```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler


drive.mount("/content/drive")
path = "/content/drive/MyDrive/Capstone/exercise_datasetV2.csv"
df = pd.read_csv(path)
print(df.head())
banyak_kategori = len(df.index)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
  Activity, Exercise or Sport (1 hour) Intensity Description  \
0          Cycling, mountain bike, bmx                   NaN
1  Cycling, <10 mph, leisure bicycling                   NaN
2             Cycling, >20 mph, racing                   NaN
3          Cycling, 10-11.9 mph, light                   NaN
4       Cycling, 12-13.9 mph, moderate                   NaN

   Duration (minutes)  Calories per kg
0                  60         1.750730
1                  60         0.823236
2                  60         3.294974
3                  60         1.234853
4                  60         1.647825
```

```
list_berat = []
for i in range(len(df.index)):
  list_berat.append(1)

df['berat'] =  list_berat
dict_df = {'Activity, Exercise or Sport (1 hour)' : [], 'Duration (minutes)': [], 'Calories per kg': [], 'berat' : []}
df_new = df
for index, row in df.iterrows():
  print(index)
  menit = row['Duration (minutes)']
  activity = row['Activity, Exercise or Sport (1 hour)']
  calories = row['Calories per kg']
  for i in range(1,menit):
    for j in range(2,101):
      new_calories = calories*1.0/60*i*j
      list_activity = dict_df.get('Activity, Exercise or Sport (1 hour)')
      list_duration = dict_df.get('Duration (minutes)')
      list_calories = dict_df.get('Calories per kg')
      list_berat = dict_df.get('berat')
      list_activity.append(activity)
      list_duration.append(i)
      list_calories.append(new_calories)
      list_berat.append(j)
      #new_row = pd.DataFrame({'Activity, Exercise or Sport (1 hour)' : [activity], 'Duration (minutes)': [i], 'Calories per k

df_curr = pd.DataFrame(dict_df)
df_new = pd.concat([df_curr, df_new.loc[:]]).reset_index(drop=True)
#df2 = pd.concat([new_row,df.loc[:]]).reset_index(drop=True)
print(df_new.head())
print(df_new.tail())
```

```
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
```

```
  Activity, Exercise or Sport (1 hour)  Duration (minutes)  Calories per kg  \
0          Cycling, mountain bike, bmx                  1         0.058358
1          Cycling, mountain bike, bmx                  1         0.087536
2          Cycling, mountain bike, bmx                  1         0.116715
3          Cycling, mountain bike, bmx                  1         0.145894
4          Cycling, mountain bike, bmx                  1         0.175073

   berat Intensity Description
0      2                   NaN
1      3                   NaN
2      4                   NaN
3      5                   NaN
4      6                   NaN
             Activity, Exercise or Sport (1 hour)  Duration (minutes)  \
1448811                         General cleaning                  60
1448812                        Cleaning, dusting                  60
1448813                         Taking out trash                  60
1448814            Walking, pushing a wheelchair                  60
1448815  Teach physical education,exercise class                  60

         Calories per kg  berat Intensity Description
1448811         0.721008      1                   NaN
1448812         0.515199      1                   NaN
1448813         0.617427      1                   NaN
1448814         0.823236      1                   NaN
1448815         0.823236      1                   NaN
```

```
print(len(df_new.index))
print(df_new.describe())
print(df_new.dtypes)
df_new.rename(columns = {'Activity, Exercise or Sport (1 hour)':'activity', 'Duration (minutes)' : 'durasi' , 'Calories per kg
print(df_new.head())
```

```
1448816
       Duration (minutes)  Calories per kg         berat
count        1.448816e+06     1.448816e+06  1.448816e+06
mean         3.000514e+01     3.467251e+01  5.099144e+01
std          1.703246e+01     3.748635e+01  2.858243e+01
min          1.000000e+00     1.033558e-02  1.000000e+00
25%          1.500000e+01     8.237434e+00  2.600000e+01
50%          3.000000e+01     2.219663e+01  5.100000e+01
75%          4.500000e+01     4.774767e+01  7.600000e+01
max          6.000000e+01     3.644815e+02  1.000000e+02
Activity, Exercise or Sport (1 hour)     object
Duration (minutes)                        int64
Calories per kg                         float64
berat                                     int64
Intensity Description                    object
dtype: object
                      activity  durasi  calories  berat Intensity Description
0  Cycling, mountain bike, bmx       1  0.058358      2                   NaN
1  Cycling, mountain bike, bmx       1  0.087536      3                   NaN
2  Cycling, mountain bike, bmx       1  0.116715      4                   NaN
3  Cycling, mountain bike, bmx       1  0.145894      5                   NaN
4  Cycling, mountain bike, bmx       1  0.175073      6                   NaN
```

```
target = df['Activity, Exercise or Sport (1 hour)']
print(df_new.head())
numeric_feature_names = ['durasi', 'calories', 'berat']
numeric_features = df_new[numeric_feature_names]
numeric_features.head()
```

```
                      activity  durasi  calories  berat Intensity Description
0  Cycling, mountain bike, bmx       1  0.058358      2                    NaN
1  Cycling, mountain bike, bmx       1  0.087536      3                    NaN
2  Cycling, mountain bike, bmx       1  0.116715      4                    NaN
3  Cycling, mountain bike, bmx       1  0.145894      5                    NaN
4  Cycling, mountain bike, bmx       1  0.175073      6                    NaN
```

