```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/drive')
! ls
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
drive  sample_data
```

```
df_org = pd.read_csv('/content/drive/My Drive/1-Master/Projekte und Ideen/fussball/data/df_final.
```

```
df = df_org.copy()
```

```
df = df.drop('Unnamed: 0', axis=1)
```

```
df.head()
```

	competition	saison	gameday	date	home_team_name	guest_team_name	home_coach
0	Bundesliga	2009/10	1	2009-07-18 19:30:00	SV Kapfenberg	SV Ried	Werne Gregoritsch
1	Bundesliga	2009/10	2	2009-07-25 19:30:00	SV Ried	Red Bull Salzburg	Pau Gludovatz
2	Bundesliga	2009/10	3	2009-07-31 20:30:00	SC Wiener Neustadt	SV Ried	Helmu Kraf
3	Bundesliga	2009/10	4	2009-09-08 19:30:00	SV Ried	SK Rapid Wien	Pau Gludovatz
4	Bundesliga	2009/10	5	2009-08-21 20:30:00	SK Austria Kärnten	SV Ried	Frenkie Schinkels

```
df.shape
```

```
(2509, 39)
```

```
y_reg = df.resultNumeric
y_cat = (pd.get_dummies(np.where(df.resultNumeric < 0, 'lost', np.where(df.resultNumeric > 0, 'wo
```

```
ndf = df.shape[0]
df = df.drop(['resultNumeric'], axis=1)
df.shape
```

```
(2509, 38)
```

```
ndf
```

```
2509
```

```
df.head()
```

	competition	saison	gameday	date	home_team_name	guest_team_name	home_coach
0	Bundesliga	2009/10	1	2009-07-18 19:30:00	SV Kapfenberg	SV Ried	Werne Gregoritsch
1	Bundesliga	2009/10	2	2009-07-25 19:30:00	SV Ried	Red Bull Salzburg	Pau Gludovatz
2	Bundesliga	2009/10	3	2009-07-31 20:30:00	SC Wiener Neustadt	SV Ried	Helmu Kraf
3	Bundesliga	2009/10	4	2009-09-08 19:30:00	SV Ried	SK Rapid Wien	Pau Gludovatz
4	Bundesliga	2009/10	5	2009-08-21 20:30:00	SK Austria Kärnten	SV Ried	Frenkie Schinkels

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2509 entries, 0 to 2508
Data columns (total 38 columns):
competition                    2509 non-null object
saison                         2509 non-null object
gameday                        2509 non-null object
date                           2509 non-null object
home_team_name                 2509 non-null object
guest_team_name                2509 non-null object
home_coach                     2509 non-null object
home_game_system               2509 non-null object
home_team_id                   2509 non-null int64
guest_coach                    2509 non-null object
guest_game_system              2509 non-null object
guest_team_id                  2509 non-null int64
time                           2509 non-null object
viewers                        2509 non-null int64
weekday                        2509 non-null int64
day                            2509 non-null int64
month                          2509 non-null int64
year                           2509 non-null int64
homeTotalGoalsShoot            2509 non-null float64
homeTotalGoalsReceived         2509 non-null float64
homeTotalGoalDiff              2509 non-null float64
guestTotalGoalsShoot           2509 non-null float64
guestTotalGoalsReceived        2509 non-null float64
guestTotalGoalDiff             2509 non-null float64
homeTeamGoalsShootAtHome       2509 non-null float64
homeTeamGoalsReceivedAtHome    2509 non-null float64
homeTeamGoalsDiffAtHome        2509 non-null float64
homeTeamGoalsShootAway         2509 non-null float64
homeTeamGoalsReceivedAway      2509 non-null float64
homeTeamGoalsDiffAway          2509 non-null float64
guestTeamGoalsShootAtHome      2509 non-null float64
guestTeamGoalsReceivedAtHome   2509 non-null float64
guestTeamGoalsDiffAtHome       2509 non-null float64
guestTeamGoalsShootAway        2509 non-null float64
guestTeamGoalsReceivedAway     2509 non-null float64
guestTeamGoalsDiffAway         2509 non-null float64
```

