In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_excel('oil_spill.xlsx')
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_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

In [3]:
```
df.shape
```

Out[3]:
```
(937, 50)
```

In [4]:
```
df.columns
```

Out[4]:
```
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 [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
df.dtypes
```

Out[7]:
```
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
```

In [8]:
```
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
```

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

Out[9]:

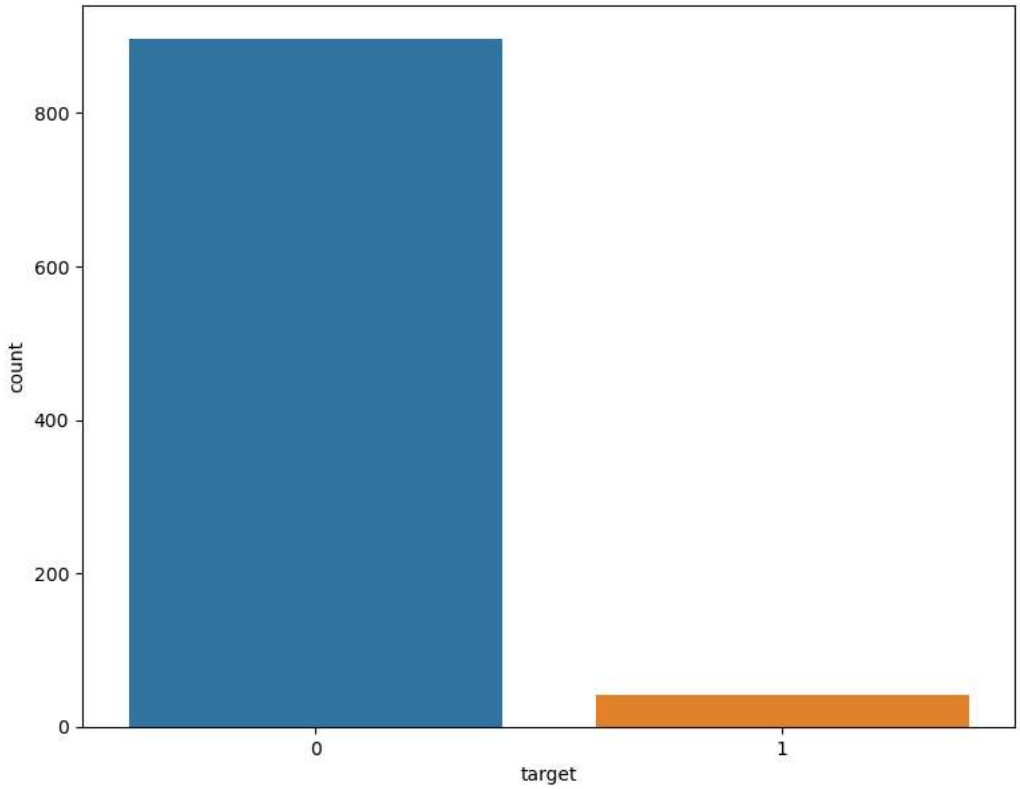
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_10	...	f_41	
count	937.000000	937.000000	937.000000	937.000000	937.000000	9.370000e+02	937.000000	937.000000	937.000000	937.000000	...	937.000000	9
mean	81.588047	332.842049	698.707086	870.992209	84.121665	7.696964e+05	43.242721	9.127887	3940.712914	0.221003	...	933.928677	4
std	64.976730	1931.938570	599.965577	522.799325	45.361771	3.831151e+06	12.718404	3.588878	8167.427625	0.090316	...	1001.681331	7
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	1
50%	64.000000	65.000000	704.370000	761.280000	73.000000	1.863000e+05	39.970000	8.200000	2090.000000	0.200000	...	685.420000	2
75%	124.000000	132.000000	1223.480000	1260.370000	117.000000	3.304680e+05	52.420000	10.760000	3435.000000	0.260000	...	1053.420000	4
max	352.000000	32389.000000	1893.080000	2724.570000	180.000000	7.131500e+07	82.640000	24.690000	160740.000000	0.740000	...	11949.330000	11

8 rows × 50 columns

In [10]:
```
df['target'].value_counts()
```

Out[10]:
```
0    896
1     41
Name: target, dtype: int64
```

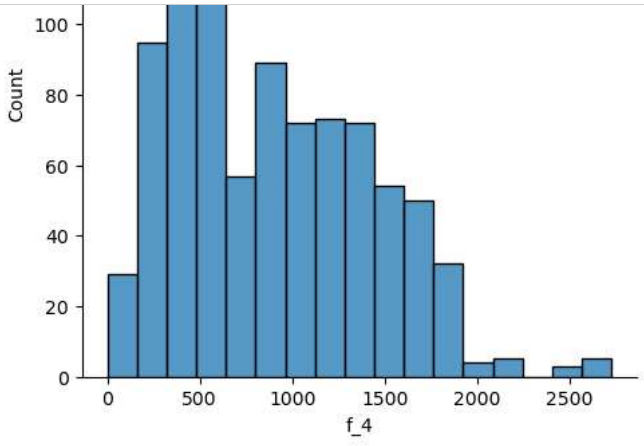
In [11]:
```
plt.figure(figsize= (9,7))
sns.countplot(x = df['target'])
plt.show()
```



