Answer Sheet :

Machine Learning Assignment Submission - (Parikshit Prajapati)

Name : Prajapati Parikshit Virabhai

Phone No. : 9998428706

Email : parikshitprajapti2001@gmail.com

Use the Oil Spill Dataset and solve the following question by using the

dataset, to download the dataset click on the dataset name.

About Dataset :

The dataset was developed by starting with satellite images of the ocean,

Some of which contain an oil spill and some that do not.

Images were split into sections and processed using computer vision

algorithms to provide a vector of features to describe the contents of the

image section or patch.

The task is, given a vector that describes the contents of a patch of a satellite

image, then predicts whether the patch contains an oil spill or not, e.g. from

the illegal or accidental dumping of oil in the ocean.

There are two classes and the goal is to distinguish between spill and

non-spill using the features of 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 a target.

QUESTIONS:

Q1) Download the Oil Spill Dataset and perform Data cleaning and Data Pre-Processing if Necessary.

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

Q3) Derive some insights from the dataset.

Q4) 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.

Q5) Save the best model and Load the model

Q6) Take the original data set and make another dataset by randomly picking 20 data points from the oil spill dataset and applying the save model to the same.

INPUT AND OUTPUT :

Q1) Download the Oil Spill Dataset and perform Data cleaning and Data:

Input :

Import libraries :

```
In [59]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         from sklearn.preprocessing import OneHotEncoder
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
         from sklearn.metrics import accuracy_score, classification_report
         import joblib
```

Load the Dataset

```
In [61]: df= pd.read_csv('oil_spill.csv')
         df.head()
```

Out[61]:

	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	...	2850.00	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 rows × 50 columns

Data Cleaning and Pre-processing:

Data Cleaning and Pre-processing:

In [63]:
```
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
```

In [64]:
```
df.isnull().sum()
```

Out[64]:
```
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
```

Describe Database:

In [65]:
```
df.describe()
```

Out[65]:

	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 [66]:
```
df.duplicated().sum()
```

Out[66]: 0

In [70]:
```
df.dtypes
```

Out[70]:
```
f_1       int64
f_2       int64
f_3     float64
f_4     float64
f_5       int64
f_6       int64
f_7     float64
```

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

Feuture Scalling:

In [71]:
```
features = df.loc[:,'f_2':'f_49']
# features
scaler = StandardScaler()

scaled_features = scaler.fit_transform(features)
df.loc[:, 'f_2':'f_49'] = scaled_features
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_12512\3951132622.py:6: DeprecationWarning: In a future version, `df.iloc[:, i] = ne
wvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df
[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`
  df.loc[:, 'f_2':'f_49'] = scaled_features
```

Correlation Analysis:

In [72]:
```
correlation_metrix = df.corr()
```

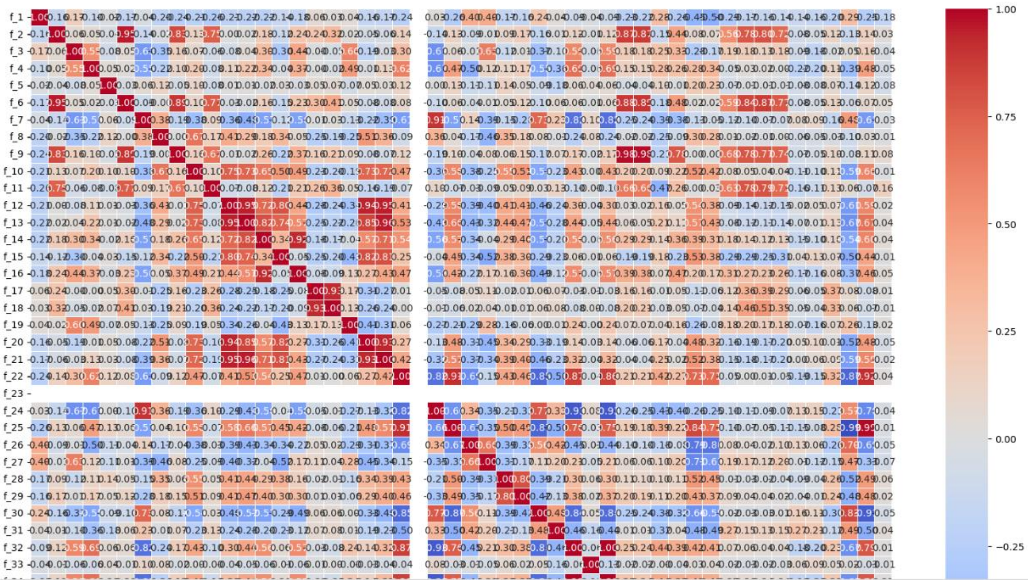
In [73]:
```
correlation_metrix
```

Out[73]:

