APPENDIX

IMPORT LIBRARY

```
1. import numpy as np
2. import pandas as pd
```

DATA COLLECTION

```
3. df = pd.read_csv('train.csv')
4. df
5. df.info()
6. df.drop('Loan_ID', axis=1, inplace=True)
7. df
8. df.describe()
9. df.describe(include=['O'])
```

EDA

```
10. print(df['Loan_Status'].value_counts())
11. print(df['Gender'].value_counts())
12. print('\n')
13. print('Male and Loan Status accepted:', df.loc[(df['Gender'] == '
    Male') & (df['Loan_Status'] == 'Y')].shape[0])
14. print('Male and Loan Status not accepted:', df.loc[(df['Gender']
    == 'Male') & (df['Loan_Status'] == 'N')].shape[0])
15. print('Female and Loan Status accepted:', df.loc[(df['Gender'] ==
     'Female') & (df['Loan_Status'] == 'Y')].shape[0])
16. print('Female and Loan Status not accepted:', df.loc[(df['Gender'
    ] == 'Female') & (df['Loan_Status'] == 'N')].shape[0])
17. print(df['Married'].value_counts())
18. print('\n')
19. print('Married and Loan Status accepted:', df.loc[(df['Married']
    == 'Yes') & (df['Loan_Status']  == 'Y')].shape[0])
20. print('Married and Loan Status not accepted:', df.loc[(df['Marrie
    d'] == 'Yes') & (df['Loan_Status']  == 'N')].shape[0])
21. print('Not Married and Loan Status accepted:', df.loc[(df['Marrie
    d'] == 'No') & (df['Loan_Status']  == 'Y')].shape[0])
22. print('Not Married and Loan Status not accepted:', df.loc[(df['Ma
    rried'] == 'No') & (df['Loan_Status']  == 'N')].shape[0])
```

```
23. print(df['Dependents'].value_counts())
24. print('\n')
25. print('Dependents 0 and Loan Status accepted:', df.loc[(df['Depen
    dents'] == '0') & (df['Loan_Status'] == 'Y')].shape[0])
26. print('Dependents 0 and Loan Status not accepted:', df.loc[(df['D
    ependents'] == '0') & (df['Loan_Status'] == 'N')].shape[0])
27. print('Dependents 1 and Loan Status accepted:', df.loc[(df['Depen
    dents'] == '1') & (df['Loan_Status'] == 'Y')].shape[0])
28. print('Dependents 1 and Loan Status not accepted:', df.loc[(df['D
    ependents'] == '1') & (df['Loan_Status'] == 'N')].shape[0])
29. print('Dependents 2 and Loan Status accepted:', df.loc[(df['Depen
    dents'] == '2') & (df['Loan_Status'] == 'Y')].shape[0])
30. print('Dependents 2 and Loan Status not accepted:', df.loc[(df['D
    ependents'] == '2') & (df['Loan_Status'] == 'N')].shape[0])
31. print('Dependents 3 and Loan Status accepted:', df.loc[(df['Depen
    dents'] == '3+') & (df['Loan_Status'] == 'Y')].shape[0])
32. print('Dependents 3 and Loan Status not accepted:', df.loc[(df['D
    ependents'] == '3+') & (df['Loan_Status'] == 'N')].shape[0])
33. print(df['Education'].value_counts())
34. print('\n')
35. print('Graduate and Loan Status accepted:', df.loc[(df['Education
    '] == 'Graduate') & (df['Loan_Status'] == 'Y')].shape[0])
36. print('Graduate and Loan Status not accepted:', df.loc[(df['Educa
    tion'] == 'Graduate') & (df['Loan_Status'] == 'N')].shape[0])
37. print('Not Graduate and Loan Status accepted:', df.loc[(df['Educa
    tion'] == 'Not Graduate') & (df['Loan_Status'] == 'Y')].shape[0])
38. print('Not Graduate and Loan Status not accepted:', df.loc[(df['E
    ducation'] == 'Not Graduate') & (df['Loan_Status'] == 'N')].shape
    [0])
39. print(df['Self_Employed'].value_counts())
40. print('\n')
41. print('Self Employed and Loan Status accepted:', df.loc[(df['Self
    _Employed'] == 'Yes') & (df['Loan_Status'] == 'Y')].shape[0])
42. print('Self Employed and Loan Status not accepted:', df.loc[(df['
    Self_Employed'] == 'Yes') & (df['Loan_Status'] == 'N')].shape[0])
43. print('Not Self Employed and Loan Status accepted:', df.loc[(df['
    Self_Employed'] == 'No') & (df['Loan_Status'] == 'Y')].shape[0])
```

```
print('Not Self Employed and Loan Status not accepted:', df.loc[(df['Self_Employed'] == 'No') & (df['Loan_Status'] == 'N')].shape[0])
print(df['Credit_History'].value_counts())
print('\n')
print('Credit History 1 and Loan Status accepted:', df.loc[(df['Credit_History'] == 1.0) & (df['Loan_Status'] == 'Y')].shape[0])
print('Credit History 1 and Loan Status not accepted:', df.loc[(df['Credit_History'] == 1.0) & (df['Loan_Status'] == 'N')].shape[0])
print('Credit History 0 and Loan Status accepted:', df.loc[(df['Credit_History'] == 0.0) & (df['Loan_Status'] == 'Y')].shape[0])
print('Credit History 0 and Loan Status not accepted:', df.loc[(df['Credit_History'] == 0.0) & (df['Loan_Status'] == 'N')].shape[0])
print(df['Property_Area'].value_counts())
print('\n')
print('Urban and Loan Status accepted:', df.loc[(df['Property_Area'] == 'Urban') & (df['Loan_Status'] == 'Y')].shape[0])
print('Urban and Loan Status not accepted:', df.loc[(df['Property_Area'] == 'Urban') & (df['Loan_Status'] == 'N')].shape[0])
print('SemiUrban and Loan Status accepted:', df.loc[(df['Property_Area'] == 'Semiurban') & (df['Loan_Status'] == 'Y')].shape[0])
print('SemiUrban and Loan Status not accepted:', df.loc[(df['Property_Area'] == 'Semiurban') & (df['Loan_Status'] == 'N')].shape[0])
print('Rural and Loan Status accepted:', df.loc[(df['Property_Area'] == 'Rural') & (df['Loan_Status'] == 'Y')].shape[0])
print('Rural and Loan Status not accepted:', df.loc[(df['Property_Area'] == 'Rural') & (df['Loan_Status'] == 'N')].shape[0])
print(df['Loan_Amount_Term'].value_counts())
print('\n')
print('Accepted Loan Amount Term:')
print(df.loc[df['Loan_Status'] == 'Y']['Loan_Amount_Term'].value_counts())
print('\n')
print('Declined Loan Amount Term:')
```

```
print(df.loc[df['Loan_Status'] == 'N']['Loan_Amount_Term'].value_counts())
print('Minimum Applicant Income:', df['ApplicantIncome'].min())
print('Maximum Applicant Income:', df['ApplicantIncome'].max())
print('Mean Applicant Income:', df['ApplicantIncome'].mean())
print('\n')
print('Accepted Applicant Income:')
print(df.loc[df['Loan_Status'] == 'Y']['ApplicantIncome'].value_counts())
print('\n')
print('Declined Applicant Income:')
print(df.loc[df['Loan_Status'] == 'N']['ApplicantIncome'].value_counts())
print('Minimum Coapplicant Income:', df['CoapplicantIncome'].min())
print('Maximum Coapplicant Income:', df['CoapplicantIncome'].max())
print('Mean Coapplicant Income:', df['CoapplicantIncome'].mean())
print('\n')
print('Accepted Coapplicant Income:')
print(df.loc[df['Loan_Status'] == 'Y']['CoapplicantIncome'].value_counts())
print('\n')
print('Declined Coapplicant Income:')
print(df.loc[df['Loan_Status'] == 'N']['CoapplicantIncome'].value_counts())
print('Minimum Loan Amount:', df['LoanAmount'].min())
print('Maximum Loan Amount:', df['LoanAmount'].max())
print('\n')
print('Accepted Loan Amount:')
print(df.loc[df['Loan_Status'] == 'Y']['LoanAmount'].value_counts())
print('\n')
print('Declined Loan Amount:')
print(df.loc[df['Loan_Status'] == 'N']['LoanAmount'].value_counts())
```

