_44   f_45  \
0    29780.0  0.19  ...    69  2850.00   1000.00   763.16   135.46   3.73
1    61900.0  0.02  ...    69  5750.00  11500.00  9593.48  1648.80   0.60
2     3340.0  0.18  ...    69  1400.00    250.00   150.00    45.13   9.33
3    18030.0  0.19  ...    69  6041.52    761.58   453.21   144.97  13.33
4     3350.0  0.17  ...    69  1320.04    710.63   512.54   109.16   2.58
```

Q.4. Apply various Machine Learning techniques to predict the output in target column, make use of Bagging and Ensemble as required and find the best model by evaluating the model using Model evaluation techniques

In [13]:

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
dataset = pd.read_csv('oil_spill.csv')

# Split the dataset into features (X) and target (y)
X = dataset.drop('target', axis=1)
y = dataset['target']

# 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)

# Train and evaluate a random forest classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Classifier Accuracy:", rf_accuracy)
print("Random Forest Classifier Report:")
print(classification_report(y_test, rf_predictions))

# Train and evaluate a gradient boosting classifier
gb_classifier = GradientBoostingClassifier()
gb_classifier.fit(X_train, y_train)
gb_predictions = gb_classifier.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_predictions)
print("Gradient Boosting Classifier Accuracy:", gb_accuracy)
print("Gradient Boosting Classifier Report:")
print(classification_report(y_test, gb_predictions))

# Apply bagging or ensemble techniques
# Example 1: Bagging with Random Forest Classifier
bagging_classifier = RandomForestClassifier()
bagging_classifier.fit(X_train, y_train)
bagging_predictions = bagging_classifier.predict(X_test)
bagging_accuracy = accuracy_score(y_test, bagging_predictions)
print("Bagging (Random Forest) Classifier Accuracy:", bagging_accuracy)
print("Bagging (Random Forest) Classifier Report:")
print(classification_report(y_test, bagging_predictions))

# Example 2: Ensemble with Voting Classifier
from sklearn.ensemble import VotingClassifier
ensemble_classifier = VotingClassifier(estimators=[('rf', rf_classifier), ('gb', gb_class
ensemble_classifier.fit(X_train, y_train)
ensemble_predictions = ensemble_classifier.predict(X_test)
ensemble_accuracy = accuracy_score(y_test, ensemble_predictions)
print("Ensemble (Voting) Classifier Accuracy:", ensemble_accuracy)
print("Ensemble (Voting) Classifier Report:")
print(classification_report(y_test, ensemble_predictions))

# Perform cross-validation for model evaluation
cv_scores_rf = cross_val_score(rf_classifier, X, y, cv=5)
cv_scores_gb = cross_val_score(gb_classifier, X, y, cv=5)
cv_scores_bagging = cross_val_score(bagging_classifier, X, y, cv=5)
cv_scores_ensemble = cross_val_score(ensemble_classifier, X, y, cv=5)
print("Cross-validation scores:")
print("Random Forest Classifier:", cv_scores_rf.mean())
print("Gradient Boosting Classifier:", cv_scores_gb.mean())
print("Bagging (Random Forest) Classifier:", cv_scores_bagging.mean())
```

```
print("Ensemble (Voting) Classifier:", cv_scores_ensemble.mean())
```

```
Random Forest Classifier Accuracy: 0.973404255319149
Random Forest Classifier Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       182
           1       0.67      0.33      0.44         6

    accuracy                           0.97       188
   macro avg       0.82      0.66      0.72       188
weighted avg       0.97      0.97      0.97       188

Gradient Boosting Classifier Accuracy: 0.9787234042553191
Gradient Boosting Classifier Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       182
           1       0.62      0.83      0.71         6

    accuracy                           0.98       188
   macro avg       0.81      0.91      0.85       188
weighted avg       0.98      0.98      0.98       188

Bagging (Random Forest) Classifier Accuracy: 0.9787234042553191
Bagging (Random Forest) Classifier Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       182
           1       1.00      0.33      0.50         6

    accuracy                           0.98       188
   macro avg       0.99      0.67      0.74       188
weighted avg       0.98      0.98      0.97       188

Ensemble (Voting) Classifier Accuracy: 0.9840425531914894
Ensemble (Voting) Classifier Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       182
           1       1.00      0.50      0.67         6

    accuracy                           0.98       188
   macro avg       0.99      0.75      0.83       188
weighted avg       0.98      0.98      0.98       188

Cross-validation scores:
Random Forest Classifier: 0.9402321083172147
Gradient Boosting Classifier: 0.9177437706223689
Bagging (Random Forest) Classifier: 0.9445044942541815
Ensemble (Voting) Classifier: 0.9380930708840596
```

#Q.5. Save the best model and Load the model

In [14]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from joblib import dump, load
# Load the dataset
dataset = pd.read_csv('oil_spill.csv')
# Split the dataset into features (X) and target (y)
X = dataset.drop('target', axis=1)
y = dataset['target']

# 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)

# Train a random forest classifier
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)

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

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Save the model
best_model_filename = 'best_model.joblib'
dump(classifier, best_model_filename)
print("Best model saved as", best_model_filename)

# Load the model
loaded_model = load(best_model_filename)

# Make predictions using the loaded model
new_predictions = loaded_model.predict(X_test)

# Evaluate the loaded model
loaded_accuracy = accuracy_score(y_test, new_predictions)
print("Accuracy of the loaded model:", loaded_accuracy)
```