	durasi	calories	berat
0	1	0.058358	2
1	1	0.087536	3
2	1	0.116715	4

```
import tensorflow as tf
tf.convert_to_tensor(numeric_features)
normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(numeric_features)


"""
def get_base_model():
  model = tf.keras.Sequential([
    normalizer,
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(banyak_kategori, activation = 'softmax')
  ])

  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-3),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model
"""
```

```
'\ndef get_base_model():\n  model = tf.keras.Sequential([\n    normalizer,\n    tf.keras.layers.Dense(10, activation='rel
se(10, activation='relu'),\n    tf.keras.layers.Dense(banyak_kategori, activation = 'softmax')\n  ])\n\n  model.compile(o
Adam(learning_rate=2e-3),\n                loss='categorical_crossentropy',\n                metrics=['accuracy'])\n  ret
```

```
"""
y = df_new['activity']
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
"""
```

```
'\ny = df_new['activity']\nencoder = LabelEncoder()\nencoder.fit(y)\nencoded_Y = encoder.transform(y)\n# convert integers
hot encoded)\ndummy_y = np_utils.to_categorical(encoded_Y)\n'
```

```
#est = KerasClassifier(build_fn= get_base_model, epochs=200, batch_size=5, verbose=0)



#kfold = KFold(n_splits=5, shuffle=True)



"""
x = df_new[numeric_feature_names]

results = cross_val_score(est, x, dummy_y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
"""
```

```
'\nx = df_new[numeric_feature_names]\n\nresults = cross_val_score(est, x, dummy_y, cv=kfold)\nprint("Baseline: %.2f%% (%.
results.std()*100))\n'
```

```
"""
https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
https://www.tensorflow.org/tutorials/load_data/pandas_dataframe
https://regenerativetoday.com/a-step-by-step-tutorial-to-develop-a-multi-output-model-in-tensorflow/
"""
```

```
'\nhttps://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/\nhttps://www.tens
a/pandas_dataframe\nhttps://regenerativetoday.com/a-step-by-step-tutorial-to-develop-a-multi-output-model-in-tensorflow/\
```

```
jumlah_class = len(df_new['activity'].value_counts())
print(jumlah_class)
```

```
248
```

```
df_new['activity'] = df_new['activity'].astype('category')
df_new['activity_category'] = df_new['activity'].cat.codes.astype('category')
print(df_new.head())
```

```
                        activity  durasi  calories  berat Intensity Description  \
0  Cycling, mountain bike, bmx       1  0.058358      2                   NaN
1  Cycling, mountain bike, bmx       1  0.087536      3                   NaN
2  Cycling, mountain bike, bmx       1  0.116715      4                   NaN
3  Cycling, mountain bike, bmx       1  0.145894      5                   NaN
4  Cycling, mountain bike, bmx       1  0.175073      6                   NaN

  activity_category
0                61
1                61
2                61
3                61
4                61
```

```
df_new_2 = df_new.drop(columns = ['activity', 'Intensity Description'])
sc = StandardScaler()
x = pd.DataFrame(sc.fit_transform(df_new_2))
```

```
df_new_2['durasi'] = MinMaxScaler().fit_transform(np.array(df_new_2['durasi']).reshape(-1,1))
df_new_2['calories'] = MinMaxScaler().fit_transform(np.array(df_new_2['calories']).reshape(-1,1))
df_new_2['berat'] = MinMaxScaler().fit_transform(np.array(df_new_2['berat']).reshape(-1,1))
```

```
y = tf.keras.utils.to_categorical(df_new["activity_category"].values, num_classes=jumlah_class)

x_train, x_test, y_train, y_test = train_test_split(x.values, y, test_size=0.2)
```

```
print(x_train)
print(y_train)
print(x_test)
print(y_test)
```

```
[[ 0.46939015 -0.17325493 -0.83937753 -0.76127046]
 [ 0.88037034  1.11911085  0.38515139 -1.19428668]
 [-0.11772441 -0.80949038 -1.1542564   1.36190586]
 ...
 [ 0.41067869  0.83675733  0.03528598 -0.27238117]
 [-0.46999314 -0.01845191  1.36477452  0.76127046]
 [ 1.40877344  2.22409518  1.39976106 -1.71111249]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]]
[[ 1.17392762  1.78580748  0.80498987 -1.23619147]
 [-0.46999314  0.22327747  0.21021868 -0.21650811]
 [ 1.05650471 -0.1211663   0.35016485  0.9847627 ]
 ...
 [ 1.52619636  1.22723602  0.66504371 -1.43174719]
 [-1.17453061 -0.83153079 -1.36417564  0.28634944]
 [-1.35066498 -0.3993588   1.0848822  -0.13269852]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

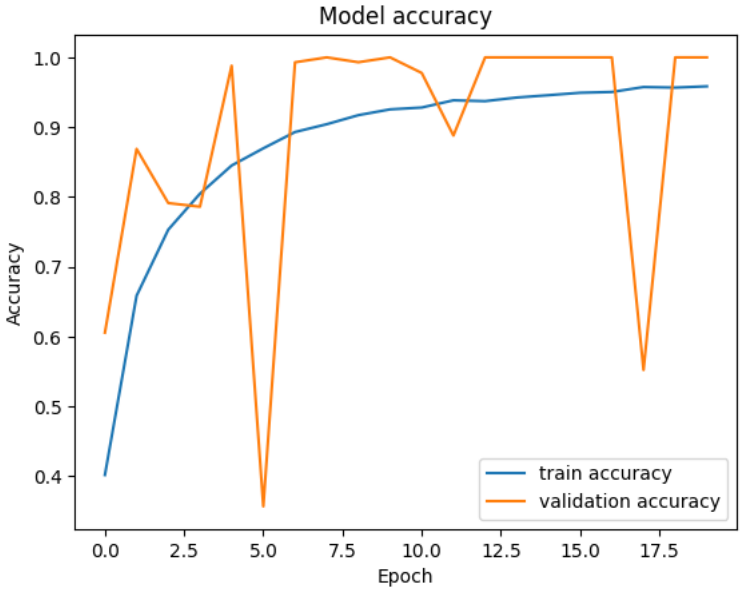
```
from keras.engine import sequential
def get_model():

