Machine Learning Assignment Submission-(Divyanshu)

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

Importing Libraries

Input-

```
# Importing Numpy
import numpy as np
# Importing Matplotlib
import matplotlib.pyplot as plt
# plt is athe alias name for pyplot
import pandas as pd
# pd is the alias for pandas
```

Loading Data into Dataframe

Input-

```
In [2]: # Loading the Dataset
        oilspill_df = pd.read_csv("oil_spill.csv")
```

- **Data Pre-Processing and Cleaning**

Showing the first five rows

Input-

```
In [3]: oilspill_df.head()
```

Output-

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	...	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

Showing the shape of the Dataset

Input-

In [4]:
```
oilspill_df.shape
```

Output-

Out[4]:
```
(937, 50)
```

Checking Datatypes of all column

Input-

In [5]:
```
oilspill_df.dtypes
```

Output-

Out[5]:
```
f_1          int64
f_2          int64
f_3        float64
f_4        float64
f_5          int64
f_6          int64
f_7        float64
f_8        float64
f_9        float64
f_10       float64
f_11       float64
f_12       float64
f_13       float64
f_14       float64
f_15       float64
f_16       float64
f_17       float64
f_18       float64
f_19       float64
f_20       float64
f_21       float64
f_22       float64
f_23         int64
f_24       float64
f_25       float64
f_26       float64
f_27       float64
f_28       float64
f_29       float64
f_30       float64
f_31       float64
f_32       float64
f_33       float64
f_34       float64
f_35         int64
f_36         int64
f_37       float64
f_38       float64
f_39         int64
f_40         int64
f_41       float64
f_42       float64
f_43       float64
f_44       float64
f_45       float64
f_46         int64
f_47       float64
f_48       float64
f_49       float64
target       int64
dtype: object
```

Checking ratings info

Input-

```
In [6]: |
oilspill_df.info()
```

Output-

```
<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
```

Checking the columns

Input-

```
In [7]: oilspill_df.columns
```

Output-

```
Out[7]: 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')
```

Checking the duplicates

Input-

```
In [8]: oilspill_df.duplicated().sum()
```

Output-

```
Out[8]: 0
```

Check the presence of missing values

Input-

```
In [9]: oilspill_df.isnull().sum()
```

Output-

```
Out[9]: 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
```

Checking the unique elements from the column 'target'

Input-

```
In [10]: oilspill_df["target"].unique()
```

Output-

```
Out[10]: array([1, 0], dtype=int64)
```

Checking the value counts from the column 'target'

Input-

```
In [11]: oilspill_df["target"].value_counts()
```

Output-

```
Out[11]: 0    896
         1     41
         Name: target, dtype: int64
```

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

Importing libraries

```
In [4]: import pandas as pd
        from sklearn.impute import SimpleImputer
        from sklearn.preprocessing import OneHotEncoder, StandardScaler
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score, classification_report
```

➢ Handling null values

Input-

```
In [5]:
        imputer = SimpleImputer(strategy='mean')

        # Identify columns with missing values (assuming numerical columns in this case)
        columns_with_null = oilspill_df.columns[oilspill_df.isnull().any()].tolist()

        # Impute missing values for each column
        for column in columns_with_null:
            oilspill_df[column] = imputer.fit_transform(oilspill_df[[column]])
```

➢ One-Hot Encoding

Input-

```
In [6]: # Check if the column exists in the DataFrame
        if 'categorical_column' in oilspill_df.columns:
            # Extract the categorical column
            categorical_column = oilspill_df[['categorical_column']]

            # Instantiate the OneHotEncoder
            encoder = OneHotEncoder()

            # Fit and transform the data
            encoded_data = encoder.fit_transform(categorical_column).toarray()

            # Create a DataFrame with the encoded data
            encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(['categorical_column']))

            # Concatenate the original DataFrame with the encoded DataFrame
            oilspill_df = pd.concat([oilspill_df, encoded_df], axis=1)

            # Drop the original categorical column if needed
            oilspill_df.drop(['categorical_column'], axis=1, inplace=True)
        else:
            print("Column 'categorical_column' not found in the DataFrame.")
```