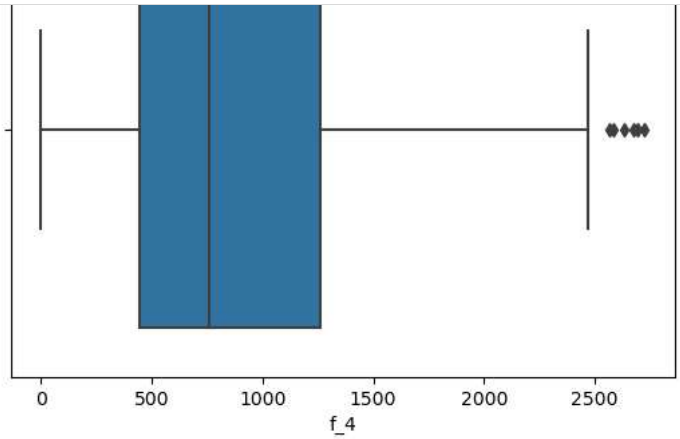
In [12]:
```
df.columns
```

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 [13]:
```
for i in df.columns:
    sns.displot(df[i])
    plt.title(f'displot for {i}')
    plt.show()
```



In [14]:
```
for i in df.columns:
    sns.boxplot(df[i])
    plt.title(f'Boxplot for {i}')
    plt.show()
```



In [ ]:

Outlier to be treated

f_8,

f_10

f_13

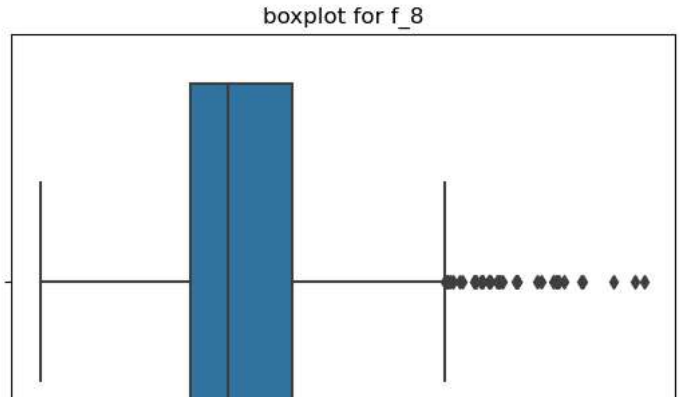
f_41

f_44

f_47

In [15]:
```
sns.boxplot(df['f_8'])
plt.title('boxplot for f_8')
```

Out[15]: Text(0.5, 1.0, 'boxplot for f_8')



In [16]:
```
q1 = np.percentile(df['f_8'],25)
q3 = np.percentile(df['f_8'],75)
iqr = q3-q1
v_max = q3 + 1.5*iqr
print(v_max)
```

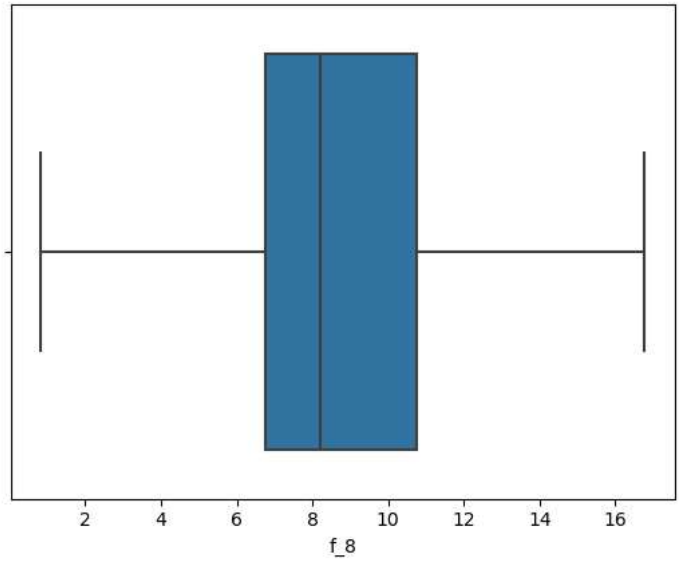
16.775

In [17]:
```
print(df[df['f_8']>16.775].shape)
```

(38, 50)

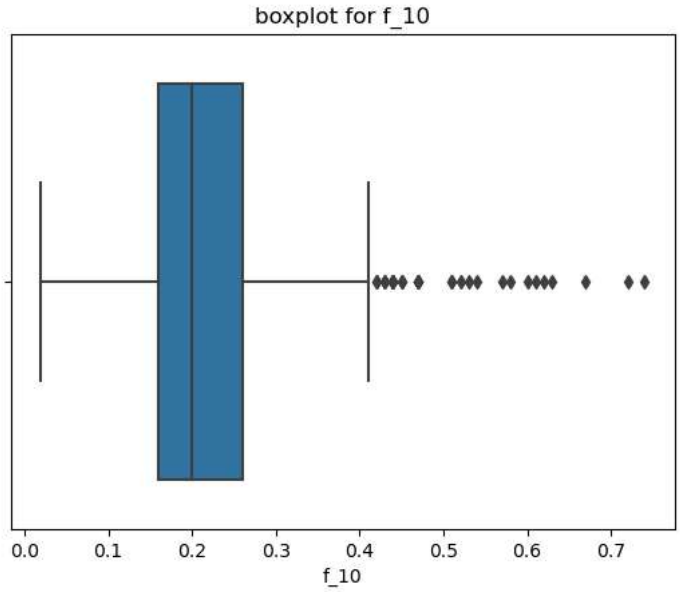
In [18]:
```
df['f_8'] = np.where(df['f_8']>16.775,16.775,df['f_8'])
sns.boxplot(x = df['f_8'])
```