  model = tf.keras.Sequential([
    #normalizer,

    Dense(50, activation='relu'),
    Dense(50, activation='relu'),

    Dense(60, activation='relu', kernel_regularizer=regularizers.L1(0.01)),

    Dense(70, activation='relu'),
    Dense(80, activation='relu'),
```

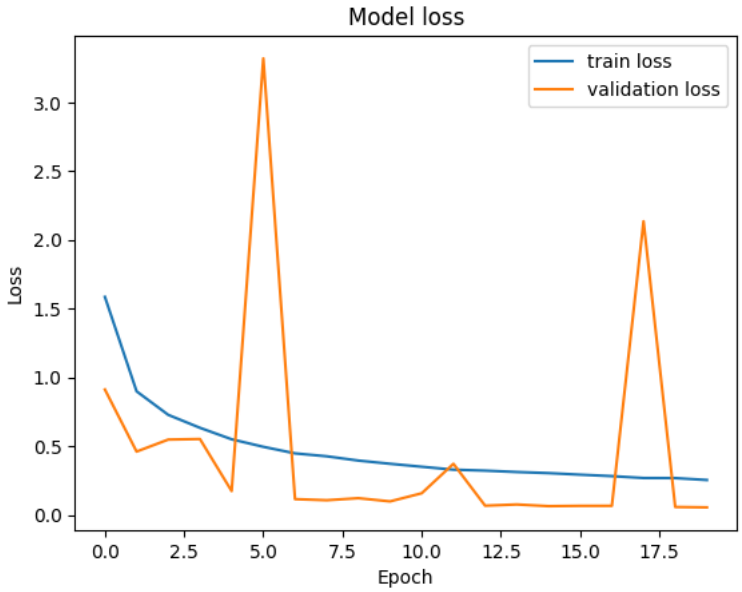
```
    Dense(90, activation='relu', kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4)),

