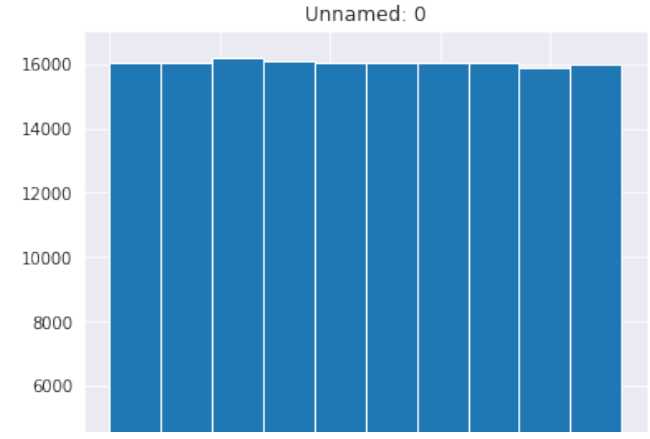
```
# load required packages
import numpy as np
import pandas as pd
pd.set_option('display.max_columns',100)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('darkgrid')
import tensorflow as tf
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelBinarizer
from sklearn.utils import class_weight
from sklearn.metrics import accuracy_score
```

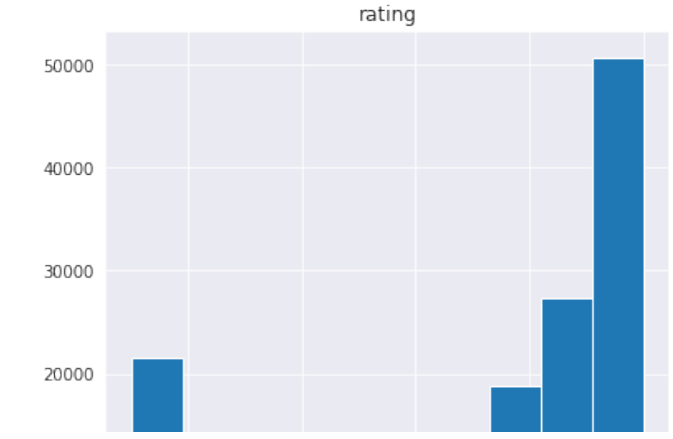
```
# load data
train_df = pd.read_csv('/content/drive/MyDrive/drugsComTrain_raw.tsv', sep = '
test_df = pd.read_csv('/content/drive/MyDrive/drugsComTest_raw.tsv', sep = '\t
```

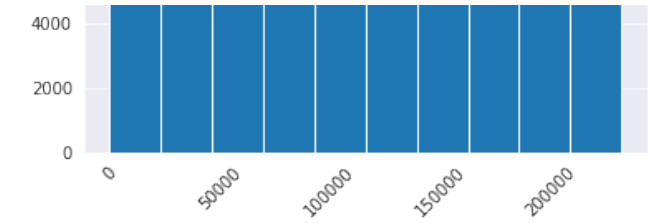
```
# remove na values from DataFrame
train_df = train_df.dropna()
test_df = test_df.dropna()
```

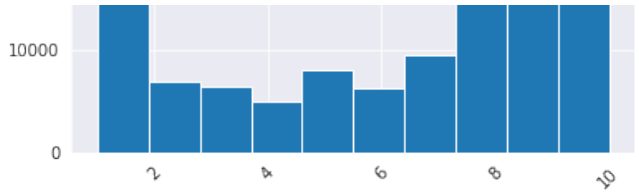
```
# exploration
def explore(df):
    print("Shape: ", df.shape, "/n")
    print(df.dtypes, "/n")
    print(df.head(), "/n")
    # numeric data statistics
    print(df.describe())
    df.hist(figsize=(14,14), xrot=45)
    plt.show()
    # categorical data statistics
    print(df.describe(include = 'object'))
    for column in df.select_dtypes(include = 'object'):
        if df[column].nunique() < 10:
            sns.countplot(y = column, data = df)
            plt.show()
            plt.savefig('{}_dist.png'.format(column))


explore(train_df)
```

```
Shape:  (160398, 7) /n
Unnamed: 0       int64
drugName        object
condition       object
review          object
rating         float64
date            object
usefulCount      int64
dtype: object /n
   Unnamed: 0                 drugName                      condition  \
0      206461                Valsartan  Left Ventricular Dysfunction
1       95260               Guanfacine                          ADHD
2       92703                   Lybrel              Birth Control
3      138000               Ortho Evra              Birth Control
4       35696  Buprenorphine / naloxone           Opiate Dependence

                                              review  rating  \
0  "It has no side effect, I take it in combinati...     9.0
1  "My son is halfway through his fourth week of ...     8.0