	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_1	1.000000	-0.155581	0.172017	-0.104116	-0.017025	-0.169533	-0.037412	-0.204983	-0.244551	-0.214447	...	-0.286190	-0.167466	-0.156916	-0.141792
f_2	-0.155581	1.000000	0.058390	0.052638	-0.036870	0.953947	-0.136761	-0.016822	0.829978	0.128465	...	0.555154	0.777807	0.800939	0.716496
f_3	0.172017	0.058390	1.000000	0.549510	-0.082764	0.050795	-0.627934	-0.349541	0.158686	0.073794	...	0.186920	0.178287	0.129653	0.176883
f_4	-0.104116	0.052638	0.549510	1.000000	0.048847	0.024693	-0.546205	-0.222063	0.097683	0.202167	...	-0.046934	0.032402	0.022234	0.000664
f_5	-0.017025	-0.036870	-0.082764	0.048847	1.000000	-0.028431	0.059128	0.123814	-0.047879	0.098573	...	-0.066930	-0.014877	-0.013742	-0.012346
f_6	-0.169533	0.953947	0.050795	0.024693	-0.028431	1.000000	-0.093589	-0.001395	0.894150	0.097449	...	0.594273	0.844597	0.868353	0.770044
f_7	-0.037412	-0.136761	-0.627934	-0.546205	0.059128	-0.093589	1.000000	0.381206	-0.188076	-0.380340	...	-0.115014	-0.100003	-0.074308	-0.073751
f_8	-0.204983	-0.016822	-0.349541	-0.222063	0.123814	-0.001395	0.381206	1.000000	0.001073	0.670628	...	0.013476	-0.015712	-0.013193	0.002439
f_9	-0.244551	0.829978	0.158686	0.097683	-0.047879	0.894150	-0.188076	0.001073	1.000000	0.164098	...	0.675610	0.784833	0.770129	0.736075
f_10	-0.214447	0.128465	0.073794	0.202167	0.098573	0.097449	-0.380340	0.670628	0.164098	1.000000	...	0.082449	0.052518	0.043116	0.042269
f_11	-0.261624	0.745590	-0.064076	-0.082742	-0.075843	0.765628	0.093376	0.167904	0.671358	0.102331	...	0.630674	0.782581	0.790649	0.710990
f_12	-0.209190	0.004035	-0.081738	0.106767	0.009470	-0.029363	-0.363593	0.406409	-0.008391	0.747509	...	-0.088211	-0.135129	-0.121701	-0.147694
f_13	-0.222342	0.020195	0.042723	0.224342	0.013574	-0.017706	-0.481003	0.289904	0.018342	0.730810	...	-0.084692	-0.120182	-0.109534	-0.140570
f_14	-0.220721	0.176080	0.299324	0.335270	-0.016254	0.155767	-0.574566	0.178362	0.261617	0.652360	...	0.177034	0.141294	0.117372	0.130096
f_15	-0.137901	-0.118317	-0.301641	-0.039329	0.028305	-0.147712	-0.115334	0.335692	-0.215468	0.502049	...	-0.292963	-0.293204	-0.250771	-0.308273
f_16	-0.178220	0.235500	0.439603	0.372116	-0.029425	0.226015	-0.563544	0.051995	0.365164	0.487945	...	0.305778	0.269345	0.227190	0.262997
f_17	0.056430	0.237388	-0.003753	-0.000815	0.045836	0.302462	-0.008360	-0.245330	0.160027	-0.231361	...	0.119187	0.361130	0.392898	0.287938
f_18	0.027526	0.321276	-0.046857	-0.020119	0.065762	0.406917	0.027642	-0.188000	0.207135	-0.196430	...	0.144958	0.459463	0.509775	0.353361

Q3)Derive some insights from the dataset.

In [74]:
```
plt.figure(figsize = (20,16))
sns.heatmap(correlation_metrix, annot = True, cmap = 'coolwarm' , fmt = '.2f', linewidths= .5)
plt.show()
```



Drop the column:

