```
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
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())
```

```
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
             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
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='rel
se(10, activation='relu'),\n    tf.keras.layers.Dense(banyak_kategori, activation = 'softmax')\n  ])\n\n  model.compile(c
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)
```

```
[[ 1.64361927  0.04188652  0.03528598 -0.70539739]
 [ 0.11712142  0.21761024  0.31517831 -1.52952504]
 [ 1.58490781  1.28834922  0.07027252 -0.45396862]
 ...
 [ 1.11521616  0.24530705  1.2598149  -1.5853981 ]
 [ 0.46939015  0.32766061 -0.52449867 -0.27238117]
 [ 1.35006199  0.21030979 -0.87436408 -0.77523872]]
[[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.]]
[[-0.05901295 -0.37253129 -0.87436408 -1.23619147]
 [ 0.11712142 -0.09042097  1.53970723  1.48762025]
 [-1.35066498 -0.81918047 -0.62945829 -1.06857229]
 ...
 [ 0.52810161  0.96034546 -0.24460634 -1.33396933]
 [-0.35257023 -0.0394629   0.38515139 -1.72508076]
 [-0.23514732 -0.12555849  0.17523214  1.06857229]]
[[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. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

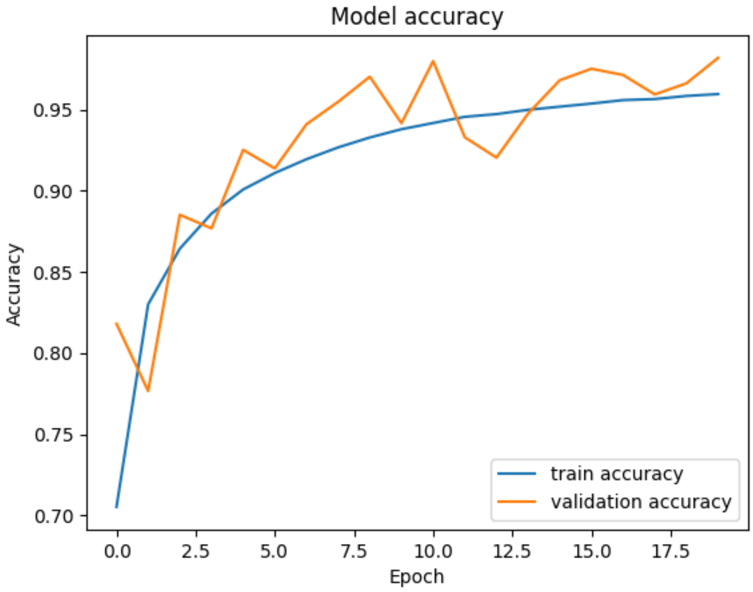
```
from keras.engine import sequential
def get_model():
    model =  tf.keras.Sequential([
        Dense(50, activation='relu'),
        Dense(50, activation='relu'),
        Dense(60, activation='relu'),
        Dense(70, activation='relu'),
        Dense(80, activation='relu'),
        Dense(90, activation='relu'),
        Dense(100, activation='relu'),
        Dense(banyak_kategori, activation='softmax')
    ])

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

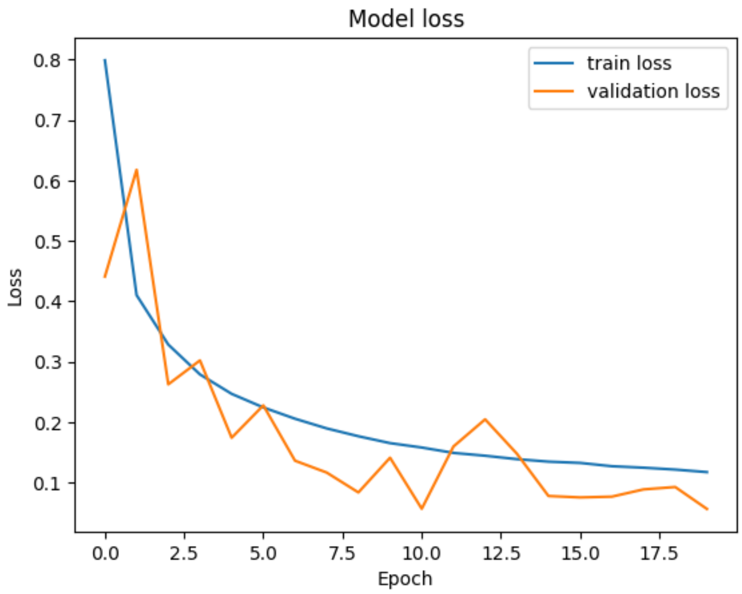
```
                 metrics=['accuracy'])
    return model


#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 = 20,
                      validation_data = (x_test, y_test))
```

```
Epoch 1/20
36221/36221 [==============================] - 227s 6ms/step - loss: 0.7989 - accuracy: 0.7052 - val_loss: 0.4410 - val_a
Epoch 2/20
36221/36221 [==============================] - 216s 6ms/step - loss: 0.4102 - accuracy: 0.8300 - val_loss: 0.6178 - val_a
Epoch 3/20
36221/36221 [==============================] - 217s 6ms/step - loss: 0.3284 - accuracy: 0.8643 - val_loss: 0.2626 - val_a
Epoch 4/20
36221/36221 [==============================] - 219s 6ms/step - loss: 0.2790 - accuracy: 0.8859 - val_loss: 0.3022 - val_a
Epoch 5/20
36221/36221 [==============================] - 219s 6ms/step - loss: 0.2469 - accuracy: 0.9008 - val_loss: 0.1742 - val_a
Epoch 6/20
36221/36221 [==============================] - 200s 6ms/step - loss: 0.2244 - accuracy: 0.9109 - val_loss: 0.2278 - val_a
Epoch 7/20
36221/36221 [==============================] - 217s 6ms/step - loss: 0.2057 - accuracy: 0.9194 - val_loss: 0.1362 - val_a
Epoch 8/20
36221/36221 [==============================] - 201s 6ms/step - loss: 0.1897 - accuracy: 0.9267 - val_loss: 0.1166 - val_a
Epoch 9/20
36221/36221 [==============================] - 199s 6ms/step - loss: 0.1767 - accuracy: 0.9328 - val_loss: 0.0837 - val_a
Epoch 10/20
36221/36221 [==============================] - 202s 6ms/step - loss: 0.1653 - accuracy: 0.9379 - val_loss: 0.1411 - val_a
Epoch 11/20
36221/36221 [==============================] - 200s 6ms/step - loss: 0.1580 - accuracy: 0.9417 - val_loss: 0.0567 - val_a
Epoch 12/20
36221/36221 [==============================] - 196s 5ms/step - loss: 0.1491 - accuracy: 0.9455 - val_loss: 0.1597 - val_a
Epoch 13/20
36221/36221 [==============================] - 197s 5ms/step - loss: 0.1444 - accuracy: 0.9472 - val_loss: 0.2047 - val_a
Epoch 14/20
36221/36221 [==============================] - 204s 6ms/step - loss: 0.1388 - accuracy: 0.9498 - val_loss: 0.1480 - val_a
Epoch 15/20
36221/36221 [==============================] - 204s 6ms/step - loss: 0.1347 - accuracy: 0.9518 - val_loss: 0.0778 - val_a
Epoch 16/20
36221/36221 [==============================] - 263s 7ms/step - loss: 0.1325 - accuracy: 0.9537 - val_loss: 0.0754 - val_a
Epoch 17/20
36221/36221 [==============================] - 220s 6ms/step - loss: 0.1271 - accuracy: 0.9558 - val_loss: 0.0767 - val_a
Epoch 18/20
36221/36221 [==============================] - 218s 6ms/step - loss: 0.1246 - accuracy: 0.9565 - val_loss: 0.0887 - val_a
Epoch 19/20
36221/36221 [==============================] - 200s 6ms/step - loss: 0.1215 - accuracy: 0.9584 - val_loss: 0.0926 - val_a
Epoch 20/20
36221/36221 [==============================] - 218s 6ms/step - loss: 0.1173 - accuracy: 0.9595 - val_loss: 0.0564 - 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 = confusion_matrix(y_test_class, classes_x)
```

```
    9056/9056 [==============================] - 17s 2ms/step
    9056/9056 [==============================] - 16s 2ms/step
```