DATA CLEANING

```
df.isna().sum()
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
df.isna().sum()
df
```

LABEL ENCODING

```
dic = {"N": 0, "Y": 1}
df = df.replace({"Loan_Status": dic})
df
```

FEATURES SELECTION

```
corr_mat = df.corr()
corr_mat['Loan_Status'] = abs(corr_mat['Loan_Status'])
sorted_corr_mat = corr_mat.sort_values(by=['Loan_Status'], ascending=False)
sorted_corr_mat['Loan_Status']
```

CATEGORICAL ENCODING

```
df['Dependents'].unique()
df['Education'].unique()
df['Property_Area'].unique()
dic = {"0": 0, "1": 1, "2": 2, "3+": 3}
df = df.replace({"Dependents": dic})
df
dic = {"Not Graduate": 0, "Graduate": 1}
df = df.replace({"Education": dic})
df
dic = {"Rural": 0, "Semiurban": 1, "Urban": 2}
df = df.replace({"Property_Area": dic})
```

```
df
catg = ["Gender", "Married", "Self_Employed"]
df = pd.get_dummies(df, columns=catg)
df
```

SPLITTING DATA

```
df['Loan_Status'].value_counts()
df_yes = df.loc[df['Loan_Status'] == 1].reset_index(drop=Tru
e)
df_no = df.loc[df['Loan_Status'] == 0].reset_index(drop=True
)
df_yes.info()
df_no.info()
df_yes
df_no
df_yes_train1 = df_yes.loc[df_yes.index[range(0, 253)]]
df_yes_test1 = df_yes.loc[df_yes.index[range(253, 422)]]
df_yes_test1_1 = df_yes.loc[df_yes.index[range(253, 309)]]
df_yes_test1_2 = df_yes.loc[df_yes.index[range(309, 365)]]
df_yes_test1_3 = df_yes.loc[df_yes.index[range(365, 422)]]
df_no_train1 = df_no.loc[df_no.index[range(0, 115)]]
df_no_test1 = df_no.loc[df_no.index[range(115, 192)]]
df_no_test1_1 = df_no.loc[df_no.index[range(115, 141)]]
df_no_test1_2 = df_no.loc[df_no.index[range(141, 167)]]
df_no_test1_3 = df_no.loc[df_no.index[range(167, 192)]]
df_yes_train2 = df_yes.loc[df_yes.index[range(0, 295)]]
df_yes_test2 = df_yes.loc[df_yes.index[range(295, 422)]]
df_yes_test2_1 = df_yes.loc[df_yes.index[range(295, 337)]]
df_yes_test2_2 = df_yes.loc[df_yes.index[range(337, 379)]]
df_yes_test2_3 = df_yes.loc[df_yes.index[range(379, 422)]]
df_no_train2 = df_no.loc[df_no.index[range(0, 134)]]
df_no_test2 = df_no.loc[df_no.index[range(134, 192)]]
df_no_test2_1 = df_no.loc[df_no.index[range(134, 153)]]
df_no_test2_2 = df_no.loc[df_no.index[range(153, 172)]]
df_no_test2_3 = df_no.loc[df_no.index[range(172, 192)]]
df_yes_train1
```

```
152.      df_train1 = pd.concat([df_yes_train1, df_no_train1], ignore_
    index=True)
153.      df_train2 = pd.concat([df_yes_train2, df_no_train2], ignore_
    index=True)
154.      df_test1 = pd.concat([df_yes_test1, df_no_test1], ignore_ind
    ex=True)
155.      df_test1_1 = pd.concat([df_yes_test1_1, df_no_test1_1], igno
    re_index=True)
156.      df_test1_2 = pd.concat([df_yes_test1_2, df_no_test1_2], igno
    re_index=True)
157.      df_test1_3 = pd.concat([df_yes_test1_3, df_no_test1_3], igno
    re_index=True)
158.      df_test2 = pd.concat([df_yes_test2, df_no_test2], ignore_ind
    ex=True)
159.      df_test2_1 = pd.concat([df_yes_test2_1, df_no_test2_1], igno
    re_index=True)
160.      df_test2_2 = pd.concat([df_yes_test2_2, df_no_test2_2], igno
    re_index=True)
161.      df_test2_3 = pd.concat([df_yes_test2_3, df_no_test2_3], igno
    re_index=True)
162.      df_train1 = df_train1.sample(frac=1).reset_index(drop=True)
163.      df_train2 = df_train2.sample(frac=1).reset_index(drop=True)
164.      df_test1 = df_test1.sample(frac=1).reset_index(drop=True)
165.      df_test1_1 = df_test1_1.sample(frac=1).reset_index(drop=True
    )
166.      df_test1_2 = df_test1_2.sample(frac=1).reset_index(drop=True
    )
167.      df_test1_3 = df_test1_3.sample(frac=1).reset_index(drop=True
    )
168.      df_test2 = df_test2.sample(frac=1).reset_index(drop=True)
169.      df_test2_1 = df_test2_1.sample(frac=1).reset_index(drop=True
    )
170.      df_test2_2 = df_test2_2.sample(frac=1).reset_index(drop=True
    )
171.      df_test2_3 = df_test2_3.sample(frac=1).reset_index(drop=True
    )
172.      df_train1
```

```
df_train1.columns
var_input = ['Dependents', 'Education', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area','Gender_Female', 'Gender_Male', 'Married_No','Married_Yes', 'Self_Employed_No', 'Self_Employed_Yes']
X_train1 = df_train1[var_input]
y_train1 = df_train1['Loan_Status']
X_train2 = df_train2[var_input]
y_train2 = df_train2['Loan_Status']
X_test1 = df_test1[var_input]
y_test1 = df_test1['Loan_Status']
X_test1_1 = df_test1_1[var_input]
y_test1_1 = df_test1_1['Loan_Status']
X_test1_2 = df_test1_2[var_input]
y_test1_2 = df_test1_2['Loan_Status']
X_test1_3 = df_test1_3[var_input]
y_test1_3 = df_test1_3['Loan_Status']
X_test2 = df_test2[var_input]
y_test2 = df_test2['Loan_Status']
X_test2_1 = df_test2_1[var_input]
y_test2_1 = df_test2_1['Loan_Status']
X_test2_2 = df_test2_2[var_input]
y_test2_2 = df_test2_2['Loan_Status']
X_test2_3 = df_test2_3[var_input]
y_test2_3 = df_test2_3['Loan_Status']
X_train1, y_train1
```

