```
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 import metrics
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)
```

```
Mounted at /content/drive
  Activity, Exercise or Sport (1 hour) Intensity Description  \
0          Cycling, mountain bike, bmx                    NaN
1  Cycling, <10 mph, leisure bicycling                    NaN
                                                          NaN
                                                          NaN
                                                          NaN
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
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())
```

```
224
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
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
                                                  ation (minutes)  \
                                                               60
                                                               60
                                                               60
                                                               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
3	1	0.145894	5

```
"""
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='re
(banyak_kategori, activation = 'softmax')\n  ])\n\n  model.compile(
egorical_crossentropy',\n                metrics=['accuracy'])\n  re
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
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)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
print(y_test)
```

```
[[-1.35066498e+00 -4.83891605e-01  1.22482836e+00 -7.61270455e-01]
 [ 6.45524518e-01 -3.50932742e-01  2.99439752e-04 -8.45080046e-01]
 [-5.90129526e-02 -4.60763979e-01 -1.74633263e-01  1.40381066e+00]
 ...
 [-4.69993144e-01  2.35554652e-01  7.35016792e-01 -1.38984239e+00]
 [-1.64422226e+00 -9.10832216e-01 -1.01431024e+00 -4.81905150e-01]
 [-1.76435864e-01  4.43372659e-02 -6.96736414e-02 -1.13841362e+00]]
[[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.]]
[[-1.70293372 -0.89776039  1.67965339  0.16063505]
 [-0.46999314 -0.44537205  0.59507063 -1.264128  ]
 [ 0.64552452  1.44016513  0.42013793 -0.16063505]
 ...
 [-0.76355042 -0.46448915  0.80498987 -0.71936566]
 [-0.58741606  0.61334078  1.15485528  1.11047709]
 [ 1.17392762 -0.02819828 -0.06967364  1.32000106]]
[[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([
        Dense(50, activation='relu'),
        Dense(50, activation='relu'),
        Dense(60, activation='relu'),
        Dense(70, activation='relu'),
        Dense(80, activation='relu'),
        Dense(90, activation='relu'),
        Dense(100, activation='relu'),
        Dense(banyak_kategori, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
```

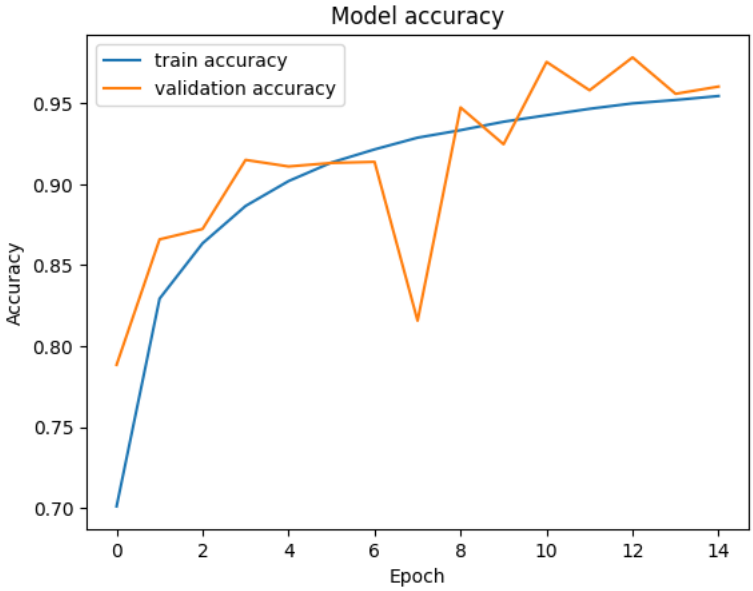
```
                  metrics=['accuracy'])
    return model


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

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


my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_loss:.2f}.h5'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]


model = get_model()