```
df = df.drop(['date', 'home_team_id', 'guest_team_id'], axis=1)
```

```
dtypes: float64(26), int64(7), object(11)
```

```
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

```
categorial_variables = df.columns[:17]
categorial_variables = categorial_variables.drop('viewers')
categorial_variables
```

```
df_cat = pd.get_dummies(df[categorial_variables])
df = pd.concat([df, df_cat], axis=1)
df = df.drop(categorial_variables, axis=1)
```

```
df.shape
```

```
df.head()
```

```
df.info()
```

```
X = df
```

```
X.shape, y_reg.shape, y_cat.shape
```

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_reg_train, y_reg_test, y_class_train, y_class_test = train_test_split(
    X, y_reg, y_cat, test_size=0.2, random_state=123)


from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)



from keras.models import Model, Sequential
from keras.layers import Dense, Input, BatchNormalization
```

```
from keras.utils import plot_model
from keras.callbacks import EarlyStopping, ModelCheckpoint

early_stopping_monitor = EarlyStopping(patience=4)


model_save = ModelCheckpoint('best_model.hdf5', save_best_only=True, monitor='val_Classification_


in_cols=X_train.shape[1]
out_cols=y_cat.shape[1]

in_cols, out_cols
```

```
(761, 3)
```

```
model_1 = Sequential()

model_1.add(Dense(12, input_shape=(in_cols,), activation='relu', ))
model_1.add(Dense(12, activation='relu'))
model_1.add(Dense(1))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0811 19:27:23.738497 140698957793152 deprecation_wrapper.py:119] From /usr/local/lib

W0811 19:27:23.772591 140698957793152 deprecation_wrapper.py:119] From /usr/local/lib

W0811 19:27:23.779088 140698957793152 deprecation_wrapper.py:119] From /usr/local/lib
```

```
model_1.compile(optimizer='adam', loss='mse')
```

```
W0811 19:27:37.624620 140698957793152 deprecation_wrapper.py:119] From /usr/local/lib
```

```
model_1_training = model_1.fit(
    X_train_scaled,
    y_reg_train,
    validation_data=[X_test_scaled,y_reg_test],
    epochs=100
)
```

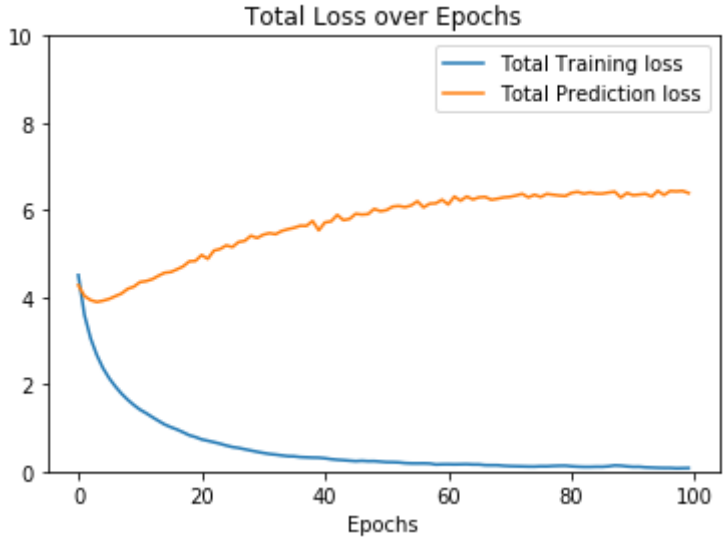
```
W0811 19:27:40.804023 140698957793152 deprecation_wrapper.py:119] From /usr/local/lib

W0811 19:27:40.887451 140698957793152 deprecation_wrapper.py:119] From /usr/local/lib