Out[18]: <AxesSubplot:xlabel='f_8'>



In [19]:
```
sns.boxplot(df['f_10'])
plt.title('boxplot for f_10')
```

Out[19]: Text(0.5, 1.0, 'boxplot for f_10')



In [20]:
```
q1 = np.percentile(df['f_10'],25)
q3 = np.percentile(df['f_10'],75)
iqr = q3-q1
v_max = q3 + 1.5*iqr
print(v_max)
```

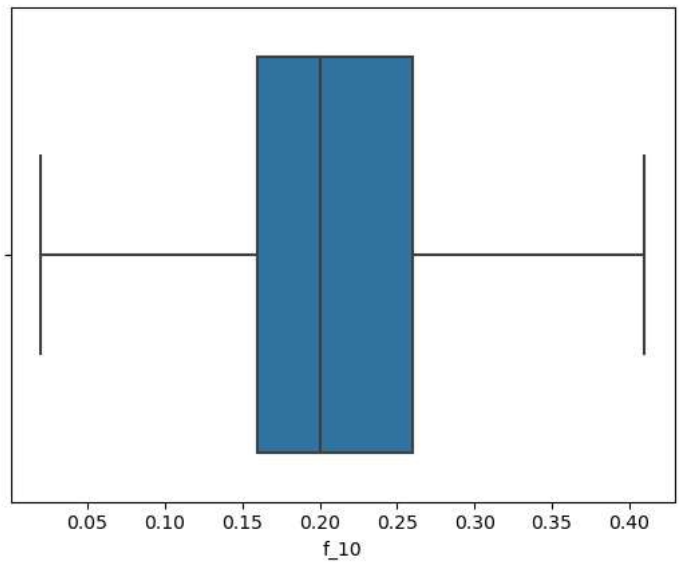
```
0.41000000000000003
```

In [21]:
```
print(df[df['f_10']>0.41].shape)
```

```
(29, 50)
```

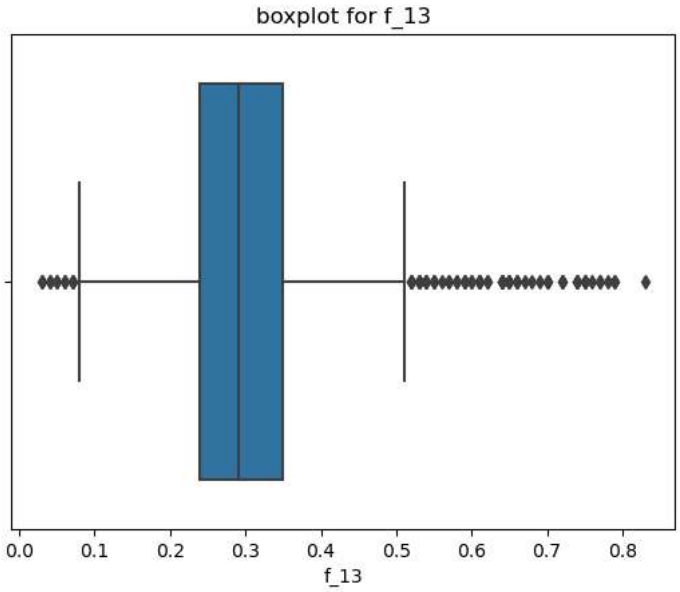
In [22]:
```
df['f_10'] = np.where(df['f_10']>0.41,0.41,df['f_10'])
sns.boxplot(x = df['f_10'])
```

Out[22]: <AxesSubplot:xlabel='f_10'>



In [23]:
```
sns.boxplot(df['f_13'])
plt.title('boxplot for f_13')
```

Out[23]: Text(0.5, 1.0, 'boxplot for f_13')

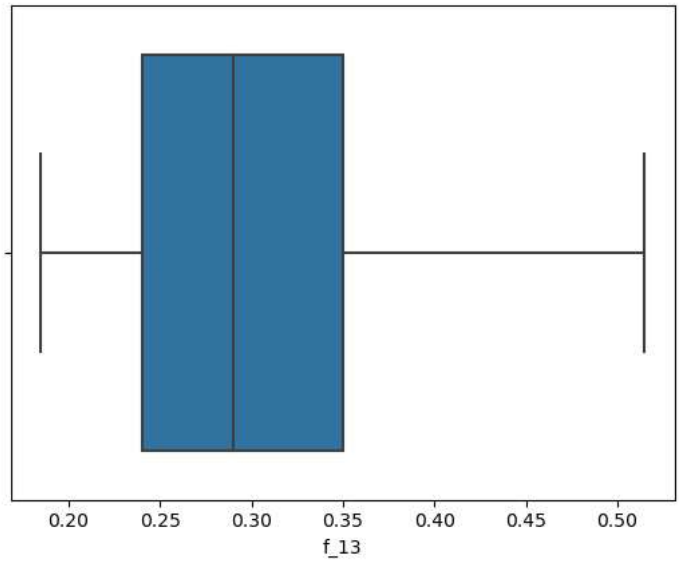


In [24]:
```
q1 = np.percentile(df['f_13'],25)
q3 = np.percentile(df['f_13'],75)
iqr = q3-q1
v_max = q3 + 1.5*iqr
v_min = q3 - 1.5*iqr
print(v_max)
print(v_min)
```

```
0.5149999999999999
0.185
```