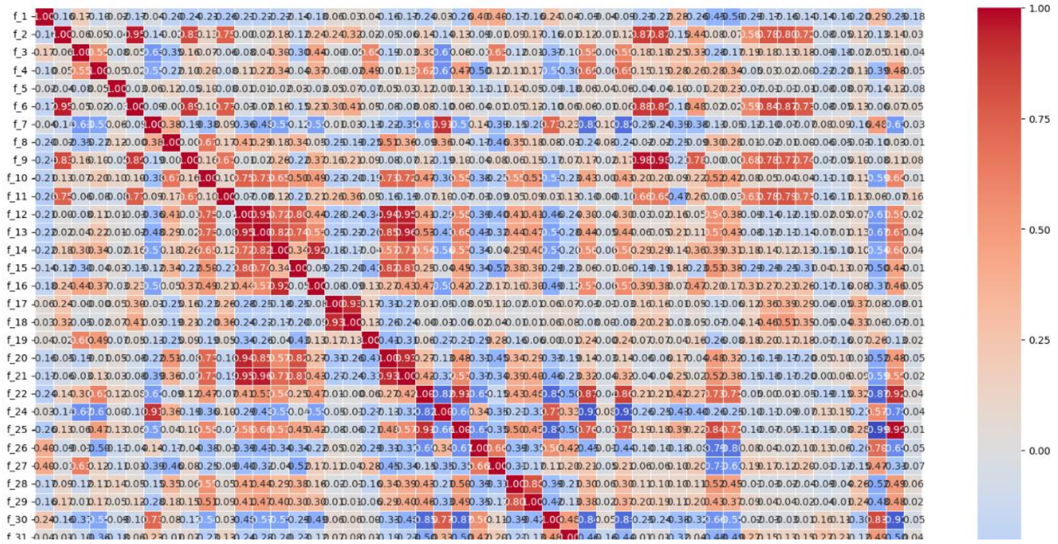
In [78]:
```
df.drop(columns=['f_23'], inplace=True)
```

In [79]:
```
df.columns
```

Out[79]:
```
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')
```

In [80]:
```
correlation_metrix = df.corr()
```

In [81]:
```
plt.figure(figsize = (20,16))
sns.heatmap(correlation_metrix, annot = True, cmap = 'coolwarm' , fmt = '.2f', linewidths= .5)
plt.show()
```

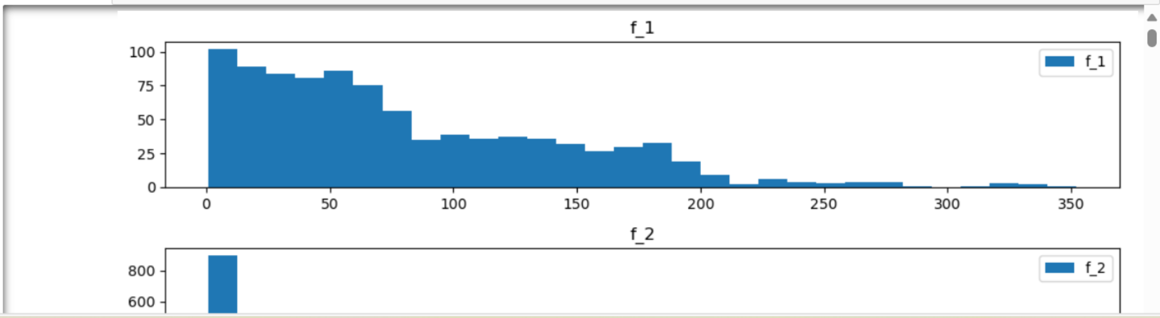


In [82]:
```
columns_of_interest = ['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', 'f_9', 'f_10',
                       'f_11', 'f_12', 'f_13', 'f_14', 'f_15', 'f_16', 'f_17', 'f_18', 'f_19',
                       'f_20', 'f_21', 'f_22', 'f_24', 'f_25', 'f_26', 'f_27', 'f_28', 'f_29',
                       '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']


fig, axes = plt.subplots(nrows=len(columns_of_interest), ncols=1, figsize=(10, 2 * len(columns_of_interest)))

for i, column in enumerate(columns_of_interest):
    df[column].plot(ax=axes[i], kind='hist', bins=30, legend=True)
    axes[i].set_title(column)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Adjusting layout
plt.tight_layout()
plt.show()
```



04) 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.

Split the Data into Train and Test Sets

In [84]:
```
X = df.drop(columns=['target'])
y = df['target']
```

In [85]:
```
X_train, X_test, y_train,y_test = train_test_split(X,y ,test_size = 0.2, random_state = 42)
```

Model Selection and Training:

```
In [20]: rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
         rf_model.fit(X_train, y_train)
```

```
Out[20]:        RandomForestClassifier
         RandomForestClassifier(random_state=42)
```

Bagging and Ensemble Techniques:

```
In [21]: ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)
         ada_model.fit(X_train, y_train)
```

```
Out[21]:        AdaBoostClassifier
         AdaBoostClassifier(n_estimators=100, random_state=42)
```

Model Evaluation:

```
In [22]: def evaluate_model(model, X_test, y_test):
             y_pred = model.predict(X_test)
             acc = accuracy_score(y_test, y_pred)
             return acc

         rf_accuracy = evaluate_model(rf_model, X_test, y_test)
         ada_accuracy = evaluate_model(ada_model, X_test, y_test)
```

Q5) Save the best model and Load the model.