Train on 2007 samples, validate on 502 samples
Epoch 1/100
2007/2007 [==============================] - 4s 2ms/step - loss: 4.5064 - val_loss: 4
Epoch 2/100
2007/2007 [==============================] - 0s 113us/step - loss: 3.5887 - val_loss:
Epoch 3/100
2007/2007 [==============================] - 0s 105us/step - loss: 3.0538 - val_loss:
Epoch 4/100
2007/2007 [==============================] - 0s 111us/step - loss: 2.6709 - val_loss:
Epoch 5/100
2007/2007 [==============================] - 0s 102us/step - loss: 2.3718 - val_loss:
Epoch 6/100
2007/2007 [==============================] - 0s 105us/step - loss: 2.1411 - val_loss:
Epoch 7/100
2007/2007 [==============================] - 0s 117us/step - loss: 1.9524 - val_loss:
Epoch 8/100
2007/2007 [==============================] - 0s 109us/step - loss: 1.7874 - val_loss:
Epoch 9/100
2007/2007 [==============================] - 0s 108us/step - loss: 1.6480 - val_loss:
Epoch 10/100
2007/2007 [==============================] - 0s 104us/step - loss: 1.5255 - val_loss:
Epoch 11/100
2007/2007 [==============================] - 0s 107us/step - loss: 1.4196 - val_loss:
Epoch 12/100
2007/2007 [==============================] - 0s 109us/step - loss: 1.3376 - val_loss:
Epoch 13/100
2007/2007 [==============================] - 0s 104us/step - loss: 1.2474 - val_loss:
Epoch 14/100
2007/2007 [==============================] - 0s 105us/step - loss: 1.1617 - val_loss:
Epoch 15/100
2007/2007 [==============================] - 0s 102us/step - loss: 1.0787 - val_loss:
Epoch 16/100
2007/2007 [==============================] - 0s 111us/step - loss: 1.0169 - val_loss:
Epoch 17/100
2007/2007 [==============================] - 0s 107us/step - loss: 0.9631 - val_loss:
Epoch 18/100
2007/2007 [==============================] - 0s 108us/step - loss: 0.8990 - val_loss:
Epoch 19/100
2007/2007 [==============================] - 0s 104us/step - loss: 0.8309 - val_loss:
Epoch 20/100
2007/2007 [==============================] - 0s 104us/step - loss: 0.7896 - val_loss:
Epoch 21/100
2007/2007 [==============================] - 0s 110us/step - loss: 0.7353 - val_loss:
Epoch 22/100
2007/2007 [==============================] - 0s 101us/step - loss: 0.7040 - val_loss:
Epoch 23/100
2007/2007 [==============================] - 0s 102us/step - loss: 0.6710 - val_loss:
Epoch 24/100
2007/2007 [==============================] - 0s 104us/step - loss: 0.6363 - val_loss:
Epoch 25/100
```

```
plt.plot(model_1_training.history['loss'])
plt.xlabel('Epochs')
plt.ylim([0, 10])
plt.title('Total Loss over Epochs')
plt.plot(model_1_training.history['val_loss'])
plt.legend(['Total Training loss', 'Total Prediction loss'])
```

```
<matplotlib.legend.Legend at 0x7ff6a41ede48>
```




```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


rfr = RandomForestRegressor(n_estimators=1000)

rfr.fit(X_train, y_reg_train)
rfr.score(X_train, y_reg_train)
```

```


rfr.score(X_test, y_reg_test)
pred_y = rfr.predict(X_test)


np.sqrt(mean_squared_error(np.exp(y_reg_test), np.exp(pred_y)))
```

```


input_tensor = Input(shape=(in_cols,))
hidden_1 = Dense(8, activation='relu', name='hidden_1')(input_tensor)
batch_1 = BatchNormalization()(hidden_1)
hidden_2 = Dense(16, activation='relu', name='hidden_2')(batch_1)
batch_2 = BatchNormalization()(hidden_2)
hidden_3 = Dense(16, activation='relu', name='hidden_3')(batch_2)
output_tensor_reg = Dense(1, name='Regression')(hidden_3)

output_tensor_class = Dense(out_cols, activation='sigmoid', name='Classification')(output_tensor_


model_2 = Model(input_tensor, [output_tensor_reg, output_tensor_class])


model_2.compile(loss=['mse', 'categorical_crossentropy'],
              optimizer='adam',
              metrics=['accuracy'])


model_2.summary()
```

```
model_2_training = model_2.fit(
    X_train_scaled,
    [y_reg_train, y_class_train],
    validation_data=(X_test_scaled, [y_reg_test, y_class_test]),
    batch_size=1,
    epochs=50,
    callbacks=[early_stopping_monitor],
    verbose=True)
```

```
input_tensor = Input(shape=(in_cols,))
hidden_1 = Dense(in_cols, activation='relu', name='hidden_1')(input_tensor)
batch_1 = BatchNormalization()(hidden_1)
hidden_2 = Dense(in_cols, activation='relu', name='hidden_2')(batch_1)
output_tensor_reg = Dense(1, name='Regression')(hidden_2)

output_tensor_class = Dense(out_cols, activation='sigmoid', name='Classification')(output_tensor_
```