2  "I used to take another oral contraceptive, wh...     5.0
3  "This is my first time using any form of birth...     8.0
4  "Suboxone has completely turned my life around...     9.0

                date  usefulCount
0       May 20, 2012           27
1      April 27, 2010          192
2  December 14, 2009           17
3   November 3, 2015           10
4  November 27, 2016           37   /n
          Unnamed: 0         rating    usefulCount
count  160398.000000  160398.000000  160398.000000
mean   115957.147309       6.995923      28.097613
std     67015.899492       3.272303      36.457876
min         2.000000       1.000000       0.000000
25%     58065.250000       5.000000       6.000000
50%    115845.500000       8.000000      16.000000
75%    173826.750000      10.000000      36.000000
max    232291.000000      10.000000    1291.000000
```










```
# obtain counts for conditions
condition_counts = train_df['condition'].value_counts()
print(condition_counts[condition_counts > 1000])
```

```
Birth Control                 28788
Depression                     9069
Pain                           6145
Anxiety                        5904
Acne                           5588
Bipolar Disorde                4224
Insomnia                       3673
Weight Loss                    3609
Obesity                        3568
ADHD                           3383
Diabetes, Type 2               2554
Emergency Contraception        2463
High Blood Pressure            2321
Vaginal Yeast Infection        2274
Abnormal Uterine Bleeding      2096
Bowel Preparation              1859
ibromyalgia                    1791
Smoking Cessation              1780
Migraine                       1694
Anxiety and Stress             1663
Major Depressive Disorde       1607
Constipation                   1595
Panic Disorde                  1463
Chronic Pain                   1455
Migraine Prevention            1413
Urinary Tract Infection        1316
Muscle Spasm                   1244
Osteoarthritis                 1239
Generalized Anxiety Disorde    1164
Erectile Dysfunction           1086
Opiate Dependence              1079
Irritable Bowel Syndrome       1014
Name: condition, dtype: int64
```

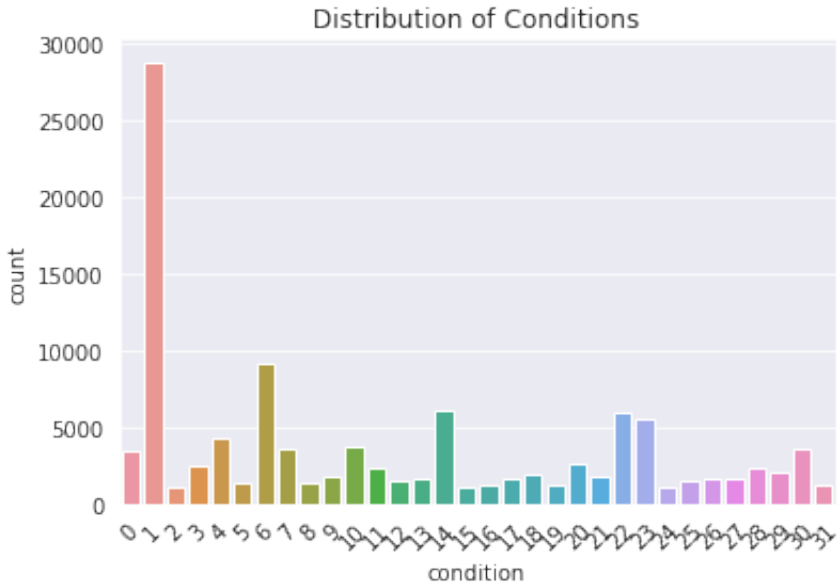
```
# reduce the number of classes to anything with more than 1000 reviews
condition_counts_1000 = condition_counts[condition_counts > 1000]

print(len(condition_counts_1000))

counts_1000_list = list()
for idx, name in enumerate(condition_counts_1000.index.tolist()):
  counts_1000_list.append(name)

train_df = train_df[train_df['condition'].isin(counts_1000_list)]
test_df = test_df[test_df['condition'].isin(counts_1000_list)]

print(train_df['review'].shape)
```

```
32
(110121,)
```

```
print(counts_1000_list)
```

```
['Birth Control', 'Depression', 'Pain', 'Anxiety', 'Acne', 'Bipolar Disorde',
```

```
# visualiztions
fig, ax = plt.subplots(1, 1)
ax = sns.countplot(x = 'condition', data = train_df)
ax.set_title('Distribution of Conditions')
ax.set_xticklabels(ax.get_xticks(), rotation = 45)
fig.show()
fig.savefig('condition_dist.png')
```




```
# set data and labels
x_train = train_df['review']
x_test = test_df['review']
y_train = train_df['condition']
y_test = test_df['condition']
```

```
# labels to one hot endoced
def prepare_targets(y_train, y_test):
  one_hot = LabelBinarizer()
  one_hot.fit(y_train)
  y_train = np.argmax(one_hot.transform(y_train), axis = 1)
  y_test_one_hot = one_hot.transform(y_test)
  y_test = np.argmax(one_hot.transform(y_test), axis = 1)
  y_test_rev = one_hot.inverse_transform(y_test_one_hot)
  label_list = one_hot.classes_
  return y_train, y_test, y_test_one_hot, y_test_rev, label_list

y_train, y_test, y_test_one_hot, y_test_rev, label_list = prepare_targets(y_tr
```

```
# create weights for classes
class_weights = class_weight.compute_class_weight('balanced', np.unique(y_trai
weights = dict(enumerate(class_weights))
print(weights)
```

```
{0: 1.0172276825302986, 1: 1.6418326574427482, 2: 0.6158341535433071, 3: 0.582
```

```
# baseline model; all predictions birth control
count_bc = condition_counts['Birth Control']
base_acc = count_bc/len(y_train)
print(base_acc)
```

```
0.26142152722913886
```

```
# vectorize text
vocab_size = 1500
review_len_max = 200
encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(
    max_tokens = vocab_size,