MODEL BUILDING

CLASSIFICATION REPORT

```
    accuracy = np.sum(true == pred) / len(true)
    return accuracy
def precision(true, pred):
    tp = 0
    fp = 0
    for i in range(len(true)):
      if (true[i] == pred[i] and true[i] == 1):
        tp = tp + 1
```

```
            if (true[i] != pred[i] and true[i] == 0):
              fp = fp + 1
          precision = tp / (tp + fp)
          return precision
        def recall(true, pred):
          tp = 0
          fn = 0
          for i in range(len(true)):
            if (true[i] == pred[i] and true[i] == 1):
              tp = tp + 1
            if (true[i] != pred[i] and true[i] == 1):
              fn = fn + 1
          recall = tp / (tp + fn)
          return recall
        def f1(true, pred):
          tp = 0
          fp = 0
          fn = 0
          for i in range(len(true)):
            if (true[i] == pred[i] and true[i] == 1):
              tp = tp + 1
            if (true[i] != pred[i] and true[i] == 0):
              fp = fp + 1
            if (true[i] != pred[i] and true[i] == 1):
              fn = fn + 1
          precision = tp / (tp + fp)
          recall = tp / (tp + fn)
          f1 = 2 * (recall * precision) / (recall + precision)
          return f1
```

LOGISTIC REGRESSION

```
        class LogisticRegression:
            def __init__(self, learning_rate=0.01, num_iterations=50
    000, fit_intercept=True, verbose=False):
                self.learning_rate = learning_rate
                self.num_iterations = num_iterations
                self.fit_intercept = fit_intercept
```

```
238.             self.verbose = verbose
239.         def b_intercept(self, X):
240.             intercept = np.ones((X.shape[0], 1))
241.             return np.concatenate((intercept, X), axis=1)
242.         def sigmoid_function(self, z):
243.             return 1 / (1 + np.exp(-z))
244.         def loss(self, yp, y):
245.             return (-
    y * np.log(yp) - (1 - y) * np.log(1 - yp)).mean()
246.         def fit(self, X, y):
247.             if self.fit_intercept:
248.                 X = self.b_intercept(X)
249.             self.W = np.zeros(X.shape[1])
250.             self.b = 0
251.             print("----- Proses training -----\n")
252.             for i in range(self.num_iterations):
253.                 z = np.dot(X, self.W) + self.b
254.                 yp = self.sigmoid_function(z)
255.                 gradient_w = np.dot(X.T, (yp - y)) / y.size
256.                 gradient_b = np.sum((yp - y)) / y.size
257.                 self.W -= self.learning_rate * gradient_w
258.                 self.b -= self.learning_rate * gradient_b
259.                 z = np.dot(X, self.W) + self.b
260.                 yp = self.sigmoid_function(z)
261.                 loss = self.loss(yp, y)
262.                 if(self.verbose == True and i % 10000 == 0):
263.                     print("--- loss: {:.6f} ---".format(loss))
264.         def predict_prob(self, X):
265.             if self.fit_intercept:
266.                 X = self.b_intercept(X)
267.             return self.sigmoid_function(np.dot(X, self.W) + sel
    f.b)
268.         def predict(self, X):
269.             return self.predict_prob(X).round()
270.     model = LogisticRegression(learning_rate=0.0000025, num_iter
    ations=500000, verbose=True)
271.     pred = model.predict(X_test1)
```

```
272.      print('\n')
273.      print('accuracy :', accuracy(y_test1, pred))
274.      print('precision :', precision(y_test1, pred))
275.      print('recall :', recall(y_test1, pred))
276.      print('f1-score :', f1(y_test1, pred))
277.      pred = model.predict(X_test1_1)
278.      print('accuracy :', accuracy(y_test1_1, pred))
279.      print('precision :', precision(y_test1_1, pred))
280.      print('recall :', recall(y_test1_1, pred))
281.      print('f1-score :', f1(y_test1_1, pred))
282.      pred = model.predict(X_test1_3)
283.      print('accuracy :', accuracy(y_test1_3, pred))
284.      print('precision :', precision(y_test1_3, pred))
285.      print('recall :', recall(y_test1_3, pred))
286.      print('f1-score :', f1(y_test1_3, pred))
287.      model = LogisticRegression(learning_rate=0.0000025,num_itera
  tions=50000, verbose=True)
288.      model.fit(X_train1, y_train1)
289.      pred = model.predict(X_test1)
290.      print('\n')
291.      print('accuracy :', accuracy(y_test1, pred))
292.      print('precision :', precision(y_test1, pred))
293.      print('recall :', recall(y_test1, pred))
294.      print('f1-score :', f1(y_test1, pred))
295.      pred = model.predict(X_test1_1)
296.      print('accuracy :', accuracy(y_test1_1, pred))
297.      print('precision :', precision(y_test1_1, pred))
298.      print('recall :', recall(y_test1_1, pred))
299.      print('f1-score :', f1(y_test1_1, pred))
300.      pred = model.predict(X_test1_2)
301.      print('accuracy :', accuracy(y_test1_2, pred))
302.      print('precision :', precision(y_test1_2, pred))
303.      print('recall :', recall(y_test1_2, pred))
304.      print('f1-score :', f1(y_test1_2, pred))
305.      pred = model.predict(X_test1_3)
306.      print('accuracy :', accuracy(y_test1_3, pred))
307.      print('precision :', precision(y_test1_3, pred))
```

```
308.      print('recall :', recall(y_test1_3, pred))
309.      print('f1-score :', f1(y_test1_3, pred))
310.      model = LogisticRegression(learning_rate=0.000001, num_itera
   tions=500000, verbose=True)
311.      model.fit(X_train1, y_train1)
312.      pred = model.predict(X_test1)
313.      print('\n')
314.      print('accuracy :', accuracy(y_test1, pred))
315.      print('precision :', precision(y_test1, pred))
316.      print('recall :', recall(y_test1, pred))
317.      print('f1-score :', f1(y_test1, pred))
318.      pred = model.predict(X_test1_1)
319.      print('accuracy :', accuracy(y_test1_1, pred))
320.      print('precision :', precision(y_test1_1, pred))
321.      print('recall :', recall(y_test1_1, pred))
322.      print('f1-score :', f1(y_test1_1, pred))
323.      pred = model.predict(X_test1_2)
324.      print('accuracy :', accuracy(y_test1_2, pred))
325.      print('precision :', precision(y_test1_2, pred))
326.      print('recall :', recall(y_test1_2, pred))
327.      print('f1-score :', f1(y_test1_2, pred))
328.      pred = model.predict(X_test1_3)
329.      print('accuracy :', accuracy(y_test1_3, pred))
330.      print('precision :', precision(y_test1_3, pred))
331.      print('recall :', recall(y_test1_3, pred))
332.      print('f1-score :', f1(y_test1_3, pred))
333.      model = LogisticRegression(learning_rate=0.000001, num_itera
   tions=50000, verbose=True)
334.      model.fit(X_train1, y_train1)
335.      pred = model.predict(X_test1)
336.      print('\n')
337.      print('accuracy :', accuracy(y_test1, pred))
338.      print('precision :', precision(y_test1, pred))
339.      print('recall :', recall(y_test1, pred))
340.      print('f1-score :', f1(y_test1, pred))
341.      pred = model.predict(X_test1_1)
342.      print('accuracy :', accuracy(y_test1_1, pred))
```

```
343.      print('precision :', precision(y_test1_1, pred))
344.      print('recall :', recall(y_test1_1, pred))
345.      print('f1-score :', f1(y_test1_1, pred))
346.      pred = model.predict(X_test1_2)
347.      print('accuracy :', accuracy(y_test1_2, pred))
348.      print('precision :', precision(y_test1_2, pred))
349.      print('recall :', recall(y_test1_2, pred))
350.      print('f1-score :', f1(y_test1_2, pred))
351.      pred = model.predict(X_test1_3)
352.      print('accuracy :', accuracy(y_test1_3, pred))
353.      print('precision :', precision(y_test1_3, pred))
354.      print('recall :', recall(y_test1_3, pred))
355.      print('f1-score :', f1(y_test1_3, pred))
356.      model = LogisticRegression(learning_rate=0.0000001, num_iter
   ations=500000, verbose=True)
357.      model.fit(X_train1, y_train1)
358.      pred = model.predict(X_test1)
359.      print('\n')
360.      print('accuracy :', accuracy(y_test1, pred))
361.      print('precision :', precision(y_test1, pred))
362.      print('recall :', recall(y_test1, pred))
363.      print('f1-score :', f1(y_test1, pred))
364.      pred = model.predict(X_test1_1)
365.      print('accuracy :', accuracy(y_test1_1, pred))
366.      print('precision :', precision(y_test1_1, pred))
367.      print('recall :', recall(y_test1_1, pred))
368.      print('f1-score :', f1(y_test1_1, pred))
369.      pred = model.predict(X_test1_2)
370.      print('accuracy :', accuracy(y_test1_2, pred))
371.      print('precision :', precision(y_test1_2, pred))
372.      print('recall :', recall(y_test1_2, pred))
373.      print('f1-score :', f1(y_test1_2, pred))
374.      pred = model.predict(X_test1_3)
375.      print('accuracy :', accuracy(y_test1_3, pred))
376.      print('precision :', precision(y_test1_3, pred))
377.      print('recall :', recall(y_test1_3, pred))
378.      print('f1-score :', f1(y_test1_3, pred))
```

```
379.      model = LogisticRegression(learning_rate=0.0000001, num_iter
   ations=50000, verbose=True)
380.      model.fit(X_train1, y_train1)
381.      pred = model.predict(X_test1)
382.      print('\n')
383.      print('accuracy :', accuracy(y_test1, pred))
384.      print('precision :', precision(y_test1, pred))
385.      print('recall :', recall(y_test1, pred))
386.      print('f1-score :', f1(y_test1, pred))
387.      pred = model.predict(X_test1_1)
388.      print('accuracy :', accuracy(y_test1_1, pred))
389.      print('precision :', precision(y_test1_1, pred))
390.      print('recall :', recall(y_test1_1, pred))
391.      print('f1-score :', f1(y_test1_1, pred))
392.      pred = model.predict(X_test1_2)
393.      print('accuracy :', accuracy(y_test1_2, pred))
394.      print('precision :', precision(y_test1_2, pred))
395.      print('recall :', recall(y_test1_2, pred))
396.      print('f1-score :', f1(y_test1_2, pred))
397.      pred = model.predict(X_test1_3)
398.      print('accuracy :', accuracy(y_test1_3, pred))
399.      print('precision :', precision(y_test1_3, pred))
400.      print('recall :', recall(y_test1_3, pred))
401.      print('f1-score :', f1(y_test1_3, pred))
```

