APPENDIX

RANDOM FOREST

IMPORT LIBRARY

```
1. import pandas as pd
2. import numpy as np
3. import random
4. from random import choice
5. import math
6. pd.set_option('display.max_columns', None)
```

DATASET READ

```
7. def PrintTable(data, limit):
8. return pd.DataFrame(data).head(limit)
9. review = pd.read_csv("REVIEWS.csv", header=0, nrows=5000)
10.    review = review.values
11.    print('Dataset :', PrintTable(review, 5000))
```

PREPROCESSING DATA

```
12.    def PreProcessingData(data):
13.        ArrTemp = []
14.        for a in data:
15.            if a[2] >= 5:
16.                label = int(1)
17.            else:
18.                label = int(0)
19.            ArrTemp.append( [ a[0], a[1], a[2], a[4], label ] )
20.        return ArrTemp
21.    review = PreProcessingData(review)
22.    print('Hasil Preprocessing Data :', PrintTable(review, 2000))
```

TF-IDF

```
23.    def TfIdf(data):
24.        WordArr = []
25.        for a in data:
26.            TempWord = str.lower(a[1]).split(' ')
27.            print('Temp Word: ', TempWord)
28.
29.            for b in TempWord:
30.                if WordArr != []:
31.                    count = 0
32.                    for c in range(len(WordArr)):
33.                        if WordArr[c][0] == b:
34.                            WordArr[c] = [WordArr[c][0], (int(WordArr
   [c][1]) + int(1))]
```

```
                            count = 1
                            break
                    if count == 0:

                        if b != '':
                            WordArr.append([b, int(1)])
                else:
                    if b != '':
                        WordArr.append([b, int(1)])

        print('\n\nCount Word : \n', WordArr)
        TfArr = []
        for a in range(len(data)):

            TempTf = []
            CountTf = 0
            TempWord = str.lower(data[a][1]).split(' ')

            print('\n', TempWord)

            for b in TempWord:
                print(b)
                if b != '':

                    if TempTf != []:

                        count = 0
                        for c in range(len(TempTf)):
                            if TempTf[c][0] == b:
                                TempTf[c] = [TempTf[c][0], TempTf[c][
  1], (int(TempTf[c][2]) + int(1))]
                                count = 1
                                break

                        if count == 0:
                            if b != '':
                                TempTf.append([data[a][0], b, int(1)]
  )

                    else:
                        if b != '':
                            TempTf.append([data[a][0], b, int(1)])

            TfArr.append(TempTf)
            print('\n temp tf: ', TempTf)
```

```
78.         result = []
79.         for a in range(len(TfArr)):
80.             CountTemp = 0
81.             for b in TfArr[a]:
82.                 for c in WordArr:
83.                     if b[1] == c[0]:
84.                         Tf = b[2]/ len(TfArr[a])
85.                         Idf = math.log(int(2000) / int(c[1]))
86.                         TfIdf = Tf * Idf
87.                         CountTemp = (float(CountTemp) + TfIdf) / 2
88.                         print('TF :', Tf, ' IDF : ', Idf, ' TF-
    IDF : ', TfIdf)
89.
90.             data[a].append(CountTemp)
91.         return data
92.     review = TfIdf(review)
93.     print('Hasil TF IDF:', PrintTable(review, 2000))
```

SPLIT DATA

```
94.     def SplitData(train, test, data):
95.         TrainingData = []
96.         TrainingLabel = []
97.         TestingData = []
98.         TestingLabel = []
99.         CountTraining = int(len(data) * (train/100))
100.        CountTesting = int(len(data) * (test/100))
101.
102.        for a in range(CountTraining):
103.            temp = choice(data)
104.            exists = temp in TrainingData
105.            while exists == True:
106.                temp = choice(data)
107.                exists = temp in TrainingData
108.            TrainingLabel.append(temp[4])
109.            temp = [temp[2], temp[5]]
110.            TrainingData.append(temp)
111.
112.        for a in range(CountTesting):
113.            temp = choice(data)
114.            exists = temp in TestingData
115.            while exists == True:
116.                temp = choice(data)
117.                exists = temp in TestingData
118.            TestingLabel.append(temp[4])
119.            temp = [temp[2],temp[5]]
```

```
            TestingData.append(temp)

        return TrainingData, TrainingLabel, TestingData, TestingLabel
    TrainingData, TrainingLabel, TestingData, TestingLabel =  SplitData(70, 30, review)
    print('Training Data:', PrintTable(TrainingData, 2000))
    print('Training Label:', TrainingLabel)
    print('Testing Data:', PrintTable(TestingData, 2000))
    print('Testing Label:', TestingLabel)
```