    output_sequence_length = review_len_max)

# develop vocabulary
encoder.adapt(x_train.values)

# vectorize text
x_train = encoder(x_train)
x_test = encoder(x_test)
```

```
print('Training input shape: ', x_train.shape)
print(len(x_train))
print(len(x_train[0]))
print(x_train[0].shape)
print('Test input shape: ', x_test.shape)
print(len(x_test))
print(len(x_test[0]))
print(x_test[0].shape)
```

```
Training input shape:  (110121, 200)
110121
200
(200,)
Test input shape:  (36827, 200)
36827
200
(200,)
```

```
print(y_train.shape)
```

```
(110121,)
```

```
# classifiying with tf.keras RNN
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim = vocab_size + 1, output_dim = 16),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(units = 16, dropout = 0
    tf.keras.layers.Dense(32, activation = 'softmax')
]
)

# compile model
model.compile(loss = tf.keras.losses.SparseCategoricalCrossentropy(),
              optimizer = 'Adam',
              metrics = ['acc'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, None, 16)          24016     
_________________________________________________________________
bidirectional (Bidirectional (None, 32)                4224      
_________________________________________________________________
dense (Dense)                (None, 32)                1056      
=================================================================
Total params: 29,296
Trainable params: 29,296
Non-trainable params: 0
_________________________________________________________________
```

```
# test model
history = model.fit(x = x_train, y = y_train,
        epochs = 50,
        batch_size = 50,
        validation_split = 0.2,
        class_weight = weights,
    )
```

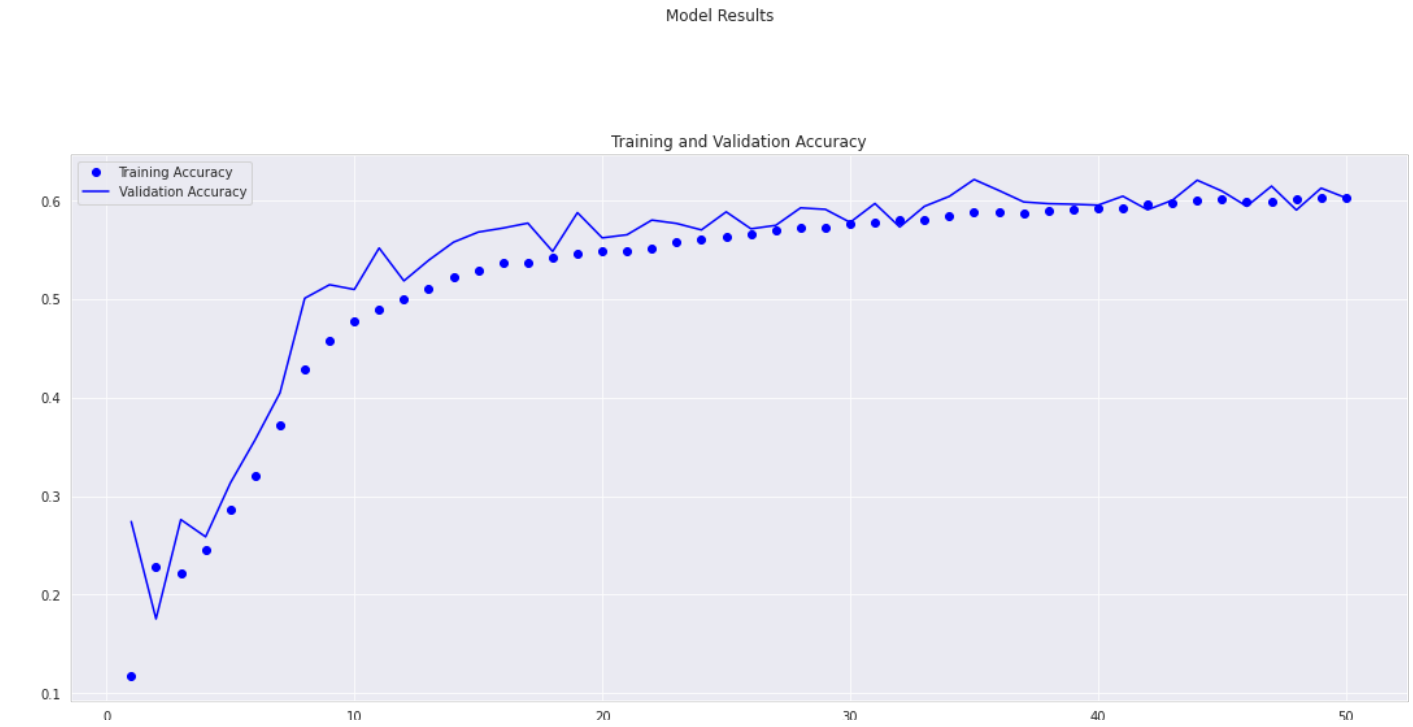
```
Epoch 1/50
1762/1762 [==============================] - 68s 20ms/step - loss: 3.2001 - ac
Epoch 2/50
1762/1762 [==============================] - 34s 19ms/step - loss: 2.6470 - ac
Epoch 3/50
1762/1762 [==============================] - 34s 19ms/step - loss: 2.3106 - ac
Epoch 4/50
1762/1762 [==============================] - 34s 20ms/step - loss: 2.0920 - ac
Epoch 5/50
```

```
Epoch 5/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.9465 - ac
Epoch 6/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.8344 - ac
Epoch 7/50
1762/1762 [==============================] - 33s 19ms/step - loss: 1.7504 - ac
Epoch 8/50
1762/1762 [==============================] - 30s 17ms/step - loss: 1.6362 - ac
Epoch 9/50
1762/1762 [==============================] - 33s 19ms/step - loss: 1.5683 - ac
Epoch 10/50
1762/1762 [==============================] - 35s 20ms/step - loss: 1.4990 - ac
Epoch 11/50
1762/1762 [==============================] - 33s 19ms/step - loss: 1.4578 - ac
Epoch 12/50
1762/1762 [==============================] - 31s 17ms/step - loss: 1.4281 - ac
Epoch 13/50
1762/1762 [==============================] - 31s 18ms/step - loss: 1.3844 - ac
Epoch 14/50
1762/1762 [==============================] - 35s 20ms/step - loss: 1.3514 - ac
Epoch 15/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.3351 - ac
Epoch 16/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.3112 - ac
Epoch 17/50
1762/1762 [==============================] - 32s 18ms/step - loss: 1.2995 - ac
Epoch 18/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.2925 - ac
Epoch 19/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.2697 - ac
Epoch 20/50
1762/1762 [==============================] - 35s 20ms/step - loss: 1.2583 - ac
Epoch 21/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.2514 - ac
Epoch 22/50
1762/1762 [==============================] - 35s 20ms/step - loss: 1.2458 - ac
Epoch 23/50
1762/1762 [==============================] - 35s 20ms/step - loss: 1.2107 - ac
Epoch 24/50
1762/1762 [==============================] - 33s 19ms/step - loss: 1.1952 - ac
Epoch 25/50
1762/1762 [==============================] - 31s 17ms/step - loss: 1.2085 - ac
Epoch 26/50
1762/1762 [==============================] - 31s 18ms/step - loss: 1.1817 - ac
Epoch 27/50
1762/1762 [==============================] - 32s 18ms/step - loss: 1.1740 - ac
Epoch 28/50
1762/1762 [==============================] - 33s 19ms/step - loss: 1.1762 - ac
Epoch 29/50
1762/1762 [==============================] - 32s 18ms/step - loss: 1.1684 - ac
Epoch 30/50
```

```
1762/1762 [==============================] - 34s 19ms/step - loss: 1.1671 - a
```

```
# save model
model.save('drug_review_rnn_50.h5')
```

```
# model results
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

fig, (ax0, ax1) = plt.subplots(2, 1, figsize = (15,15))
fig.suptitle('Model Results')
ax0.plot(epochs, acc, 'bo', label = 'Training Accuracy')
ax0.plot(epochs, val_acc, 'b', label = 'Validation Accuracy')
ax0.set_title('Training and Validation Accuracy')
ax0.legend()

ax1.plot(epochs, loss, 'bo', label = 'Training Loss')
ax1.plot(epochs, val_loss, 'b', label = 'Validation Loss')
ax1.set_title('Training and Validation Loss')
ax1.legend()
fig.tight_layout(rect=[0, 0.03, 1, 0.9])

fig.savefig('model_results.png')
```

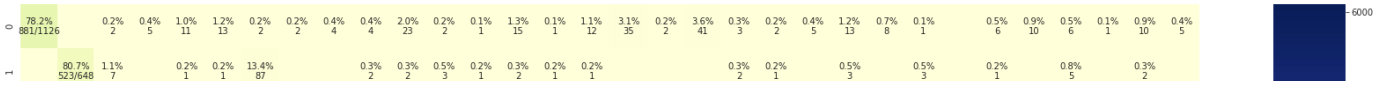


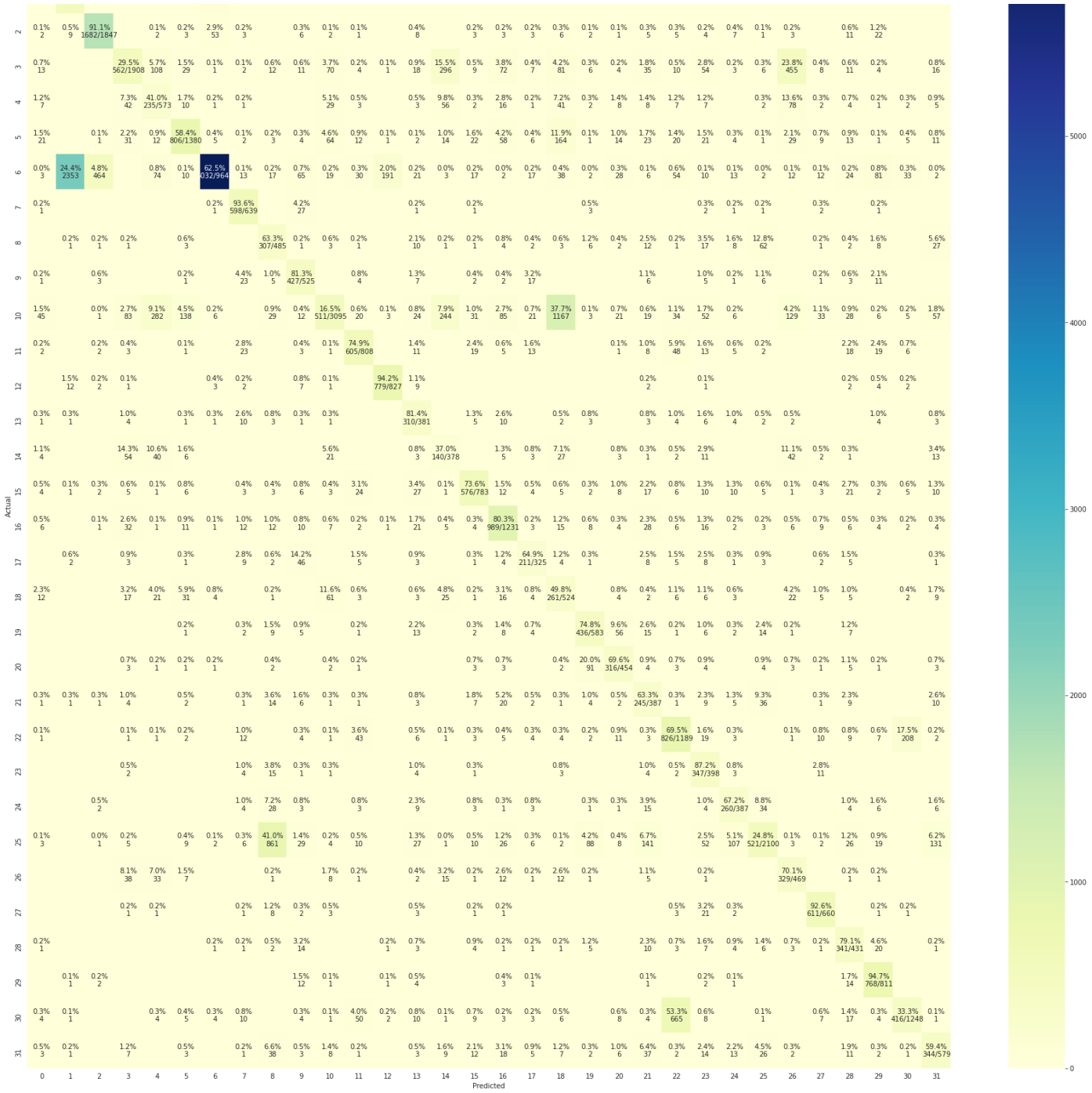

```
# model performance
predictions = model.predict(x_test)
```

```
print(predictions[0])
print(y_test[0])
print(y_test_one_hot[0])
print(y_test_rev[0])
```

```
 [4.1943714e-03 5.3141492e-05 1.1367062e-04 9.5583566e-02 2.2228974e-01
  1.9638129e-02 1.8428185e-03 7.2395586e-08 2.5014702e-05 5.0075510e-06
  2.4951330e-01 7.3006345e-06 3.9542447e-06 1.4613361e-06 1.5919979e-01
  1.8198247e-04 1.3924406e-03 4.0000530e-05 2.0378804e-01 6.9832919e-07
  7.5372423e-05 1.2797008e-06 2.0408104e-04 6.0204049e-05 4.9668624e-06
  5.3810249e-06 4.0609650e-02 6.3067378e-04 3.0860517e-06 4.8659804e-06
  1.6989521e-04 3.5602206e-04]
 10
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 Depression
```

```
# build confusion matrix
def plot_cm(y_true, y_pred, figsize = (30, 30)):
    cm = confusion_matrix(y_true, y_pred, labels = np.unique(y_true))
    cm_sum = np.sum(cm, axis = 1, keepdims = True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index = np.unique(y_true), columns = np.unique(y_tru
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    fig, ax = plt.subplots(figsize = figsize)
    sns.heatmap(cm, cmap= "YlGnBu", annot = annot, fmt = '', ax = ax)
    fig.savefig('confusion_matrix.png')

plot_cm(y_test_one_hot.argmax(axis = 1), predictions.argmax(axis = 1))
```




Actual
Predicted