EXTREME GRADIENT BOOSTING

```
402.      def sigmoid(x):
403.          return 1 / (1 + np.exp(-x))
404.      def grad(preds, labels):
405.          preds = sigmoid(preds)
406.          return (preds - labels)
407.      def hess(preds, labels):
408.          preds = sigmoid(preds)
409.          return (preds * (1 - preds))
410.      class TreeNode(object):
```

```
411.          def __init__(self, is_leaf=False, leaf_score=None, split
    _feature=None, split_threshold=None, left_child=None, right_child
    =None, NA_direction='left'):
412.              self.is_leaf = is_leaf
413.              self.leaf_score = leaf_score
414.              self.split_feature = split_feature
415.              self.split_threshold = split_threshold
416.              self.left_child = left_child
417.              self.right_child = right_child
418.              self.NA_direction = NA_direction
419.      class Tree(object):
420.          def __init__(self, root=None, min_sample_split=None, col
    _sub_frac=None, lamda=None, gamma=None, num_thread=None, min_chil
    d_weight=None):
421.              self.root = root
422.              self.min_sample_split = min_sample_split
423.              self.col_sub_frac = col_sub_frac
424.              self.lamda = lamda
425.              self.gamma = gamma
426.              self.min_child_weight = min_child_weight
427.          def cal_leaf_score(self, Y):
428.              return - (Y['grad'].sum() / (Y['hess'].sum() + self.
    lamda))
429.          def cal_split_gain(self, left_Y, right_Y, NA_grad, NA_he
    ss, NA_direction='left'):
430.              if (NA_direction == 'left'):
431.                  GL = left_Y['grad'].sum() + NA_grad
432.                  HL = left_Y['hess'].sum() + NA_hess
433.                  GR = right_Y['grad'].sum()
434.                  HR = right_Y['hess'].sum()
435.              else:
436.                  GL = left_Y['grad'].sum()
437.                  HL = left_Y['hess'].sum()
438.                  GR = right_Y['grad'].sum() + NA_grad
439.                  HR = right_Y['hess'].sum() + NA_hess
440.              gain = 0.5 * ( (GL**2/(HL+self.lamda)) + (GR**2/(HR+
    self.lamda)) - ((GL+GR)**2/(HL+HR+self.lamda)) ) - self.gamma
```

```
441.                 return gain
442.             def weighted_quantile_sketch(self, dt, feature):
443.                 best_split_value = None
444.                 best_gain = -np.inf
445.                 best_NA_direction = 'left'
446.                 selected_dt = dt[[feature, 'label', 'grad', 'hess']
447.                 mask = selected_dt[feature].isnull()
448.                 NA_dt = selected_dt[mask]
449.                 Non_NA_dt = selected_dt[~mask]
450.                 NA_grad = NA_dt['grad'].sum()
451.                 NA_hess = NA_dt['hess'].sum()
452.                 Non_NA_dt.reset_index(inplace = True)
453.                 Non_NA_dt['feature_index'] = Non_NA_dt[feature].args
    ort()
454.                 Non_NA_dt = Non_NA_dt.iloc[Non_NA_dt['feature_index'
    ]]
455.                 hess_sum = Non_NA_dt['hess'].sum()
456.                 Non_NA_dt['rank'] = Non_NA_dt.apply(lambda x : (1/he
    ss_sum)*sum(Non_NA_dt[Non_NA_dt[feature] < x[feature]]['hess']),
    axis=1)
457.                 for j in range(Non_NA_dt.shape[0]-1):
458.                     rk_sk_j, rk_sk_j_1 = Non_NA_dt['rank'].iloc[j:j+
    2]
459.                     if (abs(rk_sk_j-rk_sk_j_1) >= self.eps):
460.                         continue
461.                     split_value = (Non_NA_dt[feature].iloc[j+1] + No
    n_NA_dt[feature].iloc[j])/2
462.                     left_Y = Non_NA_dt.iloc[:(j+1)]
463.                     right_Y = Non_NA_dt.iloc[(j+1):]
464.                     go_left = self.cal_split_gain(left_Y, right_Y, N
    A_grad, NA_hess, NA_direction = 'left')
465.                     go_right = self.cal_split_gain(left_Y, right_Y,
    NA_grad, NA_hess, NA_direction = 'right')
466.                     if (go_left > go_right):
467.                         this_gain = go_left
468.                         this_direction = 'left'
469.                     else:
```

```
                    this_gain = go_right
                    this_direction = 'right'
                if (this_gain > best_gain):
                    best_split_value = split_value
                    best_gain = this_gain
                    best_NA_direction = this_direction
            return feature, best_split_value, best_gain, best_NA
_direction
        def find_best_split_value_and_feature(self, X, Y):
            best_gain = -np.inf
            best_feature, best_split_value, results = None, None
, None
            features = list(X.columns)
            data = pd.concat([X, Y], axis = 1)
            results = []
            for j in range(len(features)):
                results.append(self.weighted_quantile_sketch(dat
a, features[j]))
            best = sorted(results, key = lambda x: float(x[2]),
reverse = True)[0]
            best_feature = best[0]
            best_split_value = best[1]
            best_gain = best[2]
            best_NA_direction = best[3]
            return best_feature, best_split_value, best_gain, be
st_NA_direction
        def split(self, X, Y, feature, split_value, NA_direction
):
            data = pd.concat([X, Y], axis = 1)
            X_cols, Y_cols = list(X.columns), list(Y.columns)
            if(NA_direction == 'left'):
                mask = (data[feature] >= split_value)
                left = data[~mask]
                right = data[mask]
            else:
                mask = (data[feature] < split_value)
                left = data[mask]
```

```
501.                 right = data[~mask]
502.             return left[X_cols], left[Y_cols], right[X_cols], ri
    ght[Y_cols]
503.         def build_tree(self, X, Y, depth):
504.             if (X.shape[0] < self.min_sample_split) or (depth ==
     0) or (Y['hess'].sum() < self.min_child_weight):
505.                 l_score = self.cal_leaf_score(Y)
506.                 return TreeNode(is_leaf=True, leaf_score=l_score
    )
507.             X_sub = X.sample(frac=self.col_sub_frac, axis=1)
508.             best_feature, best_split_value, best_gain, best_NA_d
    irection = self.find_best_split_value_and_feature(X_sub, Y)
509.             print("----- Best split value and feature -----
    \n", best_feature, best_split_value, best_gain, best_NA_direction
    )
510.             if (best_gain <= 0):
511.                 l_score = self.cal_leaf_score(Y)
512.                 return TreeNode(is_leaf=True, leaf_score=l_score