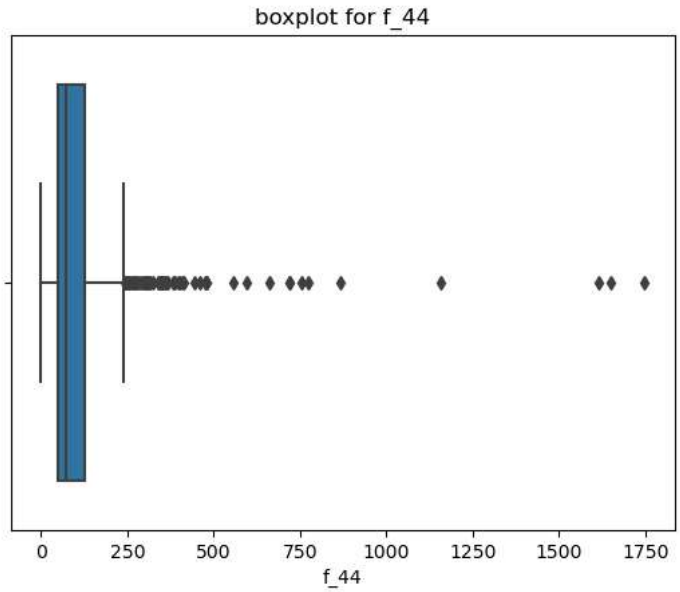
In [25]:
```
df['f_13'] = np.where(df['f_13']>0.5149999999999999,0.5149999999999999,df['f_13'])
df['f_13'] = np.where(df['f_13']<0.185,0.185,df['f_13'])
sns.boxplot(x = df['f_13'])
```

Out[25]: <AxesSubplot:xlabel='f_13'>



In [26]:
```
sns.boxplot(df['f_44'])
plt.title('boxplot for f_44')
```

Out[26]: Text(0.5, 1.0, 'boxplot for f_44')



In [27]:
```
q1 = np.percentile(df['f_44'],25)
q3 = np.percentile(df['f_44'],75)
iqr = q3-q1
v_max = q3 + 1.5*iqr
v_min = q3 - 1.5*iqr
print(v_max)
```

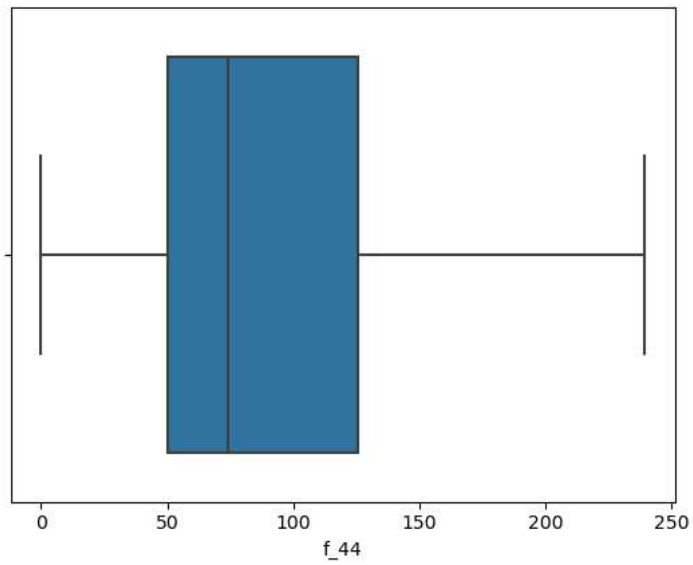
```
239.345
```

In [28]:
```
print(df[df['f_44']>239.354].shape)
```

```
(64, 50)
```

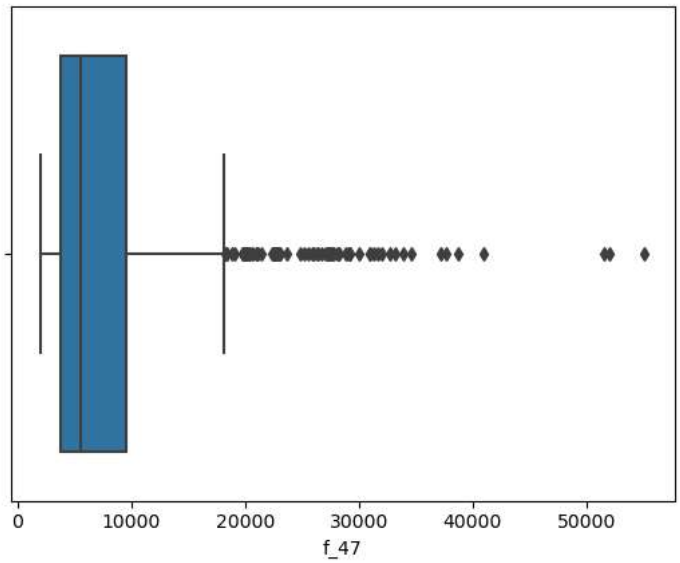
In [29]:
```
df['f_44'] = np.where(df['f_44']>239.345,239.345,df['f_44'])
sns.boxplot(x = df['f_44'])
```