```
Accuracy: 0.973404255319149
Best model saved as best_model.joblib
Accuracy of the loaded model: 0.973404255319149
```

Q.6. Take the original data set and make another dataset by randomly picking 20 data points from the oil spill dataset and apply the saved model on the same

In [15]:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from joblib import load
# Load the original dataset
original_dataset = pd.read_csv('oil_spill.csv')
# Randomly select 20 data points
new_dataset = original_dataset.sample(n=20, random_state=42)
# Load the saved model
loaded_model = load('best_model.joblib')
# Separate the features (X) and target (y) in the new dataset
X_new = new_dataset.drop('target', axis=1)
y_new = new_dataset['target']
# Apply the saved model on the new dataset
predictions = loaded_model.predict(X_new)
# Display the new dataset with predicted labels
new_dataset_with_predictions = new_dataset.copy()
new_dataset_with_predictions['predicted_target'] = predictions
print(new_dataset_with_predictions)
```

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	\
321	29	105	881.92	1128.79	83	262500	38.90	8.51	2710.0	0.22	
70	60	111	1153.32	1283.44	41	277500	41.25	5.98	1760.0	0.14	
209	17	867	1059.49	581.31	46	2167500	31.08	8.26	15780.0	0.27	
656	9	85	71.06	469.47	140	688500	70.85	11.28	4626.0	0.16	
685	38	15	32.47	582.13	156	121500	73.27	12.11	1080.0	0.17	
96	86	86	769.73	1761.26	55	215000	37.55	6.27	3090.0	0.17	
468	36	462	904.13	2689.99	129	649687	29.80	8.99	5160.0	0.30	
86	76	128	1378.47	929.73	51	320000	39.80	5.20	3370.0	0.13	
532	38	294	11.49	1559.36	40	413437	38.12	22.22	2893.5	0.58	
327	37	98	1326.06	1109.08	72	245000	41.31	7.53	2880.0	0.18	
528	34	151	465.77	1736.15	73	212343	28.96	8.14	3474.0	0.28	
247	138	144	1341.72	78.22	110	360000	31.12	6.88	4650.0	0.22	
250	156	260	1080.89	833.29	111	650000	30.52	7.95	5680.0	0.26	
485	53	84	575.19	1558.81	153	118125	30.94	8.89	1489.5	0.29	
467	35	74	619.18	1622.32	5	104062	26.45	5.92	1255.5	0.22	
723	76	10	30.80	348.90	153	81000	70.50	8.93	720.0	0.13	
483	51	60	743.88	1250.60	127	84375	33.03	11.87	1701.5	0.36	
886	154	10	182.50	460.00	90	81000	57.60	8.68	810.0	0.15	
809	77	13	160.77	420.23	63	105300	51.15	10.66	1191.0	0.21	
244	118	308	1313.18	791.35	61	770000	29.13	7.14	5880.0	0.24	

	...	f_42	f_43	f_44	f_45	f_46	f_47	f_48	f_49	\
321	...	353.55	226.91	84.74	4.21	0	3425.75	65.97	7.04	
70	...	500.00	296.40	140.92	2.40	0	5915.80	66.12	7.34	
209	...	1131.37	637.97	408.01	4.93	0	5679.31	65.74	7.42	
656	...	509.12	323.98	87.51	3.95	0	6376.53	65.98	6.22	
685	...	201.25	105.89	84.66	6.47	0	3285.95	66.11	5.98	
96	...	180.28	93.84	59.34	14.93	1	15720.91	66.30	6.71	
468	...	0.00	0.00	0.00	0.00	0	40916.70	36.71	14.53	
86	...	320.16	160.29	94.32	8.43	0	9183.53	65.98	7.73	
532	...	0.00	0.00	0.00	0.00	0	10484.87	36.02	14.82	
327	...	269.26	196.00	33.61	3.71	0	7233.16	66.02	7.54	
528	...	0.00	0.00	0.00	0.00	0	8415.67	36.35	14.83	
247	...	254.95	147.30	60.43	11.49	1	6824.45	65.55	7.90	
250	...	632.46	307.02	161.45	5.93	0	4667.21	65.86	7.36	
485	...	0.00	0.00	0.00	0.00	0	10674.79	36.41	14.92	
467	...	0.00	0.00	0.00	0.00	0	11277.47	36.44	14.90	
723	...	254.56	84.85	146.97	3.82	0	11172.62	65.80	6.22	
483	...	375.00	127.08	109.90	2.95	0	9370.56	36.51	15.08	
886	...	90.00	90.00	0.00	4.00	0	6004.08	66.01	6.58	
809	...	127.28	25.46	56.92	20.62	0	3719.47	65.95	6.55	
244	...	738.24	370.16	181.66	4.29	0	6636.30	65.87	7.63	

	target	predicted_target
321	0	0
70	0	0
209	0	0
656	0	0
685	0	0
96	0	0
468	0	0
86	0	0
532	0	0
327	0	0
528	0	0
247	0	0
250	0	0
485	0	0
467	0	0
723	0	0
483	0	0

```
886       0                0
809       0                0
244       0                0

[20 rows x 51 columns]
```

Oil_Spill_Accuracy :- 97% Using_Over_Sampling. That's a good accuracy.

In [ ]: