```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

# Display the first few rows of the dataset
df.head()
```

```
   f_1    f_2      f_3     f_4  f_5       f_6    f_7   f_8      f_9
f_10  \
0    1   2558  1506.09  456.63   90   6395000  40.88  7.89  29780.0
0.19
1    2  22325    79.11  841.03  180  55812500  51.11  1.21  61900.0
0.02
2    3    115  1449.85  608.43   88    287500  40.42  7.34   3340.0
0.18
3    4   1201  1562.53  295.65   66   3002500  42.40  7.97  18030.0
0.19
4    5    312   950.27  440.86   37    780000  41.43  7.03   3350.0
0.17

   ...     f_41      f_42     f_43     f_44   f_45  f_46      f_47
f_48  \
0  ...  2850.00   1000.00   763.16   135.46   3.73     0  33243.19
65.74
1  ...  5750.00  11500.00  9593.48  1648.80   0.60     0  51572.04
65.73
2  ...  1400.00    250.00   150.00    45.13   9.33     1  31692.84
65.81
3  ...  6041.52    761.58   453.21   144.97  13.33     1  37696.21
65.67
4  ...  1320.04    710.63   512.54   109.16   2.58     0  29038.17
65.66

   f_49  target
0  7.95       1
1  6.26       0
2  7.84       1
3  8.07       1
4  7.35       0

[5 rows x 50 columns]
```

```python
df.tail()
```

```
     f_1  f_2    f_3     f_4  f_5     f_6    f_7    f_8    f_9
f_10  ...  \
932  200   12  92.42  364.42  135   97200  59.42  10.34  884.0
```

```
0.17  ...
933  201   11  98.82  248.64  159   89100  59.64  10.18  831.0
0.17  ...
934  202   14  25.14  428.86   24  113400  60.14  17.94  847.0
0.30  ...
935  203   10  96.00  451.30   68   81000  59.90  15.01  831.0
0.25  ...
936  204   11   7.73  235.73  135   89100  61.82  12.24  831.0
0.20  ...

       f_41    f_42    f_43    f_44  f_45  f_46     f_47   f_48  f_49
target
932  381.84  254.56   84.85  146.97  4.50     0  2593.50  65.85  6.39
0
933  284.60  180.00  150.00   51.96  1.90     0  4361.25  65.70  6.53
0
934  402.49  180.00  180.00    0.00  2.24     0  2153.05  65.91  6.12
0
935  402.49  180.00   90.00   73.48  4.47     0  2421.43  65.97  6.32
0
936  254.56  254.56  127.28  180.00  2.00     0  3782.68  65.65  6.26
0

[5 rows x 50 columns]
```

```
df.describe()
```

```
              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
...
50%    1.863000e+05   39.970000    8.200000    2090.000000    0.200000
...
75%    3.304680e+05   52.420000   10.760000    3435.000000    0.260000
...
max    7.131500e+07   82.640000   24.690000  160740.000000    0.740000
...

               f_41          f_42         f_43         f_44
f_45  \
count    937.000000    937.000000   937.000000   937.000000
937.000000
mean     933.928677    427.565582   255.435902   106.112519
5.014002
std     1001.681331    715.391648   534.306194   135.617708
5.029151
min        0.000000      0.000000     0.000000     0.000000
0.000000
25%      450.000000    180.000000    90.800000    50.120000
2.370000
50%      685.420000    270.000000   161.650000    73.850000
3.850000
75%     1053.420000    460.980000   265.510000   125.810000
6.320000
max    11949.330000  11500.000000  9593.480000  1748.130000
76.630000

             f_46          f_47        f_48        f_49      target
count  937.000000    937.000000  937.000000  937.000000  937.000000
mean     0.128068   7985.718004   61.694386    8.119723    0.043757
std      0.334344   6854.504915   10.412807    2.908895    0.204662
min      0.000000   2051.500000   35.950000    5.810000    0.000000
25%      0.000000   3760.570000   65.720000    6.340000    0.000000
50%      0.000000   5509.430000   65.930000    7.220000    0.000000
75%      0.000000   9521.930000   66.130000    7.840000    0.000000
max      1.000000  55128.460000   66.450000   15.440000    1.000000

[8 rows x 50 columns]

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
```

```python
# Check for missing values
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
```

```
Missing Values in Each Column:
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
```

Q.2 -- Use various methods such as Handling null values, One-Hot Encoding, Imputation, and Scaling of Data Pre-Processing where necessary.

```python
# Check for null values
null_values = df.isnull().sum()
null_values
```

```
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
```

```python
# Check for categorical variables
categorical_columns = df.select_dtypes(include=['object']).columns
categorical_columns
```

```
Index([], dtype='object')
```

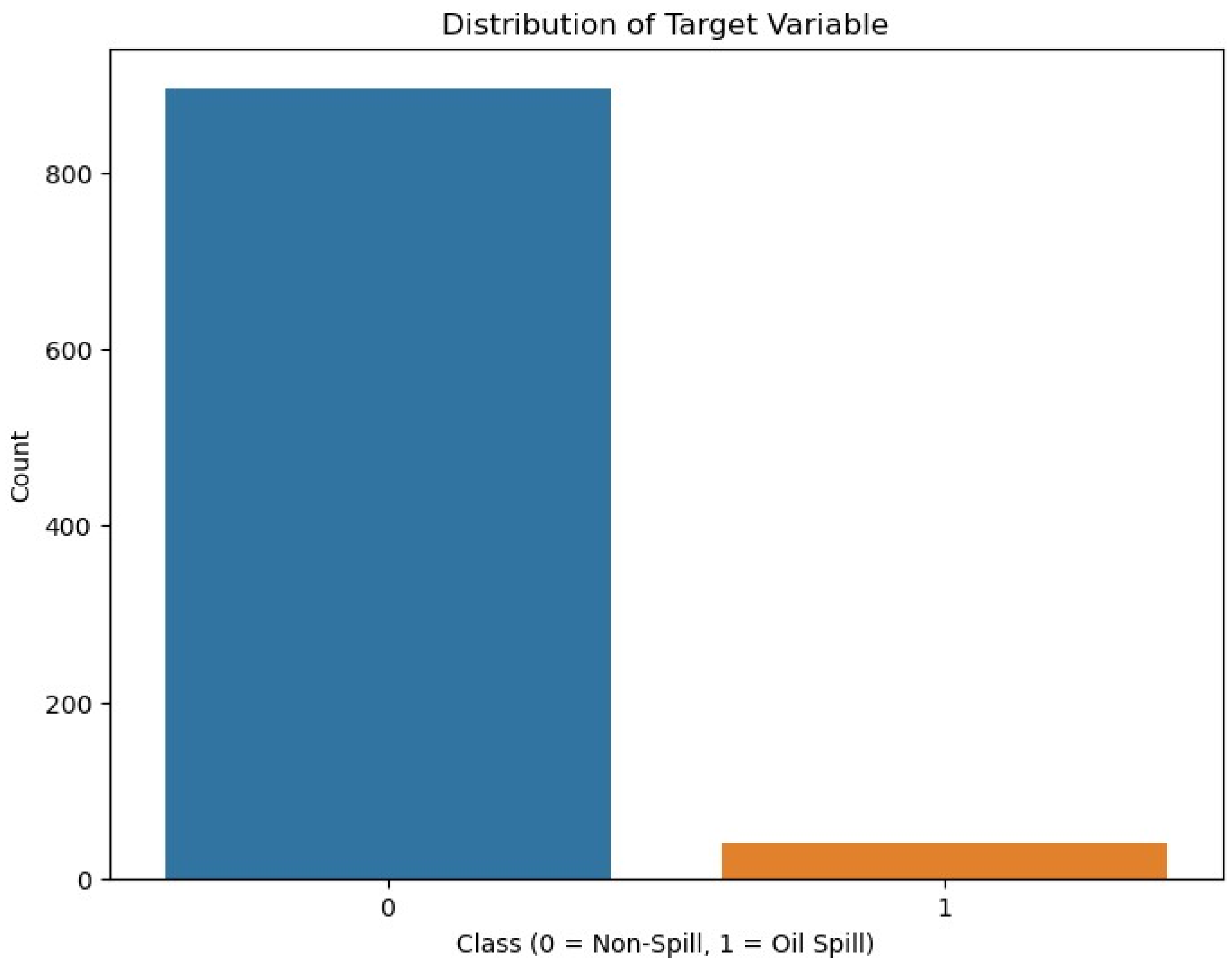
```python
from sklearn.preprocessing import StandardScaler

# Separating features and target
X = df.drop(columns=['target'])
y = df['target']

# Scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert scaled features back to DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

X_scaled_df.head()
```

```
        f_1        f_2       f_3       f_4       f_5        f_6
f_7  \
0 -1.240922   1.152390  1.346434 -0.793007  0.129657   1.469091 -
0.185871
```

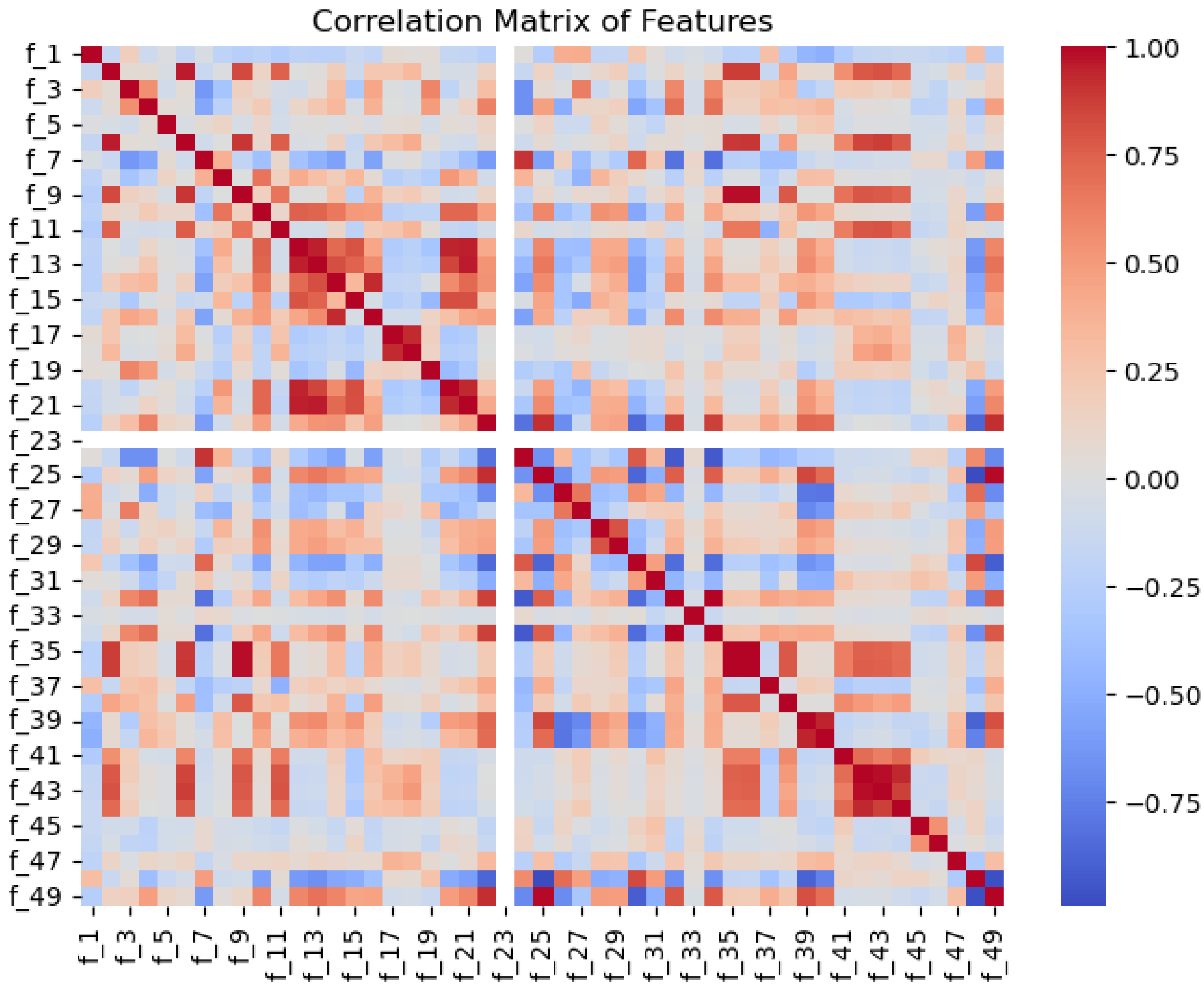
```
1 -1.225524  11.389546 -1.033273 -0.057342  2.114766  14.374844 
0.618905
2 -1.210126  -0.112818  1.252645 -0.502492  0.085544  -0.125929 -
0.222058
3 -1.194727   0.449611  1.440556 -1.101091 -0.399705   0.583114 -
0.066295
4 -1.179329  -0.010794  0.419520 -0.823188 -1.039352   0.002691 -
0.142604

        f_8       f_9      f_10  ...      f_40      f_41       f_42  \
0 -0.345107  3.165389 -0.343460  ...  0.611105  1.913877   0.800597
1 -2.207407  7.100184 -2.226754  ...  0.611105  4.810555  15.485710
2 -0.498440 -0.073589 -0.454242  ...  0.611105  0.465538  -0.248340
3 -0.322804  1.725979 -0.343460  ...  0.611105  5.101741   0.467147
4 -0.584864 -0.072364 -0.565024  ...  0.611105  0.385669   0.395889

        f_43       f_44      f_45      f_46      f_47      f_48 
f_49
0   0.950757   0.216514 -0.255448 -0.383248  3.686767  0.388730 -
0.058377
1  17.486286  11.381341 -0.878152 -0.383248  6.362181  0.387769 -
0.639664
2  -0.197438  -0.449905  0.858654  2.609278  3.460466  0.395456 -
0.096212
3   0.370349   0.286675  1.654442  2.609278  4.336762  0.382004 -
0.017102
4   0.481449   0.022483 -0.484237 -0.383248  3.072971  0.381043 -
0.264751

[5 rows x 49 columns]
```