Out[29]: <AxesSubplot:xlabel='f_44'>



In [30]:
```
sns.boxplot(df['f_47'])
```

Out[30]: <AxesSubplot:xlabel='f_47'>



In [31]:
```
q1 = np.percentile(df['f_47'],25)
q3 = np.percentile(df['f_47'],75)
iqr = q3-q1
v_max = q3 + 1.5*iqr
v_min = q3 - 1.5*iqr
print(v_max)
```

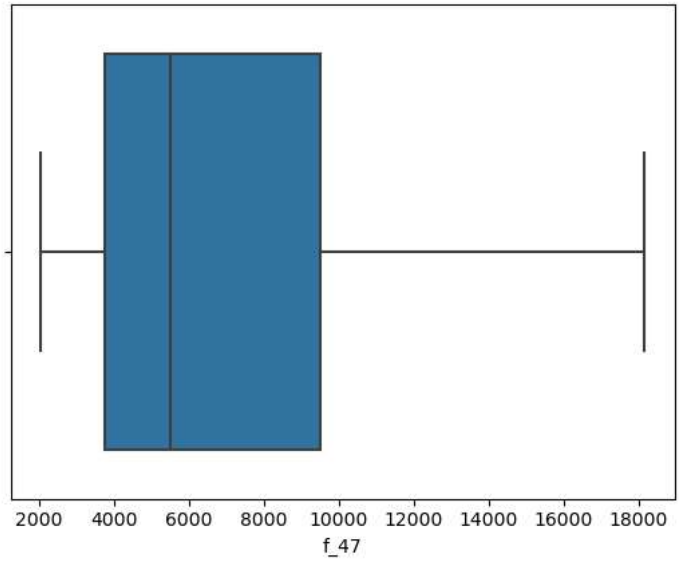
```
18163.97
```

In [32]:
```
print(df[df['f_47']>18163.97].shape)
```

```
(72, 50)
```

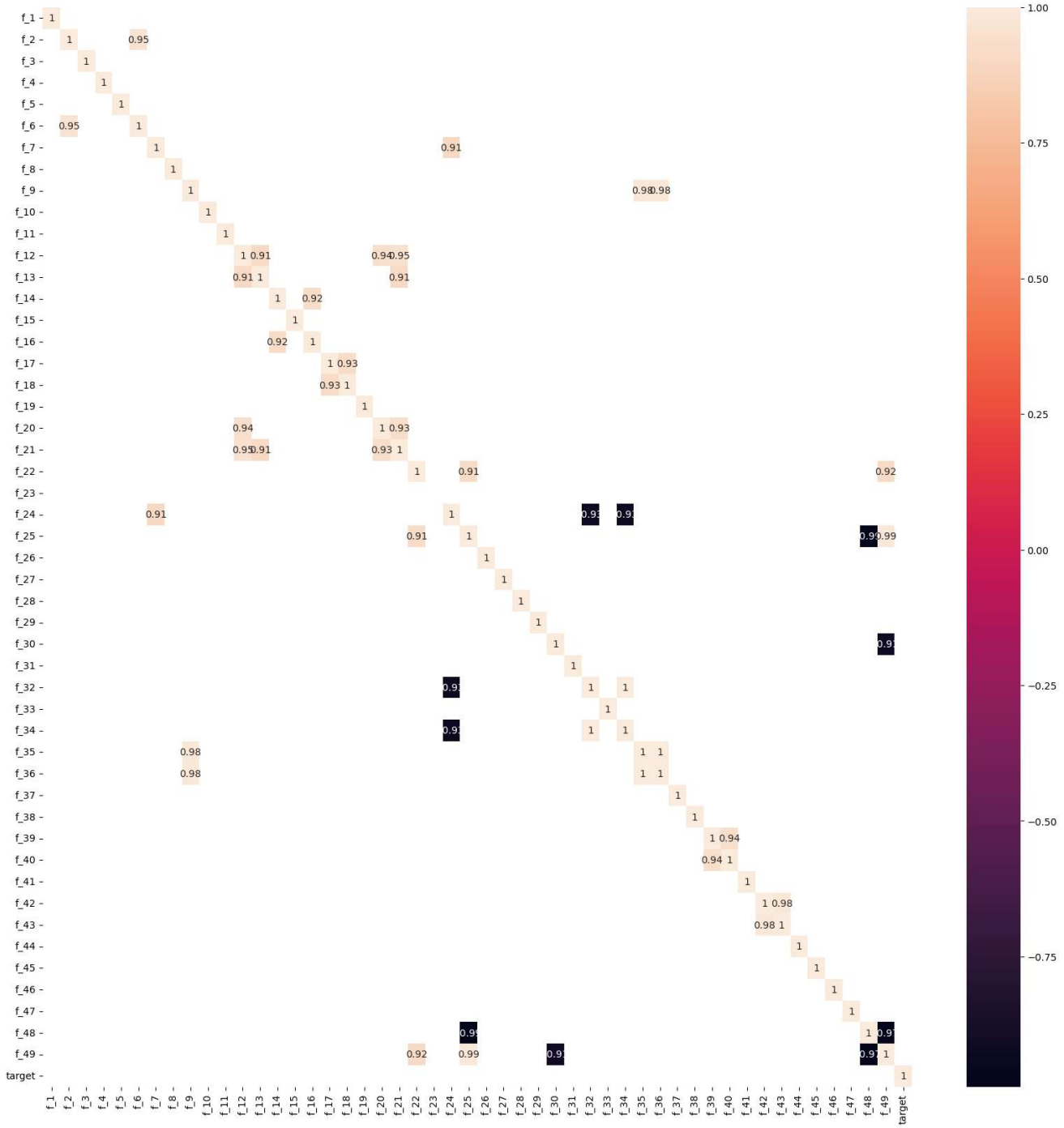
In [33]:
```
df['f_47'] = np.where(df['f_47']>18163.97,18163.97,df['f_47'])
sns.boxplot(x = df['f_47'])
```