Output-

```
Column 'categorical_column' not found in the DataFrame.
```

➢ Imputation

Input-

```
In [7]: # Check if the column exists in the DataFrame
        if 'numerical_column' in oilspill_df.columns:
            # Extract the numerical column
            numerical_column = oilspill_df[['numerical_column']]

            # Instantiate the SimpleImputer with a chosen strategy (mean, median, most_frequent, etc.)
            imputer = SimpleImputer(strategy='mean')

            # Impute missing values for the numerical column
            oilspill_df['numerical_column'] = imputer.fit_transform(numerical_column)
        else:
            print("Column 'numerical_column' not found in the DataFrame.")
```

Output-

```
Column 'numerical_column' not found in the DataFrame.
```

Q-3 Derive some insights from the dataset.

Importing libraries

```
In [8]: import matplotlib.pyplot as plt
        import seaborn as sns
```

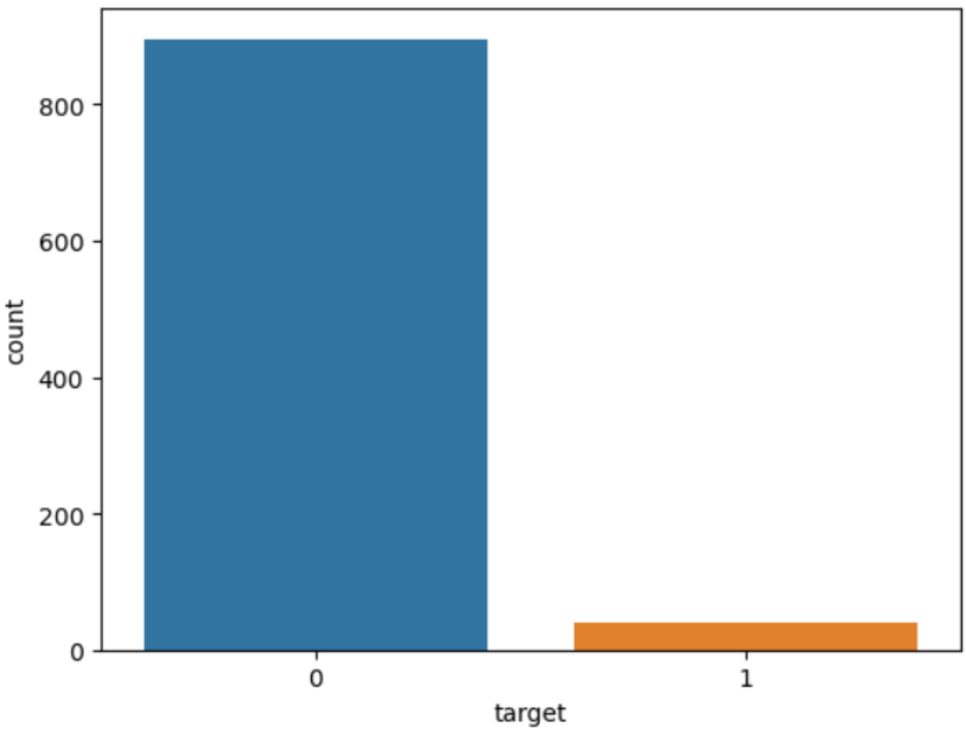
Insights

Input-

```
In [ ]: # Exploratory Data Analysis (EDA)
        # Visualize the distribution of the target variable
        sns.countplot(x='target', data=oilspill_df)
        plt.show()

        # Explore relationships between features and target variable
        sns.pairplot(oilspill_df, hue='target')
        plt.show()
```

Output-



Q-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.

Importing Libraries

In [23]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier, VotingClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
```

- ## Applying various machine learning techniques

Split the dataset

```
# Split the dataset
X = oilspill_df.drop('target', axis=1)
y = oilspill_df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Random Forest classifier

```
# Random Forest Classifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
```

Gradient Boosting classifier

```
# Gradient Boosting Classifier
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
gb_predictions = gb_model.predict(X_test)
```

- ## Bagging

```
# Bagging Classifier (using Decision Tree as base estimator)
bagging_model = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=10, random_state=42)
bagging_model.fit(X_train, y_train)
bagging_predictions = bagging_model.predict(X_test)
```