IMPLEMENTING RANDOM FOREST ALGORITHM

```
    def RandomForest(TrainingData, TrainingLabel, nIteration, maxFeature, max_depth, min_samples_split):
        tree_ls = list()
        oob_ls = list()

        for i in range(nIteration):
            bootstrapData = []
            bootstrapLabel = []
            oobData = []
            oobLabel = []
            bootstrapIndices = []
            oobIndices = []

            print('Training Data : ', TrainingData)
            print('Training Label : ', TrainingLabel)

            for count in range(len(TrainingData)):
                bootstrapIndices.append(random.randint(0, len(TrainingData)-1))
            print('Bootstrap indices : ', bootstrapIndices)

            for count in range(len(TrainingData)):
                if count not in bootstrapIndices:
                    oobIndices.append(count)
            print('Oob Indices : ', oobIndices)

            for a in range(len(bootstrapIndices)):
                bootstrapData.append(TrainingData[bootstrapIndices[a]])
                bootstrapLabel.append(TrainingLabel[bootstrapIndices[a]])

            for a in range(len(oobIndices)):
                oobData.append(TrainingData[oobIndices[a]])
```

```
158.                 oobLabel.append(TrainingLabel[oobIndices[a]])
159.
160.             print('Bootstrap Data : ', bootstrapData)
161.             print('Bootstrap Label : ', bootstrapLabel)
162.             print('Oob Data : ', oobData)
163.             print('Oob Label : ', oobLabel)
164.
165.             rootNode = countSplitPoint(bootstrapData, bootstrapLabel,
    maxFeature)
166.             print('Root Node : ', rootNode)
167.             splitNode(rootNode, maxFeature, min_samples_split, max_de
   pth, 1)
168.
169.             tree_ls.append(rootNode)
170.             oob_error = OobScore(rootNode, oobData, oobLabel)
171.             oob_ls.append(oob_error)
172.
173.         return tree_ls
174.
175.     def countSplitPoint(bootstrapData, bootstrapLabel, maxFeature):
176.         featureLs = list()
177.
178.         numFeatures = len(bootstrapData[0])
179.         print('Number of Feature : ', numFeatures)
180.
181.         while len(featureLs) < maxFeature:
182.             feature_idx = random.sample(range(numFeatures), 1)
183.             print('Feature IDX : ', feature_idx)
184.             featureLs.extend(feature_idx)
185.
186.         print('\nFeature ls : ', featureLs)
187.
188.         best_info_gain = -999
189.         node = None
190.
191.         for featureIdx in featureLs:
192.             print('Feature IDX :', featureIdx)
193.
194.             for splitPoint in bootstrapData[:featureIdx+1]:
195.                 print('Split Point : ', splitPoint)
196.                 leftChild = {'bootstrapData': [], 'bootstrapLabel': [
   ]}
197.                 rightChild = {'bootstrapData': [], 'bootstrapLabel':
   []}
198.
```

```
199.                for i, value in enumerate(bootstrapData[featureIdx+1:
   ]):
200.                    print('Continuous Value : ', value)
201.
202.                    if value <= splitPoint:
203.                        leftChild['bootstrapData'].append(bootstrapDa
   ta[i])
204.                        leftChild['bootstrapLabel'].append(bootstrapL
   abel[i])
205.                    else:
206.
207.                        rightChild['bootstrapData'].append(bootstrapD
   ata[i])
208.                        rightChild['bootstrapLabel'].append(bootstrap
   Label[i])
209.
210.                print('Left child : ', leftChild)
211.                print('Right child : ', rightChild)
212.
213.                splitInfoGain = countInformationGain(leftChild['boots
   trapLabel'], rightChild['bootstrapLabel'])
214.
215.                if splitInfoGain > best_info_gain:
216.                    best_info_gain = splitInfoGain
217.                    leftChild['bootstrapData'] = leftChild['bootstrap
   Data']
218.                    rightChild['bootstrapData'] = rightChild['bootstr
   apData']
219.                    node = {'informationGain': splitInfoGain,
220.                            'leftChild': leftChild,
221.                            'rightChild': rightChild,
222.                            'splitPoint': splitPoint,
223.                            'featureIdx': featureIdx}
224.        print('Node : ', node)
225.
226.        return node
227.