model_fit = model.fit(x_train,
                      y_train,
                      epochs = 15,
                      validation_data = (x_test, y_test))
```

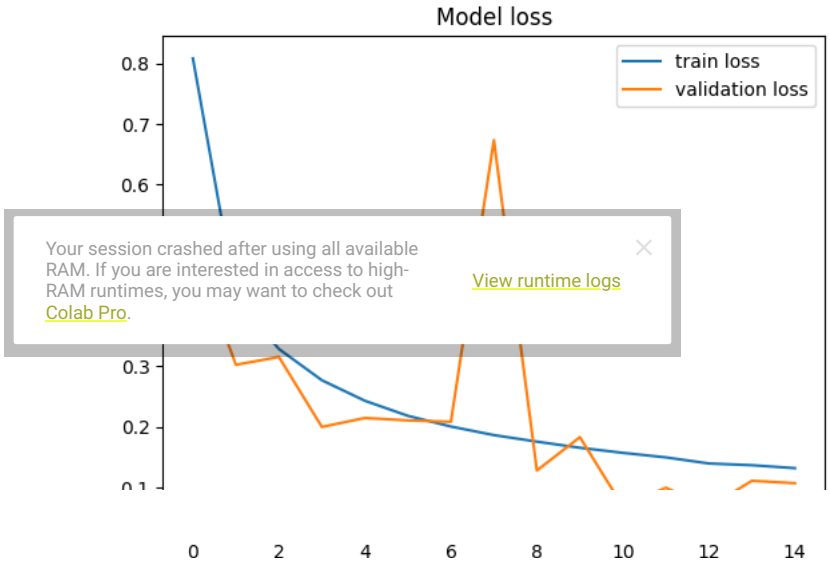
```
Epoch 1/15
36221/36221 [==============================] - 198s 5ms/step - loss: 0.8080 - accuracy: 0.7010 - val_loss: 0.4960 - val_a
Epoch 2/15
36221/36221 [==============================] - 172s 5ms/step - loss: 0.4120 - accuracy: 0.8293 - val_loss: 0.3023 - val_a
Epoch 3/15
36221/36221 [==============================] - 177s 5ms/step - loss: 0.3285 - accuracy: 0.8635 - val_loss: 0.3153 - val_a
Epoch 4/15
36221/36221 [==============================] - 168s 5ms/step - loss: 0.2768 - accuracy: 0.8866 - val_loss: 0.1998 - val_a
Epoch 5/15
36221/36221 [==============================] - 181s 5ms/step - loss: 0.2430 - accuracy: 0.9019 - val_loss: 0.2146 - val_a
```



```
ms/step - loss: 0.2181 - accuracy: 0.9134 - val_loss: 0.2106 - val_a
ms/step - loss: 0.2005 - accuracy: 0.9215 - val_loss: 0.2084 - val_a
ms/step - loss: 0.1864 - accuracy: 0.9287 - val_loss: 0.6731 - val_a
Epoch 9/15
36221/36221 [==============================] - 180s 5ms/step - loss: 0.1756 - accuracy: 0.9334 - val_loss: 0.1280 - val_a
Epoch 10/15
36221/36221 [==============================] - 180s 5ms/step - loss: 0.1654 - accuracy: 0.9386 - val_loss: 0.1829 - val_a
Epoch 11/15
36221/36221 [==============================] - 176s 5ms/step - loss: 0.1571 - accuracy: 0.9426 - val_loss: 0.0686 - val_a
Epoch 12/15
36221/36221 [==============================] - 196s 5ms/step - loss: 0.1496 - accuracy: 0.9466 - val_loss: 0.0995 - val_a
Epoch 13/15
36221/36221 [==============================] - 181s 5ms/step - loss: 0.1395 - accuracy: 0.9499 - val_loss: 0.0671 - val_a
Epoch 14/15
36221/36221 [==============================] - 182s 5ms/step - loss: 0.1367 - accuracy: 0.9520 - val_loss: 0.1110 - val_a
Epoch 15/15
36221/36221 [==============================] - 183s 5ms/step - loss: 0.1318 - accuracy: 0.9545 - val_loss: 0.1070 - 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)
```





Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
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 = confusion_matrix(y_test_class, classes_x)
```

```
9056/9056 [==============================] - 21s 2ms/step
9056/9056 [==============================] - 18s 2ms/step
```