Out[33]: <AxesSubplot:xlabel='f_47'>



In [34]:
```
plt.figure(figsize= (20,20))
corr = df.corr()
corr = corr[abs(corr)>0.9]
sns.heatmap(corr , annot = True)
```

Out[34]: <AxesSubplot:>



Here droping highly corelated columns are following

f_9

f_25

f_30

f_48

f_49

In [35]:
```
df.drop(['f_9','f_25','f_30','f_48','f_49'], axis = 1, inplace = True)
```

In [36]:
```
df.columns
```

Out[36]:
```
Index(['f_1', 'f_2', 'f_3', 'f_4', 'f_5', 'f_6', 'f_7', 'f_8', '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_26', 'f_27', 'f_28', 'f_29', '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', 'target'],
      dtype='object')
```

In [37]:
```
df.shape
```

Out[37]:
```
(937, 45)
```

In [38]:
```
x = df.drop(['target'], axis = 1)
y = df['target']
print(type(x))
print(type(y))
print(x.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(937, 44)
(937,)
```

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y , test_size=0.25, random_state= 55)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(702, 44)
(235, 44)
(702,)
(235,)
```

In [41]:
```
from sklearn.preprocessing import StandardScaler
```

In [42]:
```
scaler = StandardScaler()
#fit the scaler to train set, it will learn the paerameters
scaler.fit(x_train)
#transform the train and test sets
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

In [43]:
```
x_train_scaled = pd.DataFrame(x_train_scaled, columns = x_train.columns)
x_test_scaled = pd.DataFrame(x_test_scaled, columns = x_test.columns)
```

In [44]:
```
x_train_scaled.head(3)
```

Out[44]:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_10	f_11	...	f_38	f_39	f_40	f_41	
0	-0.838841	-0.168880	-1.050269	-0.483907	-0.350673	-0.166102	0.802541	-0.008039	-0.634880	-0.339192	...	-0.262874	-0.429432	-0.779191	0.132371	-0.3
1	2.006927	-0.123646	1.071601	-0.031226	-0.219135	-0.147670	-0.987585	-1.084454	-0.508013	-0.377937	...	0.013008	-1.256487	-1.583377	-0.041428	-0.1
2	0.711943	-0.172499	-0.499858	-1.026330	0.197400	-0.181362	0.796218	1.326837	0.380061	-0.091565	...	-0.588407	-0.429432	-0.779191	-0.400611	-0.0

3 rows × 44 columns

In [45]:
```
x_test_scaled.head(3)
```

Out[45]:

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_10	f_11	...	f_38	f_39	f_40	f_41	
0	-0.902790	-0.147168	-1.007065	-0.532250	1.907387	-0.074541	2.103445	1.095742	-0.508013	0.231867	...	-0.010577	0.489517	0.902287	0.360100	0.0
1	1.895015	-0.175515	-1.093557	-0.982231	0.416629	-0.194079	1.195341	0.986276	-0.127410	-0.142101	...	-0.687091	-0.429432	-0.779191	-0.587469	-0.2
2	-0.087430	-0.117012	1.475213	0.830962	-0.789131	-0.139035	-0.631140	0.475435	1.014399	-0.576712	...	0.495877	-0.613222	-0.413652	-0.004108	0.3

3 rows × 44 columns

In [46]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

In [47]:
```
def report(y_test, ypred):
    print('Confusion Matrix\n',confusion_matrix(y_test, ypred))
    print('Classification Report\n',classification_report(y_test, ypred))
    print('Accuracy Score\n',accuracy_score(y_test, ypred))
def score(model):
    print('Train Score', model.score(x_train, y_train))
    print('Test Score', model.score(x_test, y_test))
```

LinearRegression

In [48]:
```
from sklearn.linear_model import LogisticRegression
```

In [49]:
```
model_lr = LogisticRegression(max_iter = 500, solver='liblinear')
model_lr.fit(x_train,y_train)
```