228.    def countInformationGain(leftChild, rightChild):
229.        parent = leftChild + rightChild
230.        pParent = parent.count(1) / len(parent) if len(parent) > 0 el
   se 0
231.        pLeft = leftChild.count(1) / len(leftChild) if len(leftChild)
    > 0 else 0
232.        pRight = rightChild.count(1) / len(rightChild) if len(rightCh
   ild) > 0 else 0
```

```
233.       print('\nP Parent : ', pParent)
234.       print('\nP Left : ', pLeft)
235.       print('\nP Right : ', pRight)
236.
237.       igParent = entropy(pParent)
238.       igLeft = entropy(pLeft)
239.       igRight = entropy(pRight)
240.
241.       informationGain = igParent -
    len(leftChild) / len(parent) * igLeft -
    len(rightChild) / len(parent) * igRight
242.
243.       return informationGain
244.
245.   def entropy(p):
246.       if p == 0:
247.           return 0
248.       elif p == 1:
249.           return 0
250.       else:
251.           return - (p * np.log2(p) + (1 - p) * np.log2(1-p))
252.
253.   def splitNode(node, maxFeature, minSampleSplit, maxDepth, depth):
254.       print('Depth Forest : ', depth)
255.
256.       left_child = node['leftChild']
257.       right_child = node['rightChild']
258.
259.       del(node['leftChild'])
260.       del(node['rightChild'])
261.
262.       if len(left_child['bootstrapLabel']) == 0 or len(right_child[
   'bootstrapLabel']) == 0:
263.           empty_child = {'bootstrapLabel': left_child['bootstrapLab
   el'] + right_child['bootstrapLabel']}
264.           node['left_split'] = TerminalNode(empty_child)
265.           node['right_split'] = TerminalNode(empty_child)
266.           print('Node Left Split Terminal Node masuk ke kondisi Ata
   s :', TerminalNode(empty_child), ' - ', empty_child)
267.           return
268.
269.       if depth >= maxDepth:
270.           node['left_split'] = TerminalNode(left_child)
271.           node['right_split'] = TerminalNode(right_child)
```

```
            print('Node Left Split Terminal Node Bawah : ', node['lef
    t_split'])
            return node

        if len(left_child['bootstrapData']) <= minSampleSplit:
            node['left_split'] = node['right_split'] = TerminalNode(l
    eft_child)
        else:

            node['left_split'] = countSplitPoint(left_child['bootstra
    pData'], left_child['bootstrapLabel'], maxFeature)
            print('Node Left Split : ', node['left_split'])
            splitNode(node['left_split'], maxDepth, minSampleSplit, m
    axDepth, depth + 1)

        if len(right_child['bootstrapData']) <= minSampleSplit:
            node['right_split'] = node['left_split'] = TerminalNode(r
    ight_child)
        else:
            node['right_split'] = countSplitPoint(right_child['bootst
    rapData'], right_child['bootstrapLabel'], maxFeature)
            print('Node Right Split : ', node['right_split'])
            splitNode(node['right_split'], maxFeature, minSampleSplit
    , maxDepth, depth + 1)

    def TerminalNode(node):
        bootstrapLabel = node['bootstrapLabel']
        pred = max(bootstrapLabel, key = bootstrapLabel.count)
        return pred
    def OobScore(tree, TestingData, TestingLabel):
        print('Tree in OOB Score: ', tree)
        mis_label = 0
        for i in range(len(TestingData)):
            pred = PredictTree(tree, TestingData[i])
            if pred != TestingLabel[i]:
                mis_label += 1
        return mis_label / len(TestingData) if len(TestingData) > 0 e
    lse 0
    def PredictTree(tree, TestingData):
        feature_idx = tree['featureIdx']
        if TestingData[feature_idx] <= tree['splitPoint'][feature_idx
    ]:
                value = tree['left_split']
                return value
        else:
```

```
            return tree['right_split']
def PredictRF(tree_ls, X_test):
    pred_ls = list()

    for i in range(len(X_test)):

        ensemble_preds = [PredictTree(tree, X_test[i]) for tree i
n tree_ls]
        print('Ensemble Preds : ', ensemble_preds)

        final_pred = max(ensemble_preds, key = ensemble_preds.cou
nt)

        print('Majority Voting : ', final_pred)
        pred_ls.append(final_pred)

    return np.array(pred_ls)
nIteration = 50
maxFeature = 2
maxDepth = 10
minSampleSplit = 2

model = RandomForest(TrainingData, TrainingLabel, nIteration, max
Feature, maxDepth, minSampleSplit)
print(model)
```