    )
513.             left_X, left_Y, right_X, right_Y = self.split(X_sub,
     Y, best_feature, best_split_value, best_NA_direction)
514.             left_child = self.build_tree(left_X, left_Y, depth -
     1)
515.             right_child = self.build_tree(right_X, right_Y, dept
    h - 1)
516.             sub_tree = TreeNode(is_leaf=False, leaf_score=None,
    split_feature=best_feature, split_threshold=best_split_value, lef
    t_child=left_child, right_child=right_child, NA_direction=best_NA
    _direction)
517.             return sub_tree
518.         def fit(self, X, Y, max_depth=3, min_child_weight=1, col
    _sub_frac=1, min_sample_split=10, lamda=1, gamma=0.05, eps=0.001)
    :
519.             self.min_child_weight = min_child_weight
520.             self.col_sub_frac = col_sub_frac
521.             self.min_sample_split = min_sample_split
522.             self.lamda = lamda
```

```
523.                 self.gamma = gamma
524.                 self.eps = eps
525.                 self.root = self.build_tree(X, Y, max_depth)
526.             def predict_one(self, tree_node, X):
527.                 if tree_node.is_leaf == True:
528.                     return tree_node.leaf_score
529.                 elif (type(X[tree_node.split_feature].item()) != int
    ) and (type(X[tree_node.split_feature].item()) != float) and (tre
    e_node.NA_direction == 'left'):
530.                     return self.predict_one(tree_node.left_child, X)
    ]
531.                 elif ((X[tree_node.split_feature] < tree_node.split_
    threshold).item()):
532.                     return self.predict_one(tree_node.left_child, X)
533.                 else:
534.                     return self.predict_one(tree_node.right_child, X
    )
535.             def predict(self, X):
536.                 preds = []
537.                 for n in range(X.shape[0]):
538.                     preds.append(self.predict_one(self.root, X.iloc[
    [n]]))
539.                 return np.array(preds)
540.         def xgboost_train(X, Y, eta, max_round, max_depth, row_sub_f
    rac, col_sub_frac, min_child_weight, min_sample_split, lamda, gam
    ma, eps, metric):
541.             trees = []
542.             initialize_pred = 1
543.             best_metric_value, best_round = -np.inf, None
544.             metric_value_list = []
545.             X.reset_index(drop=True, inplace=True)
546.             Y = Y.to_frame(name='label')
547.             Y.reset_index(drop=True, inplace=True)
548.             Y['y_pred'] = initialize_pred
549.             Y['grad'] = grad(Y['y_pred'], Y['label'])
550.             Y['hess'] = hess(Y['y_pred'], Y['label'])
551.             for i in range(max_round):
```

```
552.                data = pd.concat([X, Y], axis=1)
553.                data = data.sample(frac=row_sub_frac, axis=0)
554.                print("----- Data sub-sampling -----\n", data)
555.                Y_Selected = data[['label', 'y_pred', 'grad', 'hess'
    ]]
556.                X_Selected = data[list(X.columns)]
557.                tree = Tree()
558.                tree.fit(X_Selected, Y_Selected, max_depth=max_depth
    , min_child_weight=min_child_weight, col_sub_frac=col_sub_frac, m
    in_sample_split=min_sample_split, lamda=lamda, gamma=gamma, eps=e
    ps)
559.                preds = tree.predict(X)
560.                Y['y_pred'] = Y['y_pred'] + eta * preds
561.                Y['grad'] = grad(Y['y_pred'], Y['label'])
562.                Y['hess'] = hess(Y['y_pred'], Y['label'])
563.                trees.append(tree)
564.                test_perf = []
565.                avg = Y['y_pred'].mean()
566.                for j in Y['y_pred']:
567.                    if (j > avg):
568.                        test_perf.append(1)
569.                    else:
570.                        test_perf.append(0)
571.                if (metric == 'f1'):
572.                    m = f1(Y['label'], test_perf)
573.                    print("F1-Score: ", m)
574.                if (metric == 'accuracy'):
575.                    m = accuracy(Y['label'], test_perf)
576.                    print("Accuracy: ", m, "\n")
577.                metric_value_list.append(m)
578.                if (m > best_metric_value):
579.                    best_metric_value = m
580.                    best_round = i
581.            best_trees = trees[:(i+1)]
582.            return best_trees, eta
583.        def xgboost_predict(trees, X_test, eta):
584.            preds = np.ones(X_test.shape[0])
```

```
585.            for tree in trees:
586.                preds = preds + tree.predict(X_test) * eta
587.            adj_preds = []
588.            avg = preds.mean()
589.            for i in preds:
590.                if (i > avg):
591.                    adj_preds.append(1)
592.                else:
593.                    adj_preds.append(0)
594.            return adj_preds
595.        model, eta = xgboost_train(X_train1, y_train1,
596.        eta = 0.3,
597.        max_round = 30,
598.        max_depth = 3,
599.        row_sub_frac =
600.        col_sub_frac = 1,
601.        min_child_weight = 1,
602.        min_sample_split = 10,
603.        lamda = 1,
604.        gamma = 0,
605.        eps = 0.003,
606.        metric = 'accuracy')
607.        pred = xgboost_predict(model, X_test1, eta)
608.        print('\n')
609.        print('accuracy :', accuracy(y_test1, pred))
610.        print('precision :', precision(y_test1, pred))
611.        print('recall :', recall(y_test1, pred))
612.        print('f1-score :', f1(y_test1, pred))
613.        pred = xgboost_predict(model, X_test1_1, eta)
614.        print('accuracy :', accuracy(y_test1_1, pred))
615.        print('precision :', precision(y_test1_1, pred))
616.        print('recall :', recall(y_test1_1, pred))
617.        print('f1-score :', f1(y_test1_1, pred))
618.        pred = xgboost_predict(model, X_test1_2, eta)
619.        print('accuracy :', accuracy(y_test1_2, pred))
620.        print('precision :', precision(y_test1_2, pred))
621.        print('recall :', recall(y_test1_2, pred))
```

```
622.      print('f1-score :', f1(y_test1_2, pred))
623.      pred = xgboost_predict(model, X_test1_3, eta)
624.      print('accuracy :', accuracy(y_test1_3, pred))
625.      print('precision :', precision(y_test1_3, pred))
626.      print('recall :', recall(y_test1_3, pred))
627.      print('f1-score :', f1(y_test1_3, pred))
628.      model, eta = xgboost_train(X_train1, y_train1,
629.      eta = 0.4,
630.      max_round = 30,
631.      max_depth = 3,