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

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93      1170
           1       0.96      0.89      0.93      1160
           2       0.99      0.93      0.96      1182
           3       0.95      0.99      0.97      1155
           4       0.99      0.97      0.98      1171
           5       0.97      0.97      0.97      1189
           6       0.81      0.96      0.88      1142
           7       0.96      0.77      0.85      1126
           8       0.98      0.93      0.95      1178
```

9	0.94	0.99	0.97	1204
10	1.00	0.98	0.99	1122
11	0.89	1.00	0.94	1167
12	0.98	0.89	0.93	1203
13	0.97	0.96	0.96	1181
14	0.92	1.00	0.96	1182
15	0.90	0.92	0.91	1143
16	0.96	0.84	0.90	1177
17	0.87	0.94	0.90	1169
18	1.00	0.73	0.84	1192
19	0.84	0.97	0.90	1155
20	0.99	1.00	0.99	1147
21	0.96	0.99	0.98	1200
22	0.99	1.00	1.00	1129
23	1.00	0.96	0.98	1177
24	0.98	1.00	0.99	1183
25	0.97	1.00	0.99	1166
26	1.00	0.97	0.99	1091
27	0.98	0.99	0.99	1207
28	0.99	1.00	0.99	1156
29	1.00	0.94	0.97	1187
30	0.95	0.96	0.96	1152
31	0.98	0.84	0.90	1109
32	0.85	1.00	0.92	1180
33	1.00	0.93	0.97	1161
34	0.94	0.98	0.96	1120
35	0.98	1.00	0.99	1159
36	1.00	1.00	1.00	1151
37	1.00	0.97	0.98	1215
38	0.96	1.00	0.98	1144
39	0.97	0.99	0.98	1119
40	1.00	0.96	0.98	1135
41	0.83	1.00	0.90	1165
42	0.89	0.93	0.91	1138
43	0.95	0.99	0.97	1150
44	0.87	0.74	0.80	1198
51	0.99	0.98	0.98	1171
52	0.96	0.98	0.97	1205
53	0.67	0.91	0.77	1167
54	1.00	0.94	0.97	1166

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
report = classification_report(y_test_class, classes_x, output_dict=True, zero_division=0)


# 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: 0.9603090791126572
Precision: 0.9660990410937471
Recall: 0.9608365246358126
F1-score: 0.9592716520206644
support 289764
```

```
def plot_confusion_matrix(matrix, labels, title='Confusion matrix'):
    fig, ax  = plt.subplots()
    ax.set_xticks([x for x in range(len(labels))])
    ax.set_yticks([y for y in range(len(labels))])
    # Place labels on minor ticks
    ax.set_xticks([x + 0.5 for x in range(len(labels))], minor=True)
    ax.set_xticklabels(labels, rotation='90', fontsize=10, minor=True)
    ax.set_yticks([y + 0.5 for y in range(len(labels))], minor=True)
    ax.set_yticklabels(labels[::-1], fontsize=10, minor=True)
    # Hide major tick labels
    ax.tick_params(which='major', labelbottom='off', labelleft='off')
    # Finally, hide minor tick marks
    ax.tick_params(which='minor', width=0)

    # Plot heat map
    proportions = [1. * row / sum(row) for row in matrix]
```

```
    ax.pcolor(np.array(proportions[::-1]), cmap=plt.cm.Blues)

    # Plot counts as text
    for row in range(len(matrix)):
        for col in range(len(matrix[row])):
            confusion = matrix[::-1][row][col]
            if confusion != 0:
                ax.text(col + 0.5, row + 0.5, confusion, fontsize=9,
                    horizontalalignment='center',
                    verticalalignment='center')

    # Add finishing touches
    ax.grid(True, linestyle=':')
    ax.set_title(title)
    fig.tight_layout()
    plt.show()


#print(type(confusion_matrix(y_test_class, classes_x)))
#print(y_test_class)
#print(y_test)
#print(len(y_test_class))


dict_activity = dict(enumerate(df_new['activity'].cat.categories))
df_new['activity_code'] = df_new['activity'].cat.codes
print(df_new['activity_code'])
print(dict_activity)
df_new['activity_reversed'] = df_new['activity_code'].map(dict_activity)
df_y_test_class = pd.DataFrame(y_test_class, columns = ['activity_class'])
df_y_test_class['activity_class_reversed'] = df_y_test_class['activity_class'].map(dict_activity)
print(df_y_test_class)
```

```
0           61
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
1448811     73
1448812     40
1448813    207
1448814    232
1448815    208
Name: activity_code, Length: 1448816, dtype: int16
{0: 'Aerobics, general', 1: 'Aerobics, high impact', 2: 'Aerobics, low impact', 3: 'Aerobics, step aerobics', 4: 'Archery
        activity_class                   activity_class_reversed
0                  135                       Riding, snow blower
1                   33             Carrying infant, level ground
2                  112       Martial arts, judo, karate, jujitsu
3                  123                           Playing paddleball
4                   54                                   Curling
...                ...                                      ...
289759             199            Swimming laps, freestyle, slow
289760              71                 Frisbee, ultimate frisbee
289761              72                       Gardening, general
289762             203  Swimming, treading water, fast, vigorous
289763             218          Walk / run, playing with animals

[289764 rows x 2 columns]
```

```
#cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix)
#cm_display.plot()
#plt.show()
```


```
#import seaborn as sns
#sns.heatmap(confusion_matrix,figsize=(200,200), annot=True)
```



```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

df_cm = pd.DataFrame(confusion_matrix(y_test_class, classes_x), columns=np.unique(y_test_class), index=np.unique(classes_x))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'


f, ax = plt.subplots(figsize=(500, 500))
cmap = sns.cubehelix_palette(light=1, as_cmap=True)

sns.heatmap(df_cm, cbar=False, annot=True, cmap=cmap, square=True, fmt='.0f',
            annot_kws={'size': 10})
plt.title('Actuals vs Predicted')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-32b9e77ce9e7> in <cell line: 6>()
      4 get_ipython().run_line_magic('matplotlib', 'inline')
      5
----> 6 df_cm = pd.DataFrame(confusion_matrix(y_test_class, classes_x), columns=np.unique(y_test_class), index=np.unique(
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

SEARCH STACK OVERFLOW

6s completed at 10:32 PM