2 0.1% 2 0.5% 9 91.1% 1682/1847 0.1% 2 0.2% 3 2.9% 53 0.2% 3 0.3% 6 0.1% 2 0.1% 1 0.4% 8 0.2% 3 0.2% 3 0.2% 3 0.3% 6 0.1% 2 0.1% 1 0.3% 5 0.3% 5 0.2% 4 0.4% 7 0.1% 1 0.2% 3 0.6% 11 1.2% 22
3 0.7% 13 29.5% 562/1908 5.7% 108 1.5% 29 0.1% 1 0.1% 2 0.6% 12 0.6% 11 3.7% 70 0.2% 4 0.1% 1 0.9% 18 15.5% 296 0.5% 9 3.8% 72 0.4% 7 4.2% 81 0.3% 6 0.2% 4 1.8% 35 0.5% 10 2.8% 54 0.2% 3 0.3% 6 23.8% 455 0.4% 8 0.6% 11 0.2% 4 0.8% 16
4 1.2% 7 7.3% 42 41.0% 235/573 1.7% 10 0.2% 1 0.2% 1 5.1% 29 0.5% 3 0.5% 3 9.8% 56 0.3% 2 2.8% 16 0.2% 1 7.2% 41 0.3% 2 1.4% 8 1.4% 8 1.2% 7 1.2% 7 0.3% 2 13.6% 78 0.3% 2 0.7% 4 0.2% 1 0.3% 2 0.9% 5
5 1.5% 21 0.1% 1 2.2% 31 0.9% 12 58.4% 806/1380 0.4% 5 0.1% 2 0.2% 3 0.3% 4 4.6% 64 0.9% 12 0.1% 1 0.1% 2 1.0% 14 1.6% 22 4.2% 58 0.4% 6 11.9% 164 0.1% 1 1.0% 14 1.7% 23 1.4% 20 1.5% 21 0.3% 4 0.1% 1 2.1% 29 0.7% 9 0.9% 13 0.1% 1 0.4% 5 0.8% 11
6 0.0% 3 24.4% 2353 4.8% 464 0.8% 74 0.1% 10 62.5% 032/964 0.1% 13 0.2% 17 0.7% 65 0.2% 19 0.3% 30 2.0% 191 0.2% 21 0.0% 3 0.2% 17 0.0% 2 0.2% 17 0.4% 38 0.0% 2 0.3% 28 0.1% 6 0.6% 54 0.1% 10 0.1% 13 0.0% 2 0.1% 12 0.1% 12 0.2% 24 0.8% 81 0.3% 33 0.0% 2
7 0.2% 1 0.2% 1 93.6% 598/639 4.2% 27 0.2% 1 0.2% 1 0.5% 3 0.3% 2 0.2% 1 0.2% 1 0.3% 2 0.2% 1
8 0.2% 1 0.2% 1 0.2% 1 0.6% 3 63.3% 307/485 0.2% 1 0.6% 3 0.2% 1 2.1% 10 0.2% 1 0.2% 1 0.8% 4 0.4% 2 0.6% 3 1.2% 6 0.4% 2 2.5% 12 0.2% 1 3.5% 17 1.6% 8 12.8% 62 0.2% 1 0.4% 2 1.6% 8 5.6% 27
9 0.2% 1 0.6% 3 0.2% 1 4.4% 23 1.0% 5 81.3% 427/525 0.8% 4 1.3% 7 0.4% 2 0.4% 2 3.2% 17 1.1% 6 1.0% 5 0.2% 1 1.1% 6 0.2% 1 0.6% 3 2.1% 11
10 1.5% 45 0.0% 1 2.7% 83 9.1% 282 4.5% 138 0.2% 6 0.9% 29 0.4% 12 16.5% 511/3095 0.6% 20 0.1% 3 0.8% 24 7.9% 244 1.0% 31 2.7% 85 0.7% 21 37.7% 1167 0.1% 3 0.7% 21 0.6% 19 1.1% 34 1.7% 52 0.2% 6 4.2% 129 1.1% 33 0.9% 28 0.2% 6 0.2% 5 1.8% 57
11 0.2% 2 0.2% 2 0.4% 3 0.1% 1 2.8% 23 0.4% 3 0.1% 1 74.9% 605/808 1.4% 11 2.4% 19 0.6% 5 1.6% 13 0.1% 1 1.0% 8 5.9% 48 1.6% 13 0.6% 5 0.2% 2 2.2% 18 2.4% 19 0.7% 6
12 1.5% 12 0.2% 2 0.1% 1 0.4% 3 0.2% 2 0.8% 7 0.1% 1 94.2% 779/827 1.1% 9 0.2% 2 0.1% 1 0.2% 2 0.5% 4 0.2% 2
13 0.3% 1 0.3% 1 1.0% 4 0.3% 1 0.3% 1 2.6% 10 0.8% 3 0.3% 1 0.3% 1 81.4% 310/381 1.3% 5 2.6% 10 0.5% 2 0.8% 3 0.8% 3 1.0% 4 1.6% 6 1.0% 4 0.5% 2 0.5% 2 1.0% 4 0.8% 3