    Dense(100, activation='relu'),



    Dense(banyak_kategori, activation = 'softmax')
  ])
  """
  model = sequential()
  model.add(Dense(50, activation='relu'))
  model.add(Dense(50, activation='relu'))
  model.add(Dense(60, activation='relu'))
  model.add(regularizers.L1L2(l1=1e-5, l2=1e-4))
  model.add(Dense(70, activation='relu'))
  model.add(Dense(80, activation='relu'))
  model.add(Dense(90, activation='relu'))
  model.add(regularizers.L2(1e-4))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(banyak_kategori, activation = 'softmax'))
  """
  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=2e-3),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model


#x_train=np.asarray(x_train).astype(np.int)

#y_train=np.asarray(y_train).astype(np.int)


model = get_model()



model_fit = model.fit(x_train,
                      y_train,
                      epochs = 20,
                      validation_data = (x_test, y_test))
```

```
Epoch 1/20
36221/36221 [==============================] - 140s 4ms/step - loss: 1.5855 - accuracy: 0.4015 - val_loss: 0.9121 - val_a
Epoch 2/20
36221/36221 [==============================] - 131s 4ms/step - loss: 0.8991 - accuracy: 0.6584 - val_loss: 0.4610 - val_a
Epoch 3/20
36221/36221 [==============================] - 129s 4ms/step - loss: 0.7279 - accuracy: 0.7531 - val_loss: 0.5483 - val_a
Epoch 4/20
36221/36221 [==============================] - 139s 4ms/step - loss: 0.6338 - accuracy: 0.8046 - val_loss: 0.5523 - val_a
Epoch 5/20
36221/36221 [==============================] - 140s 4ms/step - loss: 0.5504 - accuracy: 0.8450 - val_loss: 0.1739 - val_a
Epoch 6/20
36221/36221 [==============================] - 146s 4ms/step - loss: 0.4958 - accuracy: 0.8697 - val_loss: 3.3196 - val_a
Epoch 7/20
36221/36221 [==============================] - 133s 4ms/step - loss: 0.4480 - accuracy: 0.8930 - val_loss: 0.1154 - val_a
Epoch 8/20
36221/36221 [==============================] - 146s 4ms/step - loss: 0.4268 - accuracy: 0.9042 - val_loss: 0.1074 - val_a
Epoch 9/20
36221/36221 [==============================] - 144s 4ms/step - loss: 0.3956 - accuracy: 0.9172 - val_loss: 0.1223 - val_a
Epoch 10/20
36221/36221 [==============================] - 155s 4ms/step - loss: 0.3724 - accuracy: 0.9255 - val_loss: 0.0988 - val_a
Epoch 11/20
36221/36221 [==============================] - 136s 4ms/step - loss: 0.3508 - accuracy: 0.9281 - val_loss: 0.1581 - val_a
Epoch 12/20
36221/36221 [==============================] - 140s 4ms/step - loss: 0.3295 - accuracy: 0.9385 - val_loss: 0.3723 - val_a
Epoch 13/20
36221/36221 [==============================] - 141s 4ms/step - loss: 0.3225 - accuracy: 0.9373 - val_loss: 0.0677 - val_a
Epoch 14/20
36221/36221 [==============================] - 162s 4ms/step - loss: 0.3125 - accuracy: 0.9425 - val_loss: 0.0766 - val_a
Epoch 15/20
36221/36221 [==============================] - 147s 4ms/step - loss: 0.3045 - accuracy: 0.9458 - val_loss: 0.0643 - val_a
Epoch 16/20
36221/36221 [==============================] - 152s 4ms/step - loss: 0.2932 - accuracy: 0.9493 - val_loss: 0.0665 - val_a
Epoch 17/20
36221/36221 [==============================] - 139s 4ms/step - loss: 0.2826 - accuracy: 0.9504 - val_loss: 0.0666 - val_a
Epoch 18/20
36221/36221 [==============================] - 146s 4ms/step - loss: 0.2688 - accuracy: 0.9575 - val_loss: 2.1356 - val_a
Epoch 19/20
36221/36221 [==============================] - 138s 4ms/step - loss: 0.2683 - accuracy: 0.9567 - val_loss: 0.0582 - val_a
Epoch 20/20
36221/36221 [==============================] - 147s 4ms/step - loss: 0.2544 - accuracy: 0.9585 - val_loss: 0.0555 - val_a
```

```
def plot_accuracy(history):

    plt.plot(history.history['accuracy'],label='train accuracy')
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
```