➢ Ensemble

```
# Ensemble using Voting Classifier (combining Random Forest and Gradient Boosting)
ensemble_model = VotingClassifier(estimators=[('RandomForest', rf_model), ('GradientBoosting', gb_model)], voting='hard')
ensemble_model.fit(X_train, y_train)
ensemble_predictions = ensemble_model.predict(X_test)
```

➢ Evaluating the model for best model

Input-

```
# Evaluate models
models = {
    'Random Forest': rf_predictions,
    'Gradient Boosting': gb_predictions,
    'Bagging': bagging_predictions,
    'Ensemble': ensemble_predictions
}

for model_name, predictions in models.items():
    accuracy = accuracy_score(y_test, predictions)
    report = classification_report(y_test, predictions)

    print(f'Model: {model_name}')
    print(f'Accuracy: {accuracy}')
    print(f'Classification Report:\n{report}')
    print('------------------------')
```

Output-

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

           0       0.98      0.99      0.99       182
           1       0.60      0.50      0.55         6

    accuracy                           0.97       188
   macro avg       0.79      0.74      0.77       188
weighted avg       0.97      0.97      0.97       188

------------------------
Model: Gradient Boosting
Accuracy: 0.9787234042553191
Classification 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

------------------------

Model: Bagging
Accuracy: 0.9680851063829787
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       182
           1       0.50      0.83      0.62         6

    accuracy                           0.97       188
   macro avg       0.75      0.90      0.80       188
weighted avg       0.98      0.97      0.97       188

------------------------
Model: Ensemble
Accuracy: 0.973404255319149
Classification 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

------------------------
```

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

Importing library

```
import joblib
```

Saving and Loading the best model

Input-

```
# Train the Random Forest model (or your best model)
best_model = RandomForestClassifier()
best_model.fit(X_train, y_train)

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

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

Output-

```
Best model saved as 'best_model.joblib'
```

Q-6 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.

Input-

```
# Randomly pick 20 data points from the original dataset
random_sample = oilspill_df.sample(n=20, random_state=42)

# Extract features from the random sample (excluding the target column)
X_random_sample = random_sample.drop('target', axis=1)

# Apply the saved model to make predictions
predictions = loaded_model.predict(X_random_sample)

# Display the original features and predicted labels
result_df = pd.concat([X_random_sample, pd.Series(predictions, name='predicted_label')], axis=1)
print(result_df)
```

Output-

```
       f_1    f_2      f_3      f_4    f_5        f_6    f_7    f_8      f_9  \
321   29.0  105.0   881.92  1128.79   83.0   262500.0  38.90   8.51   2710.0
70    60.0  111.0  1153.32  1283.44   41.0   277500.0  41.25   5.98   1760.0
209   17.0  867.0  1059.49   581.31   46.0  2167500.0  31.08   8.26  15780.0
656    9.0   85.0    71.06   469.47  140.0   688500.0  70.85  11.28   4626.0
685   38.0   15.0    32.47   582.13  156.0   121500.0  73.27  12.11   1080.0
96    86.0   86.0   769.73  1761.26   55.0   215000.0  37.55   6.27   3090.0
468   36.0  462.0   904.13  2689.99  129.0   649687.0  29.80   8.99   5160.0
86    76.0  128.0  1378.47   929.73   51.0   320000.0  39.80   5.20   3370.0
532   38.0  294.0    11.49  1559.36   40.0   413437.0  38.12  22.22   2893.5
327   37.0   98.0  1326.06  1109.08   72.0   245000.0  41.31   7.53   2880.0
528   34.0  151.0   465.77  1736.15   73.0   212343.0  28.96   8.14   3474.0
247  138.0  144.0  1341.72    78.22  110.0   360000.0  31.12   6.88   4650.0
250  156.0  260.0  1080.89   833.29  111.0   650000.0  30.52   7.95   5680.0
485   53.0   84.0   575.19  1558.81  153.0   118125.0  30.94   8.89   1489.5
467   35.0   74.0   619.18  1622.32    5.0   104062.0  26.45   5.92   1255.5
723   76.0   10.0    30.80   348.90  153.0    81000.0  70.50   8.93    720.0
483   51.0   60.0   743.88  1250.60  127.0    84375.0  33.03  11.87   1701.5
886  154.0   10.0   182.50   460.00   90.0    81000.0  57.60   8.68    810.0
809   77.0   13.0   160.77   420.23   63.0   105300.0  51.15  10.66   1191.0
244  118.0  308.0  1313.18   791.35   61.0   770000.0  29.13   7.14   5880.0
0      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
1      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
2      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
3      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
4      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
5      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
6      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
7      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
8      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
9      NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
10     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
11     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
12     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
13     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
14     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
15     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
16     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
17     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
18     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
19     NaN    NaN      NaN      NaN    NaN        NaN    NaN    NaN      NaN
```