Out[49]:
```
LogisticRegression(max_iter=500, solver='liblinear')
```

In [50]:
```
ypred_lr = model_lr.predict(x_test)
print(ypred_lr[:5])
```

```
[0 0 0 0 0]
```

In [51]:
```
report(y_test, ypred_lr)
```

```
Confusion Matrix
 [[219   4]
 [  9   3]]
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       223
           1       0.43      0.25      0.32        12

    accuracy                           0.94       235
   macro avg       0.69      0.62      0.64       235
weighted avg       0.93      0.94      0.94       235

Accuracy Score
 0.9446808510638298
```

In [52]:
```
score(model_lr)
```

```
Train Score 0.9672364672364673
Test Score 0.9446808510638298
```

KNN Classifier

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [54]:
```
model_kn = KNeighborsClassifier(n_neighbors=10)
model_kn.fit(x_train,y_train)
```

Out[54]:
```
KNeighborsClassifier(n_neighbors=10)
```

In [55]:
```
ypred_kn = model_kn.predict(x_test)
print(ypred_kn[:5])
```

```
[0 0 0 0 0]
```

In [56]:
```
report(y_test, ypred_kn)
```

```
Confusion Matrix
 [[223   0]
 [ 12   0]]
Classification Report
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       223
           1       0.00      0.00      0.00        12

    accuracy                           0.95       235
   macro avg       0.47      0.50      0.49       235
weighted avg       0.90      0.95      0.92       235

Accuracy Score
 0.948936170212766
```

In [57]:
```
score(model_kn)
```

```
Train Score 0.9586894586894587
Test Score 0.948936170212766
```

DecisionTreeClassifier

In [58]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [59]:
```
model_dt = DecisionTreeClassifier(criterion='entropy', max_depth=4 , min_samples_split=25, random_state=45)
model_dt.fit(x_train,y_train)
```

Out[59]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_split=25,
                       random_state=45)
```

In [60]:
```
ypred_dt = model_dt.predict(x_test)
print(ypred_dt[:5])
```

```
[0 0 0 0 0]
```

In [61]:
```
report(y_test,ypred_dt)
```

```
Confusion Matrix
 [[215   8]
 [  8   4]]
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       223
           1       0.33      0.33      0.33        12

    accuracy                           0.93       235
   macro avg       0.65      0.65      0.65       235
weighted avg       0.93      0.93      0.93       235

Accuracy Score
 0.9319148936170213
```

In [62]:
```
#Overfit model
score(model_dt)
```

```
Train Score 0.98005698005698
Test Score 0.9319148936170213
```

RandomForestClassifier

In [63]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [64]:
```
model_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', min_samples_split=25, random_state=45)
model_rf.fit(x_train,y_train)
```

Out[64]:
```
RandomForestClassifier(criterion='entropy', min_samples_split=25,
                       random_state=45)
```

In [65]:
```
ypred_rf = model_rf.predict(x_test)
print(ypred_rf[:5])
```

```
[0 0 0 0 0]
```

In [66]:
```
report(y_test, ypred_rf)
```

```
Confusion Matrix
 [[222   1]
 [ 12   0]]
Classification Report
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       223
           1       0.00      0.00      0.00        12

    accuracy                           0.94       235
   macro avg       0.47      0.50      0.49       235
weighted avg       0.90      0.94      0.92       235

Accuracy Score
 0.9446808510638298
```

In [67]:
```
score(model_rf)
```

```
Train Score 0.9629629629629629
Test Score 0.9446808510638298
```

BaggingClassifier

In [68]:
```
from sklearn.ensemble import BaggingClassifier
```

In [69]:
```
model_bg = BaggingClassifier(base_estimator=model_dt, n_estimators=100, max_samples = x_train.shape[0],
                             max_features=x_train.shape[1], random_state=45)
model_bg.fit(x_train, y_train)
```

Out[69]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy',
                                                        max_depth=4,
                                                        min_samples_split=25,
                                                        random_state=45),
                  max_features=44, max_samples=702, n_estimators=100,
                  random_state=45)
```

In [70]:
```
ypred_bg = model_bg.predict(x_test)
print(ypred_bg[:5])
```

```
[0 0 0 0 0]
```

In [71]:
```
report(y_test, ypred_bg)
```

```
Confusion Matrix
 [[219   4]
 [  6   6]]
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       223
           1       0.60      0.50      0.55        12

    accuracy                           0.96       235
   macro avg       0.79      0.74      0.76       235
weighted avg       0.95      0.96      0.96       235

Accuracy Score
 0.9574468085106383
```

In [72]:
```
score(model_bg)
```

```
Train Score 0.9729344729344729
Test Score 0.9574468085106383
```

AdaBoostClassifier

In [73]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [74]:
```
model_ab = AdaBoostClassifier(base_estimator = model_dt, n_estimators = 90, random_state=45)
model_ab.fit(x_train,y_train)
```

Out[74]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy',
                                                         max_depth=4,
                                                         min_samples_split=25,
                                                         random_state=45),
                   n_estimators=90, random_state=45)
```

In [75]:
```
ypred_ab = model_ab.predict(x_test)
print(ypred_ab[:5])
```