Q3 -- Derive some insights from the dataset.

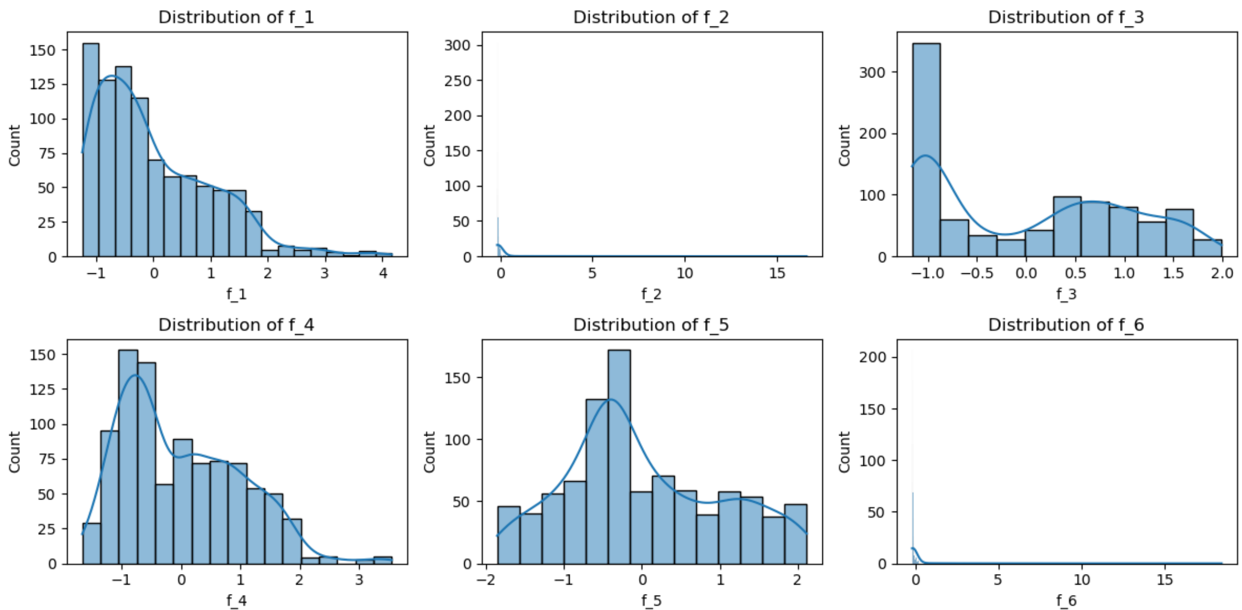
```
# Plot the distribution of the target variable

plt.figure(figsize=(8, 6))
sns.countplot(x=y)
plt.title('Distribution of Target Variable')
plt.xlabel('Class (0 = Non-Spill, 1 = Oil Spill)')
plt.ylabel('Count')
plt.show()
```