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

```
         195       0.99      1.00      0.99      1166
         196       1.00      0.97      0.98      1167
         197       1.00      0.76      0.86      1190
         198       0.78      1.00      0.88      1124
         199       1.00      0.99      0.99      1182
         200       0.99      1.00      0.99      1184
         201       1.00      0.78      0.88      1185
         202       0.80      1.00      0.89      1136
         203       1.00      0.98      0.99      1170
         204       1.00      1.00      1.00      1180
         205       0.99      1.00      0.99      1196
         206       0.99      0.99      0.99      1133
         207       1.00      0.99      1.00      1209
         208       1.00      0.99      1.00      1145
         209       0.99      1.00      1.00      1121
         210       1.00      0.99      1.00      1196
         211       1.00      1.00      1.00      1131
         212       1.00      1.00      1.00      1160
         213       1.00      1.00      1.00      1175
         214       1.00      0.98      0.99      1208
         215       0.97      1.00      0.99      1155
         216       0.98      1.00      0.99      1172
         217       1.00      0.98      0.99      1159
         218       1.00      1.00      1.00      1198
         219       1.00      1.00      1.00      1151
         220       1.00      1.00      1.00      1148
         221       1.00      1.00      1.00      1211
         222       1.00      0.97      0.98      1191
         223       0.96      1.00      0.98      1184
         224       0.99      0.99      0.99      1151
         225       1.00      1.00      1.00      1149
         226       0.99      1.00      1.00      1228
         227       0.99      0.99      0.99      1114
         228       1.00      0.98      0.99      1172
         229       0.98      0.84      0.91      1202
         230       0.85      1.00      0.92      1205
         231       1.00      0.97      0.99      1175
         232       0.99      1.00      1.00      1145
         233       0.99      1.00      1.00      1169
         234       1.00      0.99      1.00      1190
         235       1.00      1.00      1.00      1177
         236       0.97      1.00      0.98      1144
         237       1.00      0.97      0.98      1156
         238       0.98      1.00      0.99      1139
         239       1.00      0.98      0.99      1177
         240       0.99      1.00      0.99      1070
         241       1.00      0.99      1.00      1176
         242       1.00      1.00      1.00      1160
         243       1.00      1.00      1.00      1122
         244       1.00      1.00      1.00      1184
         245       1.00      1.00      1.00      1098
         246       1.00      1.00      1.00      1178
         247       1.00      1.00      1.00      1166

    accuracy                           0.98    289764
   macro avg       0.98      0.98      0.98    289764
weighted avg       0.98      0.98      0.98    289764
```

```
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: 0.9818472964205354
Precision: 0.9831678996834177
Recall: 0.9819389461042477
F1-score: 0.9818066971646372
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))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-fa9251af7942> in <cell line: 1>()
----> 1 print(type(confusion_matrix(y_test_class, classes_x)))
      2 print(y_test_class)
      3 print(y_test)
      4 print(len(y_test_class))

TypeError: 'numpy.ndarray' object is not callable
```

SEARCH STACK OVERFLOW

```
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)
```

```
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=(100, 100))
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.title('Actuals vs Predicted')
plt.show()
```