CALCULATE ACCURACY

```
preds = PredictRF(model, TestingData)
print('Predict Tree: ', preds)
acc = sum(preds == TestingLabel) / len(TestingLabel)

print("Testing accuracy: {}".format(np.round(acc,3)))
```

VISUALIZATION WITH PIE CHART

```
import matplotlib.pyplot as plt
pos = int(0)
for value in preds:

    if value == 1:
        pos += 1
neg = len(preds) - pos

print('Positif : ', pos, ' Negatif : ', neg, ' Total Data : ', le
n(preds))
SenLabels = ['Positif', 'Negatif']
```

```
345.   preds = np.array([pos, neg])
346.   plt.pie(preds, labels= SenLabels, colors= ['red', 'pink'])
347.   plt.show()
```

LOGISTIC REGRESSION

IMPORT LIBRARY

```
1. import pandas as pd
2. import numpy as np
3. import random
4. from random import choice
5. import math
6. pd.set_option('display.max_columns', None)
```

DATASET READ

TF-IDF

```
7. def PrintTable(data, limit):
8.     return pd.DataFrame(data).head(limit)
9. def SplitSentence(data, seperator):
10.        return data.split(seperator)
11.    review = pd.read_csv("REVIEWS.csv", header=0, nrows=2000)
12.    review = review.values
13.    print('Dataset : ', PrintTable(review, 2000))
```

TF-IDF

```
247.   def TfIdf(data):
248.       WordArr = []
249.       for a in data:
250.           TempWord = str.lower(a[1]).split(' ')
251.           print('Temp Word - ', TempWord)
252.
253.           for b in TempWord:
254.               if WordArr != []:
255.
256.                   count = 0
257.                   for c in range(len(WordArr)):
258.
259.                       if WordArr[c][0] == b:
260.
261.                           WordArr[c]        =        [WordArr[c][0],
    (int(WordArr[c][1]) + int(1))]
262.                           count = 1
263.                           break
264.
```

```
                if count == 0:

                    if b != '':
                        WordArr.append([b, int(1)])
            else:
                if b != '':
                    WordArr.append([b, int(1)])

    print('Count Word : ', WordArr)

TfArr = []
    for a in range(len(data)):
        TempTf = []
        CountTf = 0
        TempWord = str.lower(data[a][1]).split(' ')

        print('TempWord: ', TempWord)
        for b in TempWord:
            print(b)
            if b != '':
                if TempTf != []:

                    count = 0
                    for c in range(len(TempTf)):

                        if TempTf[c][0] == b:

                            TempTf[c]       =       [TempTf[c][0],
 TempTf[c][1], (int(TempTf[c][2]) + int(1))]
                            count = 1
                            break

                    if count == 0:
                        if b != '':
                            TempTf.append([data[a][0],          b,
 int(1)])

                else:
                    if b != '':
                        TempTf.append([data[a][0], b, int(1)])

         TfArr.append(TempTf)
        print('temp tf: ', TempTf)

    result = []
```

```
308.      for a in range(len(TfArr)):
309.          CountTemp = 0
310.          for b in TfArr[a]:
311.
312.              for c in WordArr:
313.
314.                  if b[1] == c[0]:
315.                      Tf = b[2]/ len(TfArr[a])
316.                      Idf = math.log(int(20000) / int(c[1]))
317.                      TfIdf = Tf * Idf
318.                      CountTemp = (float(CountTemp) + TfIdf) / 2
319.                      print('TF :', Tf, ' IDF : ', Idf, ' TF-IDF :
   ', TfIdf)
320.
321.          result.append([data[a][0],    data[a][1],     data[a][2],
   data[a][3], data[a][4], CountTemp])
322.
323.      return result
324.
325.  review = TfIdf(review)
326.  print('Hasil TF IF : ', PrintTable(review, 2000))
```