	f_10	...	f_41	f_42	f_43	f_44	f_45	f_46	f_47	\
321	0.22	...	955.25	353.55	226.91	84.74	4.21	0.0	3425.75	
70	0.14	...	710.63	500.00	296.40	140.92	2.40	0.0	5915.80	
209	0.27	...	3146.82	1131.37	637.97	408.01	4.93	0.0	5679.31	
656	0.16	...	1279.14	509.12	323.98	87.51	3.95	0.0	6376.53	
685	0.17	...	685.42	201.25	105.89	84.66	6.47	0.0	3285.95	
96	0.17	...	1400.89	180.28	93.84	59.34	14.93	1.0	15720.91	
468	0.30	...	0.00	0.00	0.00	0.00	0.00	0.0	40916.70	
86	0.13	...	1350.93	320.16	160.29	94.32	8.43	0.0	9183.53	
532	0.58	...	0.00	0.00	0.00	0.00	0.00	0.0	10484.87	
327	0.18	...	728.01	269.26	196.00	33.61	3.71	0.0	7233.16	
528	0.28	...	0.00	0.00	0.00	0.00	0.00	0.0	8415.67	
247	0.22	...	1691.89	254.95	147.30	60.43	11.49	1.0	6824.45	
250	0.26	...	1820.03	632.46	307.02	161.45	5.93	0.0	4667.21	
485	0.29	...	0.00	0.00	0.00	0.00	0.00	0.0	10674.79	
467	0.22	...	0.00	0.00	0.00	0.00	0.00	0.0	11277.47	
723	0.13	...	324.50	254.56	84.85	146.97	3.82	0.0	11172.62	
483	0.36	...	375.00	375.00	127.08	109.90	2.95	0.0	9370.56	
886	0.15	...	360.00	90.00	90.00	0.00	4.00	0.0	6004.08	
809	0.21	...	524.79	127.28	25.46	56.92	20.62	0.0	3719.47	
244	0.24	...	1588.24	738.24	370.16	181.66	4.29	0.0	6636.30	
0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
6	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
8	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
11	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
12	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
13	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
14	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
15	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
16	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
17	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
18	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
19	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

	f_48	f_49	predicted_label
321	65.97	7.04	NaN
70	66.12	7.34	NaN
209	65.74	7.42	NaN
656	65.98	6.22	NaN
685	66.11	5.98	NaN
96	66.30	6.71	NaN
468	36.71	14.53	NaN
86	65.98	7.73	NaN
532	36.02	14.82	NaN
327	66.02	7.54	NaN
528	36.35	14.83	NaN
247	65.55	7.90	NaN
250	65.86	7.36	NaN
485	36.41	14.92	NaN
467	36.44	14.90	NaN
723	65.80	6.22	NaN
483	36.51	15.08	NaN
886	66.01	6.58	NaN
809	65.95	6.55	NaN
244	65.87	7.63	NaN
0	NaN	NaN	0.0
1	NaN	NaN	0.0
2	NaN	NaN	0.0
3	NaN	NaN	0.0
4	NaN	NaN	0.0
5	NaN	NaN	0.0
6	NaN	NaN	0.0
7	NaN	NaN	0.0
8	NaN	NaN	0.0
9	NaN	NaN	0.0
10	NaN	NaN	0.0
11	NaN	NaN	0.0
12	NaN	NaN	0.0
13	NaN	NaN	0.0
14	NaN	NaN	0.0
15	NaN	NaN	0.0
16	NaN	NaN	0.0
17	NaN	NaN	0.0
18	NaN	NaN	0.0
19	NaN	NaN	0.0

[40 rows x 50 columns]