632.      row_sub_frac =
633.      col_sub_frac = 1,
634.      min_child_weight = 1,
635.      min_sample_split = 10,
636.      lamda = 1,
637.      gamma = 0,
638.      eps = 0.003,
639.      metric = 'accuracy')
640.      pred = xgboost_predict(model, X_test1, eta)
641.      print('\n')
642.      print('accuracy :', accuracy(y_test1, pred))
643.      print('precision :', precision(y_test1, pred))
644.      print('recall :', recall(y_test1, pred))
645.      print('f1-score :', f1(y_test1, pred))
646.      pred = xgboost_predict(model, X_test1_1, eta)
647.      print('accuracy :', accuracy(y_test1_1, pred))
648.      print('precision :', precision(y_test1_1, pred))
649.      print('recall :', recall(y_test1_1, pred))
650.      print('f1-score :', f1(y_test1_1, pred))
651.      pred = xgboost_predict(model, X_test1_2, eta)
652.      print('accuracy :', accuracy(y_test1_2, pred))
653.      print('precision :', precision(y_test1_2, pred))
654.      print('recall :', recall(y_test1_2, pred))
655.      print('f1-score :', f1(y_test1_2, pred))
656.      pred = xgboost_predict(model, X_test1_3, eta)
657.      print('accuracy :', accuracy(y_test1_3, pred))
658.      print('precision :', precision(y_test1_3, pred))
```

```
659.      print('recall :', recall(y_test1_3, pred))
660.      print('f1-score :', f1(y_test1_3, pred))
661.      model, eta = xgboost_train(X_train1, y_train1,
662.      eta = 0.5,
663.      max_round = 30,
664.      max_depth = 3,
665.      row_sub_frac =
666.      col_sub_frac = 1,
667.      min_child_weight = 1,
668.      min_sample_split = 10,
669.      lamda = 1,
670.      gamma = 0,
671.      eps = 0.003,
672.      metric = 'accuracy')
673.      pred = xgboost_predict(model, X_test1, eta)
674.      print('\n')
675.      print('accuracy :', accuracy(y_test1, pred))
676.      print('precision :', precision(y_test1, pred))
677.      print('recall :', recall(y_test1, pred))
678.      print('f1-score :', f1(y_test1, pred))
679.      pred = xgboost_predict(model, X_test1_1, eta)
680.      print('accuracy :', accuracy(y_test1_1, pred))
681.      print('precision :', precision(y_test1_1, pred))
682.      print('recall :', recall(y_test1_1, pred))
683.      print('f1-score :', f1(y_test1_1, pred))
684.      pred = xgboost_predict(model, X_test1_2, eta)
685.      print('accuracy :', accuracy(y_test1_2, pred))
686.      print('precision :', precision(y_test1_2, pred))
687.      print('recall :', recall(y_test1_2, pred))
688.      print('f1-score :', f1(y_test1_2, pred))
689.      pred = xgboost_predict(model, X_test1_3, eta)
690.      print('accuracy :', accuracy(y_test1_3, pred))
691.      print('precision :', precision(y_test1_3, pred))
692.      print('recall :', recall(y_test1_3, pred))
693.      print('f1-score :', f1(y_test1_3, pred))
694.      model, eta = xgboost_train(X_train1, y_train1,
695.      eta = 0.3,
```

```
696.      max_round = 45,
697.      max_depth = 3,
698.      row_sub_frac =
699.      col_sub_frac = 1,
700.      min_child_weight = 1,
701.      min_sample_split = 10,
702.      lamda = 1,
703.      gamma = 0,
704.      eps = 0.003,
705.      metric = 'accuracy')
706.      pred = xgboost_predict(model, X_test1, eta)
707.      print('\n')
708.      print('accuracy :', accuracy(y_test1, pred))
709.      print('precision :', precision(y_test1, pred))
710.      print('recall :', recall(y_test1, pred))
711.      print('f1-score :', f1(y_test1, pred))
712.      pred = xgboost_predict(model, X_test1_1, eta)
713.      print('accuracy :', accuracy(y_test1_1, pred))
714.      print('precision :', precision(y_test1_1, pred))
715.      print('recall :', recall(y_test1_1, pred))
716.      print('f1-score :', f1(y_test1_1, pred))
717.      pred = xgboost_predict(model, X_test1_2, eta)
718.      print('accuracy :', accuracy(y_test1_2, pred))
719.      print('precision :', precision(y_test1_2, pred))
720.      print('recall :', recall(y_test1_2, pred))
721.      print('f1-score :', f1(y_test1_2, pred))
722.      pred = xgboost_predict(model, X_test1_3, eta)
723.      print('accuracy :', accuracy(y_test1_3, pred))
724.      print('precision :', precision(y_test1_3, pred))
725.      print('recall :', recall(y_test1_3, pred))
726.      print('f1-score :', f1(y_test1_3, pred))
727.      model, eta = xgboost_train(X_train1, y_train1,
728.      eta = 0.4,
729.      max_round = 45,
730.      max_depth = 3,
731.      row_sub_frac =
732.      col_sub_frac = 1,
```

```
733.      min_child_weight = 1,
734.      min_sample_split = 10,
735.      lamda = 1,
736.      gamma = 0,
737.      eps = 0.003,
738.      metric = 'accuracy')
739.      pred = xgboost_predict(model, X_test1, eta)
740.      print('\n')
741.      print('accuracy :', accuracy(y_test1, pred))
742.      print('precision :', precision(y_test1, pred))
743.      print('recall :', recall(y_test1, pred))
744.      print('f1-score :', f1(y_test1, pred))
745.      pred = xgboost_predict(model, X_test1_1, eta)
746.      print('accuracy :', accuracy(y_test1_1, pred))
747.      print('precision :', precision(y_test1_1, pred))
748.      print('recall :', recall(y_test1_1, pred))
749.      print('f1-score :', f1(y_test1_1, pred))
750.      pred = xgboost_predict(model, X_test1_2, eta)
751.      print('accuracy :', accuracy(y_test1_2, pred))
752.      print('precision :', precision(y_test1_2, pred))
753.      print('recall :', recall(y_test1_2, pred))
754.      print('f1-score :', f1(y_test1_2, pred))
755.      pred = xgboost_predict(model, X_test1_3, eta)
756.      print('accuracy :', accuracy(y_test1_3, pred))
757.      print('precision :', precision(y_test1_3, pred))
758.      print('recall :', recall(y_test1_3, pred))
759.      print('f1-score :', f1(y_test1_3, pred))
760.      model, eta = xgboost_train(X_train1, y_train1,
761.      eta = 0.5,
762.      max_round = 45,
763.      max_depth = 3,
764.      row_sub_frac =
765.      col_sub_frac = 1,
766.      min_child_weight = 1,
767.      min_sample_split = 10,
768.      lamda = 1,
769.      gamma = 0,
```

```
770.      eps = 0.003,
771.      metric = 'accuracy')
772.      pred = xgboost_predict(model, X_test1, eta)
773.      print('\n')
774.      print('accuracy :', accuracy(y_test1, pred))
775.      print('precision :', precision(y_test1, pred))
776.      print('recall :', recall(y_test1, pred))
777.      print('f1-score :', f1(y_test1, pred))
778.      pred = xgboost_predict(model, X_test1_1, eta)
779.      print('accuracy :', accuracy(y_test1_1, pred))
780.      print('precision :', precision(y_test1_1, pred))