TEXT PROCESSING

```
327.  def TextProcessing(train, test, data):
328.      print('Text Processing')
329.      posData = []
330.      negData = []
331.      for value in data:
332.          if value[2] >= 5:
333.              label = 1
334.              posData.append([value[0],     value[1],     value[2],
   value[4], label, value[5]])
335.          else:
336.              label = 0
337.              negData.append([value[0],     value[1],     value[2],
   value[4], label, value[5]])
338.
339.      countTrainPos = int(train * (len(posData)/100))
340.      countTrainNeg = int(train * (len(negData)/100))
341.      countTestPos = int(test * (len(posData)/100))
342.      countTestNeg = int(test * (len(negData)/100))
343.      sizeArrayTest = countTestPos+countTestNeg
344.
345.      TrainingData = []
346.      TrainingLabel = []
```

```
347.      TestingData = []
348.      TestingLabel = [].
349.
350.      for iteration in range(countTrainPos):
351.          temp = choice(posData)
352.          checkTemp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
353.          exists = checkTemp in TrainingData
354.          while exists == True:
355.              temp = choice(posData)
356.              checkTemp  =  [temp[0],  temp[1],  temp[2],  temp[3],
   temp[5]]
357.              exists = checkTemp in TrainingData
358.          TrainingLabel.append(temp[4])
359.          temp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
360.          TrainingData.append(temp)
361.
362.      for iteration in range(countTrainNeg):
363.          temp = choice(negData)
364.          checkTemp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
365.          exists = checkTemp in TrainingData
366.          while exists == True:
367.              temp = choice(negData)
368.              checkTemp  =  [temp[0],  temp[1],  temp[2],  temp[3],
   temp[5]]
369.              exists = checkTemp in TrainingData
370.          TrainingLabel.append(temp[4])
371.          temp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
372.          TrainingData.append(temp)
373.
374.      for iteration in range(countTestPos):
375.          temp = choice(posData)
376.          checkTemp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
377.          exists = checkTemp in TestingData
378.          while exists == True:
379.              temp = choice(posData)
380.              checkTemp  =  [temp[0],  temp[1],  temp[2],  temp[3],
   temp[5]]
381.              exists = checkTemp in TestingData
382.          TestingLabel.append(temp[4])
383.          temp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
384.          TestingData.append(temp)
385.
386.      for iteration in range(countTestNeg):
387.          temp = choice(negData)
388.          checkTemp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
```

```
389.          exists = checkTemp in TestingData
390.          while exists == True:
391.              temp = choice(negData)
392.              checkTemp  =  [temp[0],  temp[1],  temp[2],  temp[3],
   temp[5]]
393.              exists = checkTemp in TestingData
394.          TestingLabel.append(temp[4])
395.          temp = [temp[0], temp[1], temp[2], temp[3], temp[5]]
396.          TestingData.append(temp)
397.
398.      return     TrainingData,     TrainingLabel,     TestingData,
   TestingLabel, posData, negData
399.
400.  TrainingData, TrainingLabel, TestingData, TestingLabel, posData,
   negData = TextProcessing(70, 30, review)
```

FEATURE EXTRACTION

```
401.  def WordDict(posData, negData):
402.      print('Feature Extraction')
403.      posWord={}
404.      for values in posData:
405.          values = SplitSentence(str.lower(values[1]), ' ')
406.          for value in values:
407.
408.              if value != '':
409.                  if (value,1) not in posWord:
410.                      posWord[(value,1)]=1
411.                  else:
412.                      posWord[(value,1)]=posWord[(value,1)]+1
413.
414.      negWord={}
415.      for values in negData:
416.          values = SplitSentence(str.lower(values[1]), ' ')
417.
418.          for value in values:
419.              if value != '':
420.                  if (value,1) not in negWord:
421.                      negWord[(value,0)]=1
422.                  else:
423.                      negWord[(value,0)]=negWord[(value,1)]+1
424.
425.      wordDict = dict(posWord)
426.      wordDict.update(negWord)
427.      print(wordDict)
428.      return wordDict
```

```
429.
430.  def FeatureExtraction(data, wordDict):
431.      word_l = SplitSentence(str.lower(data[1]), ' ')
432.      x = np.zeros((1, 4))
433.      x[0,0] = data[2]
434.      x[0,1] = data[4]
435.      for word in word_l:
436.          try:
437.              x[0,2] += wordDict[(word,1)]
438.          except:
439.              x[0,2] += 0
440.          try:
441.              x[0,3] += wordDict[(word,0.0)]
442.          except:
443.              x[0,3] += 0
444.
445.      assert(x.shape == (1, 4))
446.      print('X in Feature Extraction : ', x)
447.      return x
448.
449.  WordDicts = WordDict(posData, negData)
450.  X = np.zeros((len(TrainingData), 4))
451.
452.  for i in range(len(TrainingData)):
453.      print('Training Data : ', TrainingData[i])
454.      print('Current X :\n', X)
455.      X[i, :]= FeatureExtraction(TrainingData[i], WordDicts)
456.  print('Feature Extraction : ', X)
```