```
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc='best')
    plt.savefig('Accuracy_v1_model_inceptionv3')
    plt.show()

def plot_loss(history):

    plt.plot(history.history['loss'],label="train loss")
    plt.plot(history.history['val_loss'],label="validation loss")
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(loc='best')
    plt.savefig('Loss_v1_model_inceptionv3')
    plt.show()

plot_accuracy(model_fit)
plot_loss(model_fit)
```






```
model.save('/content/drive/MyDrive/Capstone/model_exercise.h5')
# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('/content/drive/MyDrive/Capstone/model_exercise.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving (showing 1 of 1). These functions will not be
```

```
predict_x = model.predict(x_test)
classes_x = np.argmax(predict_x,axis=1)
#y_pred_class = model.predict_classes(x_test)

y_pred = model.predict(x_test)
y_test_class = np.argmax(y_test, axis=1)
confusion_matrix(y_test_class, classes_x)
```

```
9056/9056 [==============================] - 15s 2ms/step
9056/9056 [==============================] - 16s 2ms/step
array([[1146,    0,    0, ...,    0,    0,    0],
       [   0, 1218,    0, ...,    0,    0,    0],
       [   0,    0, 1165, ...,    0,    0,    0],
       ...,
       [   0,    0,    0, ..., 1178,    0,    0],
       [   0,    0,    0, ...,    0, 1195,    0],
       [   0,    0,    0, ...,    0,    0, 1203]])
```

```
print(classification_report(y_test_class, classes_x))
```

```
              195       1.00      1.00      1.00      1113
              196       1.00      1.00      1.00      1167
              197       1.00      1.00      1.00      1123
              198       1.00      1.00      1.00      1184
              199       1.00      1.00      1.00      1097
              200       1.00      1.00      1.00      1226
              201       1.00      1.00      1.00      1157
              202       1.00      1.00      1.00      1159
              203       1.00      1.00      1.00      1192
              204       1.00      1.00      1.00      1188
              205       1.00      1.00      1.00      1235
              206       1.00      1.00      1.00      1159
              207       1.00      1.00      1.00      1156
              208       1.00      1.00      1.00      1221
              209       1.00      1.00      1.00      1188
              210       1.00      1.00      1.00      1231
              211       1.00      1.00      1.00      1215
              212       1.00      1.00      1.00      1174
              213       1.00      1.00      1.00      1181
              214       1.00      1.00      1.00      1178
              215       1.00      1.00      1.00      1195
              216       1.00      1.00      1.00      1147
              217       1.00      1.00      1.00      1151
              218       1.00      1.00      1.00      1146
              219       1.00      1.00      1.00      1199
              220       1.00      1.00      1.00      1140
              221       1.00      1.00      1.00      1185
              222       1.00      1.00      1.00      1144
              223       1.00      1.00      1.00      1179
              224       1.00      1.00      1.00      1212
              225       1.00      1.00      1.00      1186
              226       1.00      1.00      1.00      1140
              227       1.00      1.00      1.00      1164
              228       1.00      1.00      1.00      1128
              229       1.00      1.00      1.00      1170
              230       1.00      1.00      1.00      1195
              231       1.00      1.00      1.00      1128
              232       1.00      1.00      1.00      1232
              233       1.00      1.00      1.00      1193
              234       1.00      1.00      1.00      1128
              235       1.00      1.00      1.00      1175
              236       1.00      1.00      1.00      1136
              237       1.00      1.00      1.00      1153
              238       1.00      1.00      1.00      1181
              239       1.00      1.00      1.00      1221
              240       1.00      1.00      1.00      1172
              241       1.00      1.00      1.00      1169
              242       1.00      1.00      1.00      1226
              243       1.00      1.00      1.00      1179
              244       1.00      1.00      1.00      1154
              245       1.00      1.00      1.00      1178
              246       1.00      1.00      1.00      1195
              247       1.00      1.00      1.00      1203

         accuracy                           1.00    289764
        macro avg       1.00      1.00      1.00    289764
     weighted avg       1.00      1.00      1.00    289764
```

```
report = classification_report(y_test_class, classes_x, output_dict=True)

# Extract the metrics
```

```
precision = report['macro avg']['precision']
recall = report['macro avg']['recall']
f1_score = report['macro avg']['f1-score']
support = report['macro avg']['support']
accuracy = report['accuracy']

print("accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
print("support" , support)
```

```
accuracy: 0.9999930978313386
Precision: 0.9999932907519725
Recall: 0.9999931887532694
F1-score: 0.9999932340806261
support 289764
```

✓ 1s completed at 7:00 PM