781.      print('recall :', recall(y_test1_1, pred))
782.      print('f1-score :', f1(y_test1_1, pred))
783.      pred = xgboost_predict(model, X_test1_2, eta)
784.      print('accuracy :', accuracy(y_test1_2, pred))
785.      print('precision :', precision(y_test1_2, pred))
786.      print('recall :', recall(y_test1_2, pred))
787.      print('f1-score :', f1(y_test1_2, pred))
788.      pred = xgboost_predict(model, X_test1_3, eta)
789.      print('accuracy :', accuracy(y_test1_3, pred))
790.      print('precision :', precision(y_test1_3, pred))
791.      print('recall :', recall(y_test1_3, pred))
792.      print('f1-score :', f1(y_test1_3, pred))
793.      model, eta = xgboost_train(X_train1, y_train1,
794.      eta = 0.3,
795.      max_round = 60,
796.      max_depth = 3,
797.      row_sub_frac =
798.      col_sub_frac = 1,
799.      min_child_weight = 1,
800.      min_sample_split = 10,
801.      lamda = 1,
802.      gamma = 0,
803.      eps = 0.003,
804.      metric = 'accuracy')
805.      pred = xgboost_predict(model, X_test1, eta)
806.      print('\n')
```

```
print('accuracy :', accuracy(y_test1, pred))
print('precision :', precision(y_test1, pred))
print('recall :', recall(y_test1, pred))
print('f1-score :', f1(y_test1, pred))
pred = xgboost_predict(model, X_test1_1, eta)
print('accuracy :', accuracy(y_test1_1, pred))
print('precision :', precision(y_test1_1, pred))
print('recall :', recall(y_test1_1, pred))
print('f1-score :', f1(y_test1_1, pred))
pred = xgboost_predict(model, X_test1_2, eta)
print('accuracy :', accuracy(y_test1_2, pred))
print('precision :', precision(y_test1_2, pred))
print('recall :', recall(y_test1_2, pred))
print('f1-score :', f1(y_test1_2, pred))
pred = xgboost_predict(model, X_test1_3, eta)
print('accuracy :', accuracy(y_test1_3, pred))
print('precision :', precision(y_test1_3, pred))
print('recall :', recall(y_test1_3, pred))
print('f1-score :', f1(y_test1_3, pred))
model, eta = xgboost_train(X_train1, y_train1,
eta = 0.4,
max_round = 60,
max_depth = 3,
row_sub_frac =
col_sub_frac = 1,
min_child_weight = 1,
min_sample_split = 10,
lamda = 1,
gamma = 0,
eps = 0.003,
metric = 'accuracy')
pred = xgboost_predict(model, X_test1, eta)
print('\n')
print('accuracy :', accuracy(y_test1, pred))
print('precision :', precision(y_test1, pred))
print('recall :', recall(y_test1, pred))
print('f1-score :', f1(y_test1, pred))
```

```
pred = xgboost_predict(model, X_test1_1, eta)
print('accuracy :', accuracy(y_test1_1, pred))
print('precision :', precision(y_test1_1, pred))
print('recall :', recall(y_test1_1, pred))
print('f1-score :', f1(y_test1_1, pred))
pred = xgboost_predict(model, X_test1_2, eta)
print('accuracy :', accuracy(y_test1_2, pred))
print('precision :', precision(y_test1_2, pred))
print('recall :', recall(y_test1_2, pred))
print('f1-score :', f1(y_test1_2, pred))
pred = xgboost_predict(model, X_test1_3, eta)
print('accuracy :', accuracy(y_test1_3, pred))
print('precision :', precision(y_test1_3, pred))
print('recall :', recall(y_test1_3, pred))
print('f1-score :', f1(y_test1_3, pred))
model, eta = xgboost_train(X_train1, y_train1,
eta = 0.5,
max_round = 60,
max_depth = 3,
row_sub_frac =
col_sub_frac = 1,
min_child_weight = 1,
min_sample_split = 10,
lamda = 1,
gamma = 0,
eps = 0.003,
metric = 'accuracy')
pred = xgboost_predict(model, X_test1, eta)
print('\n')
print('accuracy :', accuracy(y_test1, pred))
print('precision :', precision(y_test1, pred))
print('recall :', recall(y_test1, pred))
print('f1-score :', f1(y_test1, pred))
pred = xgboost_predict(model, X_test1_1, eta)
print('accuracy :', accuracy(y_test1_1, pred))
print('precision :', precision(y_test1_1, pred))
print('recall :', recall(y_test1_1, pred))
```

```
881.      print('f1-score :', f1(y_test1_1, pred))
882.      pred = xgboost_predict(model, X_test1_2, eta)
883.      print('accuracy :', accuracy(y_test1_2, pred))
884.      print('precision :', precision(y_test1_2, pred))
885.      print('recall :', recall(y_test1_2, pred))
886.      print('f1-score :', f1(y_test1_2, pred))
887.      pred = xgboost_predict(model, X_test1_3, eta)
888.      print('accuracy :', accuracy(y_test1_3, pred))
889.      print('precision :', precision(y_test1_3, pred))
890.      print('recall :', recall(y_test1_3, pred))
891.      print('f1-score :', f1(y_test1_3, pred))
892.      model, eta = xgboost_train(X_train2, y_train2,
893.      eta = 0.3,
894.      max_round = 30,
895.      max_depth = 3,
896.      row_sub_frac = 0.95,
897.      col_sub_frac = 1,
898.      min_child_weight = 1,
899.      min_sample_split = 10,
900.      lamda = 1,
901.      gamma = 0,
902.      eps = 0.003,
903.      metric = 'accuracy')
904.      pred = xgboost_predict(model, X_test2, eta)
905.      print('\n')
906.      print('accuracy :', accuracy(y_test2, pred))
907.      print('precision :', precision(y_test2, pred))
908.      print('recall :', recall(y_test2, pred))
909.      print('f1-score :', f1(y_test2, pred))
910.      pred = xgboost_predict(model, X_test2_1, eta)
911.      print('accuracy :', accuracy(y_test2_1, pred))
912.      print('precision :', precision(y_test2_1, pred))
913.      print('recall :', recall(y_test2_1, pred))
914.      print('f1-score :', f1(y_test2_1, pred))
915.      pred = xgboost_predict(model, X_test2_2, eta)
916.      print('accuracy :', accuracy(y_test2_2, pred))
917.      print('precision :', precision(y_test2_2, pred))
```

```
print('recall :', recall(y_test2_2, pred))
print('f1-score :', f1(y_test2_2, pred))
pred = xgboost_predict(model, X_test2_3, eta)
print('accuracy :', accuracy(y_test2_3, pred))
print('precision :', precision(y_test2_3, pred))
print('recall :', recall(y_test2_3, pred))
print('f1-score :', f1(y_test2_3, pred))
model, eta = xgboost_train(X_train2, y_train2,
eta = 0.4,
max_round = 30,
max_depth = 3,
row_sub_frac = 0.95,
col_sub_frac = 1,
min_child_weight = 1,
min_sample_split = 10,
lamda = 1,
gamma = 0,
metric = 'accuracy')
pred = xgboost_predict(model, X_test2, eta)
print('\n')
print('accuracy :', accuracy(y_test2, pred))
print('precision :', precision(y_test2, pred))
print('recall :', recall(y_test2, pred))
print('f1-score :', f1(y_test2, pred))
pred = xgboost_predict(model, X_test2_1, eta)
print('accuracy :', accuracy(y_test2_1, pred))
print('precision :', precision(y_test2_1, pred))
print('recall :', recall(y_test2_1, pred))
print('f1-score :', f1(y_test2_1, pred))
pred = xgboost_predict(model, X_test2_2, eta)
print('accuracy :', accuracy(y_test2_2, pred))
print('precision :', precision(y_test2_2, pred))