INITIALIZATION

```
457.  intercept = np.ones((X.shape[0], 1))
458.  xData = np.concatenate((intercept, X), axis=1)
459.  weight = np.zeros(xData.shape[1])
460.  yData = np.asarray(TrainingLabel)
461.
462.  print('Intercept : ', intercept.T)
463.  print('X data : ', xData)
464.  print('weight : ', weight)
465.  print('y : ', yData)
```

IMPLEMENTATION OF THE LOGISTIC REGRESSION ALGORITHM

```
466.  def countSigmoid(x, weight):
467.      z = np.dot(x, weight)
468.      print('\n Z : ', z)
469.      return 1 / (1 + np.exp(-z))
```

```
470.
471.  def countLoss(h, y):
472.      result = (-y * np.log(h) - (1 - y) * np.log(1 - h))
473.      print('\nLoss : ', result)
474.      mean = 0
475.
476.      for value in result:
477.          if np.isnan(value):
478.              mean = (mean + 0) / 2
479.          else:
480.              mean = (mean + value) / 2
481.      return mean
482.
483.  def countGradientDescent(X, h, y):
484.      hY = np.nan_to_num(h-y, copy=True, nan=0.0)
485.      X = np.nan_to_num(X, copy=True, nan=0.0)
486.      result = np.dot(X.T, hY)/ y.shape[0]
487.      return result
488.
489.  def fit(lr , iterations, intercept, xData, weight, yData):
490.      for i in range(iterations):
491.          print('Iterasi ke-', i)
492.          sigma = countSigmoid(xData, weight)
493.          sigma = np.nan_to_num(sigma, copy=True, nan=0.0)
494.          print('Sigma : ', sigma)
495.
496.          loss = countLoss(sigma,yData)
497.          print('Loss : ', loss)
498.
499.          dW = countGradientDescent(xData , sigma, yData)
500.          print('\nGradient Descent : ', dW)
501.
502.          weight -= lr * dW
503.          print('Weight : ', weight)
504.      print('fitted successfully to data')
505.      return weight
506.
507.  weight = fit(0.1 , 100, intercept, xData, weight, yData)
```

PREDICT LABEL AND CALCULATE ACCURACY

```
508.  def predict(xData , treshold, intercept, weight):
509.      result = countSigmoid(xData, weight)
510.      print('Result Before : ', result)
511.      result = result >= treshold
512.      print('Result After : ', result)
```

```
513.      predLabel = np.zeros(result.shape[0])
514.      for i in range(len(predLabel)):
515.          if result[i] == True:
516.              predLabel[i] = 1
517.          else:
518.              continue
519.      return predLabel
520.
521.  predLabel = predict(xData, 0.5, intercept, weight)
522.
523.  print('Prediction Label : ', predLabel)
524.  print('Y Label : ', yData)
525.  print('Accuracy   ->   {}\n'.format(sum(predLabel   ==   yData)   /
   yData.shape[0]))
```

VISUALIZATION WITH PIE CHART

```
526.  import matplotlib.pyplot as plt
527.  pos = int(0)
528.
529.  for value in predLabel:
530.      if value == 1:
531.          pos += 1
532.  neg = len(predLabel) - pos
533.
534.  print('Positif : ', pos, ' Negatif : ', neg, ' Total Data : ',
   len(predLabel))
535.  SenLabels = ['Positif', 'Negatif']
536.  predLabel = np.array([pos, neg])
537.
538.  plt.pie(predLabel, labels= SenLabels, colors= ['red', 'pink'])
539.  plt.show()
```

PAPER NAME
18.K1.0023_Lady Viona Sugianto

AUTHOR
Lady Viona Sugianto

WORD COUNT
17341 Words

CHARACTER COUNT
80939 Characters

PAGE COUNT
43 Pages

FILE SIZE
137.6KB

SUBMISSION DATE
May 12, 2022 3:08 PM GMT+7

REPORT DATE
May 12, 2022 3:10 PM GMT+7

2% Overall Similarity

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

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

Excluded from Similarity Report

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