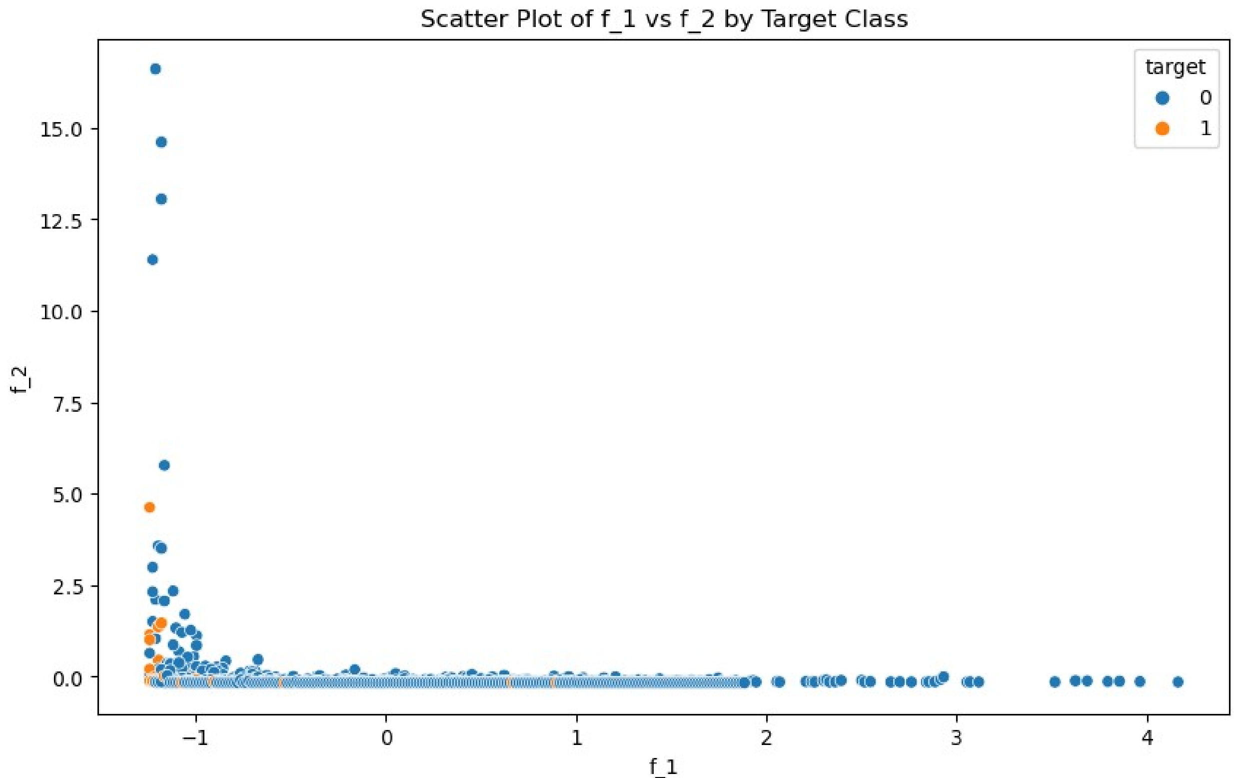

```
# Calculate the correlation matrix

correlation_matrix = X_scaled_df.corr()

# Display a heatmap of the correlation matrix
plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False)
plt.title('Correlation Matrix of Features')
plt.show()
```




```python
# Plot distribution of a few selected features
plt.figure(figsize=(12, 6))

for i, feature in enumerate(X_scaled_df.columns[:6], 1):
    plt.subplot(2, 3, i)
    sns.histplot(X_scaled_df[feature], kde=True)
    plt.title(f'Distribution of {feature}')

plt.tight_layout()
plt.show()
```




```python
# Example of feature-target relationship using a scatter plot

plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_scaled_df['f_1'], y=X_scaled_df['f_2'], hue=y)
plt.title('Scatter Plot of f_1 vs f_2 by Target Class')
plt.xlabel('f_1')
plt.ylabel('f_2')
plt.show()
```



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

1. Splitting the Data

```
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y,
test_size=0.3, random_state=42, stratify=y)
```

A)-- Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix

# Initialize the Logistic Regression model
```

```
logistic_model = LogisticRegression(max_iter=1000)