```
[0 0 0 0 0]
```

In [76]:
```
report(y_test, ypred_ab)
```

```
Confusion Matrix
 [[220   3]
 [  7   5]]
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       223
           1       0.62      0.42      0.50        12

    accuracy                           0.96       235
   macro avg       0.80      0.70      0.74       235
weighted avg       0.95      0.96      0.95       235

Accuracy Score
 0.9574468085106383
```

In [77]:
```
score(model_ab)
```

```
Train Score 1.0
Test Score 0.9574468085106383
```

From the above we can conlclude that best model is of BaggingClassifier

In [78]:
```
import pickle
```

In [79]:
```
pickle.dump(model_bg,open('model_bg.pkl','wb'))
```

In [80]:
```
model = pickle.load(open('model_bg.pkl','rb'))
```

In [81]:
```
data = x_train.sample(20)
print(data)
```

```
     f_1   f_2      f_3      f_4  f_5      f_6    f_7     f_8  f_10   f_11  \
511   17   236   256.73  1301.15  119   331875  25.98  12.110  0.41   81.5
413  152   102   839.88  1231.77   90   255000  38.96   7.800  0.20  101.6
346   59    55  1407.87   913.91   94   137500  41.25   6.510  0.16   64.6
642   74    11    65.00   386.27  105    89100  57.45   9.610  0.17  107.2
425  177    59   844.44  1106.71  101   147500  40.80   7.730  0.19   77.2
18     8  1035  1304.90  1281.72   68  2587500  35.90   7.510  0.21  167.8
36    26    81  1705.68  1492.58   19   202500  35.65   6.320  0.18   81.7
225   68    94  1062.21   719.81   83   235000  31.88   7.670  0.24   79.4
331   41   204   906.97   960.46  107   510000  39.53   7.190  0.18   88.5
764   32   145    20.19   418.42   72  1174500  61.51  16.775  0.28  184.6
184  179    74  1751.49  1095.38  119   185000  39.46   8.790  0.22   82.6
259  192   162  1178.77   993.60   99   405000  34.73   6.880  0.20   93.3
634   66    10    76.50   441.70  143    81000  49.50  16.775  0.34  104.1
97    87   125  1785.22  1487.77    3   312500  38.87  10.310  0.27  114.1
92    82    83  1260.70  1474.01   66   207500  39.83   7.830  0.20   81.7
729   82    12    13.33   483.92   34    97200  73.33   5.470  0.07  112.0
539   45    65   511.22  1709.25  144    91406  24.95   9.470  0.38   43.2
5      6    54  1438.13   544.91   82   135000  44.67   6.920  0.15   86.0
459   27    91   982.67   899.04   49   127968  27.27   9.120  0.33   54.0
873  141    10    52.70   353.40   95    81000  55.10   7.200  0.13   95.6

     ...   f_38  f_39  f_40     f_41     f_42    f_43     f_44  f_45  f_46  \
511  ...  49.91   143    86  1544.34   270.42  156.69   67.520  9.86     1
413  ...  24.71    85    63   800.00   500.00  262.50  117.620  3.05     0
346  ...  33.00    85    63   651.92   500.00  138.46  113.930  4.71     0
642  ...   7.75    99    67   284.60   180.00  150.00   51.960  1.90     0
425  ...  24.73    85    63   509.90   304.14  208.37   45.110  2.45     0
18   ...  91.89    78    55  4419.56  1234.91  632.82  239.345  6.98     0
36   ...  30.37    78    55   901.39   269.26  178.33   59.630  5.05     0
225  ...  37.28    64    39   912.41   412.31  196.18  128.280  4.65     0
331  ...  65.05    85    63  1408.01   522.02  288.58  116.270  4.88     0
764  ...  34.46    82    50  1793.24   742.16  445.58  100.170  4.02     0
184  ...  27.12    78    55   694.62   291.55  192.72   51.620  3.60     0
259  ...  46.51    64    39  1265.90   509.90  227.86  140.980  5.56     0
634  ...   7.47    99    67   254.56     0.00    0.00    0.000  0.00     1
97   ...  24.02    78    55   901.39   500.00  288.89  135.640  3.12     0
92   ...  31.09    78    55  1051.19   316.23  135.59   80.160  7.75     0
729  ...   7.75   102    73   402.49   127.28   95.46   63.640  4.22     0
539  ...  49.01   143    86     0.00     0.00    0.00    0.000  0.00     0
5    ...  18.26    89    69   608.28   200.00  150.00   52.220  4.06     0
459  ...  43.89   133    85   850.18   265.17   96.93   90.130  8.77     0
873  ...   8.86    82    50   360.00    90.00   67.50   45.000  5.33     0

        f_47
511   8424.77
413   5147.43
346   4612.24
642   8942.39
425   4065.36
18    6165.56
36    3378.60
225   5066.66
331   3009.93
764   2326.02
184   5158.25
259   5606.99
634   7133.13
97    4736.75
92    7584.28
729   5509.43
539   8487.58
5    18163.97
459  12460.92
873   2859.39

[20 rows x 44 columns]
```

In [82]:
```
model_pred = model.predict(data)
print(model_pred)
print(len(model_pred))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
20
```