Save and Load the Best Model:

```
In [23]: best_model = rf_model if rf_accuracy > ada_accuracy else ada_model
         joblib.dump(best_model, 'best_model.pkl')
```

```
Out[23]: ['best_model.pkl']
```

Load the saved model:

```
In [24]: loaded_model = joblib.load('best_model.pkl')
```

Q6) Take the original data set and make another dataset by randomly picking 20 data points from the oil spill dataset and applying the saved model to the same.

Apply the Model to a Subset of Data:

```
In [25]: subset_df = df.sample(n=20, random_state=42)
         X_subset = subset_df.drop(columns=['target'])
         y_subset_true = subset_df['target']
         y_subset_pred = loaded_model.predict(X_subset)
```

Display predictions on the subset data:

```
In [26]: print("Predictions on Subset Data:")
         print(y_subset_pred)

         Predictions on Subset Data:
         [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Analyze the characteristics of the subset data:

In [27]:
```
print("Subset Data Info:")
print(subset_df.info())
```

```
Subset Data Info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20 entries, 321 to 244
Data columns (total 49 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   f_1     20 non-null     int64
 1   f_2     20 non-null     float64
 2   f_3     20 non-null     float64
 3   f_4     20 non-null     float64
 4   f_5     20 non-null     float64
 5   f_6     20 non-null     float64
 6   f_7     20 non-null     float64
 7   f_8     20 non-null     float64
 8   f_9     20 non-null     float64
 9   f_10    20 non-null     float64
 10  f_11    20 non-null     float64
 11  f_12    20 non-null     float64
 12  f_13    20 non-null     float64
```

Compare feature distributions between training and subset data:

In [28]:
```
print("\nTraining Data Describe:")
print(X_train.describe())
print("\nSubset Data Describe:")
print(X_subset.describe())
```

```
Training Data Describe:
              f_1         f_2         f_3         f_4         f_5         f_6  \
count  749.000000  749.000000  749.000000  749.000000  749.000000  749.000000
mean    82.544726    0.019476   -0.001653   -0.001427   -0.026036    0.022394
std     64.563112    1.103986    1.003222    1.006780    1.011358    1.110317
min      1.000000   -0.167197   -1.161999   -1.664992   -1.855452   -0.182650
25%     32.000000   -0.162018   -1.023017   -0.825101   -0.686443   -0.168367
50%     65.000000   -0.139749   -0.015288   -0.247899   -0.267365   -0.154474
75%    125.000000   -0.107640    0.875639    0.735508    0.659019   -0.119400
max    352.000000   16.601603    1.818446    3.547380    2.114766   18.423439

              f_7         f_8         f_9        f_10  ...        f_40  \
count  749.000000  749.000000  749.000000  749.000000  ...  749.000000
mean     0.003833   -0.030967    0.018737   -0.029306  ...    0.005495
std      1.008952    0.968020    1.097941    0.968225  ...    1.002029
min     -1.730915   -2.313347   -0.401040   -2.226754  ...   -1.557855
25%     -0.754642   -0.676864   -0.314797   -0.675806  ...   -0.762570
50%     -0.255099   -0.267047   -0.230515   -0.232678  ...   -0.401076
75%      0.728254    0.435498   -0.072058    0.432014  ...    0.611105
max      3.099313    4.224219   19.208377    5.749552  ...    1.840183

             f_41        f_42        f_43        f_44        f_45        f_46  \
```

Analyze feature importance of the trained model:

In [29]:
```
if hasattr(loaded_model, 'feature_importances_'):
    feature_importance = loaded_model.feature_importances_
    print("\nFeature Importance:")
    for i, importance in enumerate(feature_importance):
        print(f"Feature {X_train.columns[i]}: {importance}")
else:
    print("The model does not support feature importance analysis.")
```

```
Feature Importance:
Feature f_1: 0.06281822414999656
Feature f_2: 0.019821470562562695
Feature f_3: 0.019501202791451514
Feature f_4: 0.03455306953609941
Feature f_5: 0.018332321495875964
Feature f_6: 0.02527086393365474
Feature f_7: 0.022456039846791957
Feature f_8: 0.022582236413578642
Feature f_9: 0.025628790818362734
Feature f_10: 0.0219858798777173
Feature f_11: 0.02595011178301022
Feature f_12: 0.016878204879060547
Feature f_13: 0.014379144251241332
Feature f_14: 0.016928888715298365
Feature f_15: 0.012609691320471583
Feature f_16: 0.023380394175093334
Feature f_17: 0.02392199288421556
Feature f_18: 0.017225081439483
Feature f_19: 0.0195702388954833
Feature f_20: 0.010105385431171218
Feature f_21: 0.01957605061545208
Feature f_22: 0.012141836290836728
```