# Train the model
logistic_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_logistic = logistic_model.predict(X_test)

# Evaluate the model
print("Logistic Regression Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_logistic))
print("Classification Report:\n", classification_report(y_test,
y_pred_logistic))
print("Confusion Matrix:\n", confusion_matrix(y_test,
y_pred_logistic))
```

```
Logistic Regression Model Evaluation:
Accuracy: 0.950354609929078
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.97      0.97       270
           1       0.42      0.42      0.42        12

    accuracy                           0.95       282
   macro avg       0.70      0.70      0.70       282
weighted avg       0.95      0.95      0.95       282

Confusion Matrix:
 [[263   7]
 [  7   5]]
```

B)-- Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
rf_model = RandomForestClassifier()

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test,
```

```
y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest Model Evaluation:
Accuracy: 0.9645390070921985
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98       270
           1       0.67      0.33      0.44        12

    accuracy                           0.96       282
   macro avg       0.82      0.66      0.71       282
weighted avg       0.96      0.96      0.96       282

Confusion Matrix:
 [[268   2]
 [  8   4]]
```

C)-- Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier

# Initialize the Gradient Boosting model
gb_model = GradientBoostingClassifier()

# Train the model
gb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_gb = gb_model.predict(X_test)

# Evaluate the model
print("Gradient Boosting Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
print("Classification Report:\n", classification_report(y_test,
y_pred_gb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_gb))
```

```
Gradient Boosting Model Evaluation:
Accuracy: 0.9574468085106383
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98       270
           1       0.50      0.25      0.33        12

    accuracy                           0.96       282
   macro avg       0.73      0.62      0.66       282
weighted avg       0.95      0.96      0.95       282
```

```
Confusion Matrix:
 [[267   3]
 [  9   3]]
```

D)-- Applying Bagging and Ensemble Methods

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the Bagging model
base_model = DecisionTreeClassifier()
bagging_model = BaggingClassifier(base_estimator=base_model,
n_estimators=50, random_state=42)
bagging_model.fit(X_train, y_train)

# Make predictions
y_pred_bagging = bagging_model.predict(X_test)

# Evaluate the model
bagging_accuracy = accuracy_score(y_test, y_pred_bagging)
print("Bagging Accuracy:", bagging_accuracy)
print("Classification Report:\n", classification_report(y_test,
y_pred_bagging))
```

```
C:\Users\SUDARSHAN PANDEY\anaconda3\Lib\site-packages\sklearn\
ensemble\_base.py:156: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(

Bagging Accuracy: 0.9609929078014184
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98       270
           1       0.57      0.33      0.42        12

    accuracy                           0.96       282
   macro avg       0.77      0.66      0.70       282
weighted avg       0.95      0.96      0.96       282
```

Question 4: - Save the best model and load the model.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib

# Train-test split (Assuming X_train, X_test, y_train, y_test are
```

```
already defined)
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y,
test_size=0.2, random_state=42, stratify=y)

# Initialize and train the Random Forest classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate the model (accuracy as an example)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Save the best model
joblib.dump(model, 'best_model.pkl')
```

```
Accuracy: 0.9628
```

```
['best_model.pkl']
```

```
# Load the saved model
loaded_model = joblib.load('best_model.pkl')

# Use the loaded model to make predictions
predictions = loaded_model.predict(X_test)

# Print the model
print(loaded_model)
```

```
RandomForestClassifier(random_state=42)
```

Question 5: - Take the original data set and make another dataset by randomly picking 20 data points from the oil spill data set and applying the saved model to the same.

```
# Randomly select 20 data points
sample_data = data.sample(n=20, random_state=42)

# Load the saved model
loaded_model = joblib.load('best_model.pkl')

# Separate features from the sample data
X_sample = sample_data.drop(columns=['target'])

# Scale the sample data (assuming you used a scaler during training)
scaler = StandardScaler()
X_sample_scaled = scaler.fit_transform(X_sample)

# Apply the model to predict the target for the sample data
```

```python
sample_predictions = loaded_model.predict(X_sample_scaled)

# Add predictions to the sample data
sample_data['predicted_target'] = sample_predictions

# Print the new dataset with predictions
print(sample_data)
```

```
     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]

C:\Users\SUDARSHAN PANDEY\anaconda3\Lib\site-packages\sklearn\
base.py:464: UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
```