print('recall :', recall(y_test2_2, pred))
print('f1-score :', f1(y_test2_2, pred))
pred = xgboost_predict(model, X_test2_3, eta)
print('accuracy :', accuracy(y_test2_3, pred))
print('precision :', precision(y_test2_3, pred))
```

```
955.      print('recall :', recall(y_test2_3, pred))
956.      print('f1-score :', f1(y_test2_3, pred))
957.      model, eta = xgboost_train(X_train2, y_train2,
958.      eta = 0.5,
959.      max_round = 30,
960.      max_depth = 3,
961.      row_sub_frac = 0.95,
962.      col_sub_frac = 1,
963.      min_child_weight = 1,
964.      min_sample_split = 10,
965.      lamda = 1,
966.      gamma = 0,
967.      metric = 'accuracy')
968.      pred = xgboost_predict(model, X_test2, eta)
969.      print('\n')
970.      print('accuracy :', accuracy(y_test2, pred))
971.      print('precision :', precision(y_test2, pred))
972.      print('recall :', recall(y_test2, pred))
973.      print('f1-score :', f1(y_test2, pred))
974.      pred = xgboost_predict(model, X_test2_1, eta)
975.      pred = xgboost_predict(model, X_test2_1, eta)
976.      print('accuracy :', accuracy(y_test2_1, pred))
977.      print('precision :', precision(y_test2_1, pred))
978.      print('recall :', recall(y_test2_1, pred))
979.      print('f1-score :', f1(y_test2_1, pred))
980.      pred = xgboost_predict(model, X_test2_2, eta)
981.      print('accuracy :', accuracy(y_test2_2, pred))
982.      print('precision :', precision(y_test2_2, pred))
983.      print('recall :', recall(y_test2_2, pred))
984.      print('f1-score :', f1(y_test2_2, pred))
985.      pred = xgboost_predict(model, X_test2_3, eta)
986.      print('accuracy :', accuracy(y_test2_3, pred))
987.      print('precision :', precision(y_test2_3, pred))
988.      print('recall :', recall(y_test2_3, pred))
989.      print('f1-score :', f1(y_test2_3, pred))
990.      model, eta = xgboost_train(X_train2, y_train2,
991.      eta = 0.3,
```

```
992.      max_round = 45,
993.      max_depth = 3,
994.      row_sub_frac = 0.95,
995.      col_sub_frac = 1,
996.      min_child_weight = 1,
997.      min_sample_split = 10,
998.      lamda = 1,
999.      gamma = 0,
1000.     metric = 'accuracy')
1001.     pred = xgboost_predict(model, X_test2, eta)
1002.     print('\n')
1003.     print('accuracy :', accuracy(y_test2, pred))
1004.     print('precision :', precision(y_test2, pred))
1005.     print('recall :', recall(y_test2, pred))
1006.     print('f1-score :', f1(y_test2, pred))
1007.     pred = xgboost_predict(model, X_test2_1, eta)
1008.     print('accuracy :', accuracy(y_test2_1, pred))
1009.     print('precision :', precision(y_test2_1, pred))
1010.     print('recall :', recall(y_test2_1, pred))
1011.     print('f1-score :', f1(y_test2_1, pred))
1012.     pred = xgboost_predict(model, X_test2_2, eta)
1013.     print('accuracy :', accuracy(y_test2_2, pred))
1014.     print('precision :', precision(y_test2_2, pred))
1015.     print('recall :', recall(y_test2_2, pred))
1016.     print('f1-score :', f1(y_test2_2, pred))
1017.     pred = xgboost_predict(model, X_test2_3, eta)
1018.     print('accuracy :', accuracy(y_test2_3, pred))
1019.     print('precision :', precision(y_test2_3, pred))
1020.     print('recall :', recall(y_test2_3, pred))
1021.     print('f1-score :', f1(y_test2_3, pred))
1022.     model, eta = xgboost_train(X_train2, y_train2,
1023.     eta = 0.4,
1024.     max_round = 45,
1025.     max_depth = 3,
1026.     row_sub_frac = 0.95,
1027.     col_sub_frac = 1,
1028.     min_child_weight = 1,
```

```
1029.     min_sample_split = 10,
1030.     lamda = 1,
1031.     gamma = 0,
1032.     metric = 'accuracy')
1033.     pred = xgboost_predict(model, X_test2, eta)
1034.     print('\n')
1035.     print('accuracy :', accuracy(y_test2, pred))
1036.     print('precision :', precision(y_test2, pred))
1037.     print('recall :', recall(y_test2, pred))
1038.     print('f1-score :', f1(y_test2, pred))
1039.     pred = xgboost_predict(model, X_test2_1, eta)
1040.     print('accuracy :', accuracy(y_test2_1, pred))
1041.     print('precision :', precision(y_test2_1, pred))
1042.     print('recall :', recall(y_test2_1, pred))
1043.     print('f1-score :', f1(y_test2_1, pred))
1044.     pred = xgboost_predict(model, X_test2_2, eta)
1045.     print('accuracy :', accuracy(y_test2_2, pred))
1046.     print('precision :', precision(y_test2_2, pred))
1047.     print('recall :', recall(y_test2_2, pred))
1048.     print('f1-score :', f1(y_test2_2, pred))
1049.     pred = xgboost_predict(model, X_test2_3, eta)
1050.     print('accuracy :', accuracy(y_test2_3, pred))
1051.     print('precision :', precision(y_test2_3, pred))
1052.     print('recall :', recall(y_test2_3, pred))
1053.     print('f1-score :', f1(y_test2_3, pred))
1054.     model, eta = xgboost_train(X_train2, y_train2,
1055.     eta = 0.5,
1056.     max_round = 45,
1057.     max_depth = 3,
1058.     row_sub_frac = 0.95,
1059.     col_sub_frac = 1,
1060.     min_child_weight = 1,
1061.     min_sample_split = 10,
1062.     lamda = 1,
1063.     gamma = 0,
1064.     metric = 'accuracy')
1065.     pred = xgboost_predict(model, X_test2, eta)
```

```
print('\n')
print('accuracy :', accuracy(y_test2, pred))
print('precision :', precision(y_test2, pred))
print('recall :', recall(y_test2, pred))
print('f1-score :', f1(y_test2, pred))
pred = xgboost_predict(model, X_test2_1, eta)
print('accuracy :', accuracy(y_test2_1, pred))
print('precision :', precision(y_test2_1, pred))
print('recall :', recall(y_test2_1, pred))
print('f1-score :', f1(y_test2_1, pred))
pred = xgboost_predict(model, X_test2_2, eta)
print('accuracy :', accuracy(y_test2_2, pred))
print('precision :', precision(y_test2_2, pred))
print('recall :', recall(y_test2_2, pred))
print('f1-score :', f1(y_test2_2, pred))
pred = xgboost_predict(model, X_test2_3, eta)
print('accuracy :', accuracy(y_test2_3, pred))
print('precision :', precision(y_test2_3, pred))
print('recall :', recall(y_test2_3, pred))
print('f1-score :', f1(y_test2_3, pred))
```

PAPER NAME
18.K1.0024_Yesica Sugiarto, Ting

AUTHOR
Yesica Sugiarto, Ting

WORD COUNT
11691 Words

CHARACTER COUNT
64694 Characters

PAGE COUNT
55 Pages

FILE SIZE
210.6KB

SUBMISSION DATE
May 11, 2022 11:58 AM GMT+7

REPORT DATE
May 11, 2022 11:59 AM GMT+7

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 6% Internet database
- Crossref database
- 8% Submitted Works database
- 2% Publications database
- Crossref Posted Content database

Excluded from Similarity Report

- Bibliographic material
- Cited material
- Quoted material
- Small Matches (Less then 10 words)