```
model_3 = Model(input_tensor, [output_tensor_reg, output_tensor_class])
```

```
model_3.compile(loss=['mse', 'categorical_crossentropy'],
              optimizer='adam',
              metrics=['accuracy'])
```

```
model_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 761)               0
_________________________________________________________________
hidden_1 (Dense)             (None, 761)               579882
_________________________________________________________________
batch_normalization_3 (Batch (None, 761)               3044
_________________________________________________________________
hidden_2 (Dense)             (None, 761)               579882
_________________________________________________________________
Regression (Dense)           (None, 1)                 762
_________________________________________________________________
Classification (Dense)       (None, 3)                 6
=================================================================
Total params: 1,163,576
Trainable params: 1,162,054
Non-trainable params: 1,522
_________________________________________________________________
```

```
model_3_training = model_3.fit(
    X_train_scaled,
    [y_reg_train, y_class_train],
    validation_data=(X_test_scaled, [y_reg_test, y_class_test]),
    batch_size=1,
    epochs=100,
    callbacks=[early_stopping_monitor],
    verbose=True)
```

```
Train on 2007 samples, validate on 502 samples
Epoch 1/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5139 - Regression
Epoch 2/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5137 - Regression
Epoch 3/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5140 - Regression
Epoch 4/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5140 - Regression
Epoch 5/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5141 - Regression
Epoch 6/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5138 - Regression
Epoch 7/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5138 - Regression
Epoch 8/100
2007/2007 [==============================] - 12s 6ms/step - loss: 5.5138 - Regression
```

```
model_3.predict(X_test_scaled)[0][6][0], model_3.predict(X_test_scaled)[1][6]
```

```
(-3.9668374, array([0.8196118, 0.9541346, 0.5163504], dtype=float32))
```

```
input_tensor = Input(shape=(in_cols,))
hidden_1 = Dense(in_cols*2, activation='relu', name='hidden_1')(input_tensor)
hidden_2 = Dense(in_cols*2, activation='relu', name='hidden_2')(hidden_1)
hidden_3 = Dense(in_cols*4, activation='relu', name='hidden_3')(hidden_2)
hidden_4 = Dense(in_cols*4, activation='relu', name='hidden_4')(hidden_3)
output_tensor_reg = Dense(1, name='Regression')(hidden_4)

output_tensor_class = Dense(out_cols, activation='sigmoid', name='Classification')(output_tensor_
```

```
model_4 = Model(input_tensor, [output_tensor_reg, output_tensor_class])


model_4.compile(loss=['mse', 'categorical_crossentropy'],
            optimizer='adam',
            metrics=['accuracy'])


model_4.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 761)               0
_________________________________________________________________
hidden_1 (Dense)             (None, 1522)              1159764
_________________________________________________________________
hidden_2 (Dense)             (None, 1522)              2318006
_________________________________________________________________
hidden_3 (Dense)             (None, 3044)              4636012
_________________________________________________________________
hidden_4 (Dense)             (None, 3044)              9268980
_________________________________________________________________
Regression (Dense)           (None, 1)                 3045
_________________________________________________________________
Classification (Dense)       (None, 3)                 6
=================================================================
Total params: 17,385,813
Trainable params: 17,385,813
Non-trainable params: 0
_________________________________________________________________
```

```
model_4_training = model_4.fit(
    X_train_scaled,
    [y_reg_train, y_class_train],
    validation_data=(X_test_scaled, [y_reg_test, y_class_test]),
    batch_size=1,
    epochs=50,
    callbacks=[early_stopping_monitor, model_save],
    verbose=True)
```

```
Train on 2007 samples, validate on 502 samples
Epoch 1/50
2007/2007 [==============================] - 36s 18ms/step - loss: 0.6008 - Regressio
Epoch 2/50
2007/2007 [==============================] - 36s 18ms/step - loss: 0.5289 - Regressio
Epoch 3/50
2007/2007 [==============================] - 36s 18ms/step - loss: 0.6770 - Regressio
Epoch 4/50
2007/2007 [==============================] - 36s 18ms/step - loss: 0.5620 - Regressio
Epoch 5/50
2007/2007 [==============================] - 36s 18ms/step - loss: 0.6318 - Regressio
```

```
model_3.predict(X_test_scaled)[0][6][0], model_3.predict(X_test_scaled)[1][6]
```

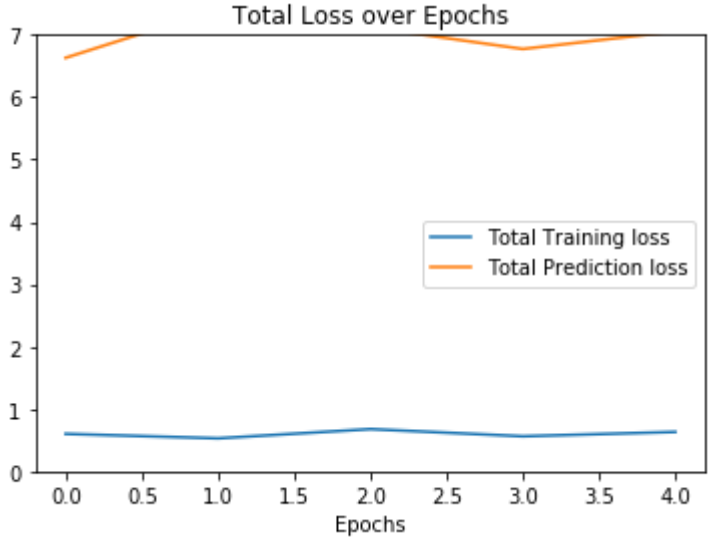
```
(-3.9668374, array([0.8196118, 0.9541346, 0.5163504], dtype=float32))
```

```
model_4.predict(X_test_scaled)[0][6][0], model_4.predict(X_test_scaled)[1][6]
```

```
(0.86977327,
 array([3.9373308e-02, 5.2958727e-05, 3.0636501e-01], dtype=float32))
```

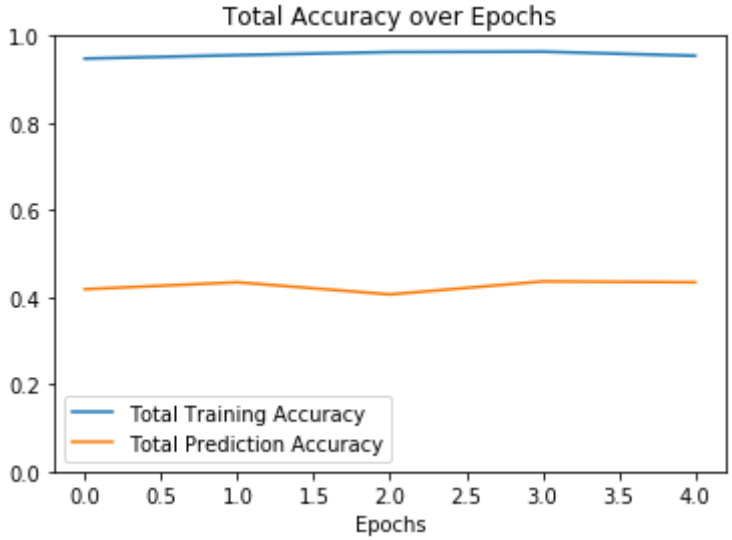
```
plt.plot(model_4_training.history['loss'])
plt.xlabel('Epochs')
plt.ylim([0, 7])
plt.title('Total Loss over Epochs')
plt.plot(model_4_training.history['val_loss'])
plt.legend(['Total Training loss', 'Total Prediction loss'])
```

```
<matplotlib.legend.Legend at 0x7ff6415e7f28>
```




```
plt.plot(model_4_training.history['Classification_acc'])
plt.xlabel('Epochs')
plt.ylim([0, 1])
plt.title('Total Accuracy over Epochs')
plt.plot(model_4_training.history['val_Classification_acc'])
plt.legend(['Total Training Accuracy', 'Total Prediction Accuracy'])
```

```
<matplotlib.legend.Legend at 0x7ff64156c668>
```




```
model_4_prediction = pd.DataFrame(model_4.predict(X_test_scaled)[1])
model_4_prediction.columns = ['draw', 'lost', 'won']
```

```
model_4_prediction.head()
```

	draw	lost	won
0	0.041764	1.195234e-02	0.001040
1	0.047456	9.993953e-01	0.000000
2	0.038507	6.794930e-06	0.811873
3	0.041174	3.246784e-03	0.004470
4	0.037337	4.172325e-07	0.990253

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix([np.argmax(x) for x in y_class_test.values], [np.argmax(x) for x in model_4
cm
```

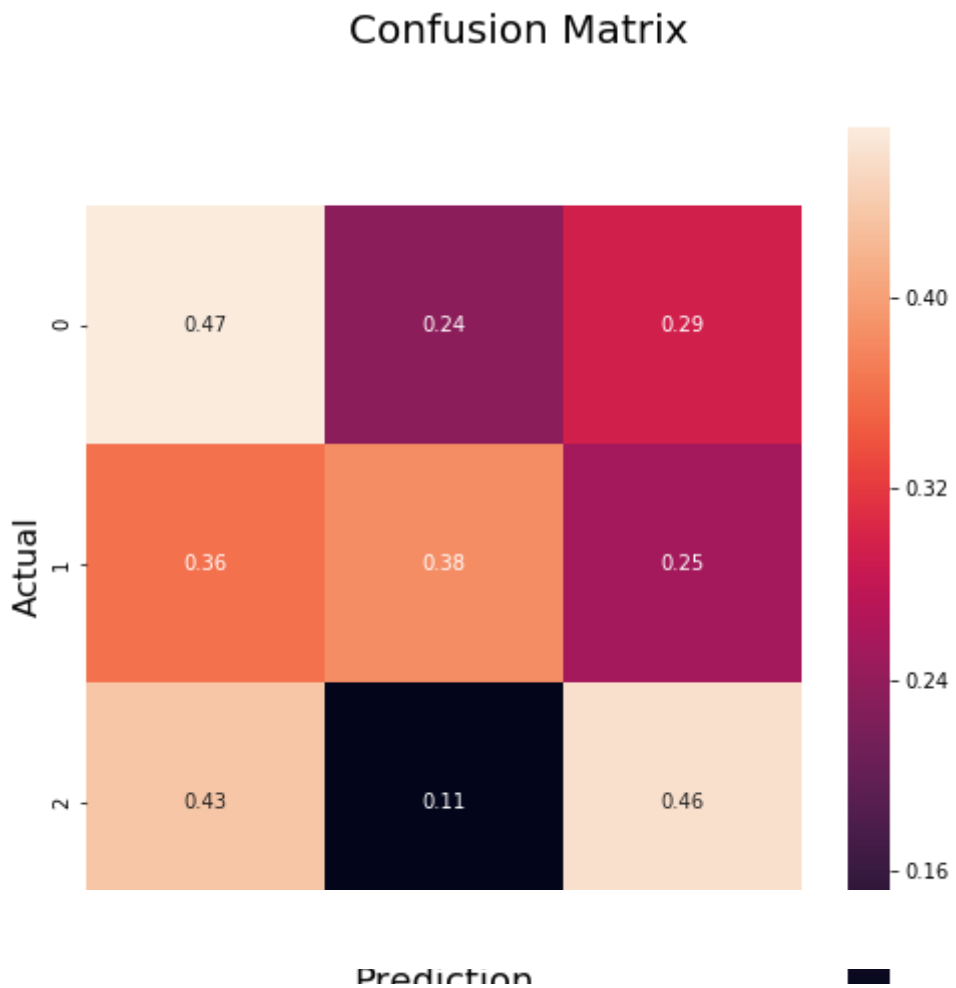
```
array([[56, 28, 35],
       [67, 71, 47],
       [86, 21, 91]])
```

```
from sklearn.metrics import confusion_matrix
fig, ax = plt.subplots(1, 1, figsize=(8,8))
cm = confusion_matrix([np.argmax(x) for x in y_class_test.values], [np.argmax(x) for x in model_4
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
sns.heatmap(cm, ax = ax, annot=True, square=True)
fig.suptitle('Confusion Matrix', fontsize=20)
plt.xlabel('Prediction', fontsize=18)
plt.ylabel('Actual', fontsize=16)
plt.show()


#### michi code
def pretty_confusion(target, prediction):
    """Prettify the on-board confusion matrix of sklearn."""

    cmc = ["Condition positive", "Condition negative"]
    cmi = ["Predicted condition positive", "Predicted condition negative"]

    matrix = confusion_matrix(target, prediction)
    cm = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
    return pd.DataFrame(cm , columns=cmc, index=cmi)
```




```
cm
```

```
y_class_test
```

```
y_cat.head()
```