14 1.1% 4 14.3% 54 10.6% 40 1.6% 6 0.8% 3 37.0% 140/378 1.3% 5 0.8% 3 7.1% 27 0.8% 3 0.3% 1 0.5% 2 2.9% 11 11.1% 42 0.5% 2 0.3% 1 3.4% 13
15 0.5% 4 0.1% 1 0.3% 2 0.6% 5 0.1% 1 0.8% 6 0.4% 3 0.4% 3 0.8% 6 0.4% 3 3.1% 24 3.4% 27 0.1% 1 73.6% 576/783 1.5% 12 0.5% 4 0.6% 5 0.3% 2 1.0% 8 2.2% 17 0.8% 6 1.3% 10 1.3% 10 0.6% 5 0.1% 1 0.4% 3 2.7% 21 0.3% 2 0.6% 5 1.3% 10
16 0.5% 6 0.1% 1 2.6% 32 0.1% 1 0.9% 11 0.1% 1 1.0% 12 1.0% 12 0.8% 10 0.6% 7 0.2% 2 0.1% 1 1.7% 21 0.4% 5 0.3% 4 80.3% 989/1231 0.2% 3 1.2% 15 0.6% 8 0.3% 4 2.3% 28 0.5% 6 1.3% 16 0.2% 2 0.2% 3 0.5% 6 0.7% 9 0.5% 6 0.3% 4 0.2% 2 0.3% 4
17 0.6% 2 0.9% 3 0.3% 1 2.8% 9 0.6% 2 14.2% 46 1.5% 5 0.9% 3 0.3% 1 1.2% 4 64.9% 211/325 1.2% 4 0.3% 1 2.5% 8 1.5% 5 2.5% 8 0.3% 1 0.9% 3 0.6% 2 1.5% 5 0.3% 1
18 2.3% 12 3.2% 17 4.0% 21 5.9% 31 0.8% 4 0.2% 1 11.6% 61 0.6% 3 0.6% 3 4.8% 25 0.2% 1 3.1% 16 0.8% 4 49.8% 261/524 0.8% 4 0.4% 2 1.1% 6 1.1% 6 0.6% 3 4.2% 22 1.0% 5 1.0% 5 0.4% 2 1.7% 9
19 0.2% 1 0.3% 2 1.5% 9 0.9% 5 0.2% 1 2.2% 13 0.3% 2 1.4% 8 0.7% 4 74.8% 436/583 9.6% 56 2.6% 15 0.2% 1 1.0% 6 0.3% 2 2.4% 14 0.2% 1 1.2% 7
20 0.7% 3 0.2% 1 0.2% 1 0.2% 1 0.4% 2 0.4% 2 0.2% 1 0.7% 3 0.7% 3 0.4% 2 20.0% 91 69.6% 316/454 0.9% 4 0.7% 3 0.9% 4 0.9% 4 0.7% 3 0.2% 1 1.1% 5 0.2% 1 0.7% 3
21 0.3% 1 0.3% 1 0.3% 1 1.0% 4 0.5% 2 0.3% 1 3.6% 14 1.6% 6 0.3% 1 0.3% 1 0.8% 3 1.8% 7 5.2% 20 0.5% 2 0.3% 1 1.0% 4 0.5% 2 63.3% 245/387 0.3% 1 2.3% 9 1.3% 5 9.3% 36 0.3% 1 2.3% 9 2.6% 10
22 0.1% 1 0.1% 1 0.1% 1 0.2% 2 1.0% 12 0.3% 4 0.1% 1 3.6% 43 0.5% 6 0.1% 1 0.3% 3 0.4% 5 0.3% 4 0.3% 4 0.2% 2 0.9% 11 0.3% 3 69.5% 826/1189 1.6% 19 0.3% 3 0.1% 1 0.8% 10 0.8% 9 0.6% 7 17.5% 208 0.2% 2
23 0.5% 2 1.0% 4 3.8% 15 0.3% 1 0.3% 1 1.0% 4 0.3% 1 0.8% 3 1.0% 4 0.5% 2 87.2% 347/398 0.8% 3 2.8% 11
24 0.5% 2 1.0% 4 7.2% 28 0.8% 3 0.8% 3 2.3% 9 0.8% 3 0.3% 1 0.8% 3 0.3% 1 0.3% 1 3.9% 15 1.0% 4 67.2% 260/387 8.8% 34 1.0% 4 1.6% 6 1.6% 6
25 0.1% 3 0.0% 1 0.2% 5 0.4% 9 0.1% 2 0.3% 6 41.0% 861 1.4% 29 0.2% 4 0.5% 10 1.3% 27 0.0% 1 0.5% 10 1.2% 26 0.3% 6 0.1% 2 4.2% 88 0.4% 8 6.7% 141 2.5% 52 5.1% 107 24.8% 521/2100 0.1% 3 0.1% 2 1.2% 26 0.9% 19 6.2% 131
26 8.1% 38 7.0% 33 1.5% 7 0.2% 1 1.7% 8 0.2% 1 0.4% 2 3.2% 15 0.2% 1 2.6% 12 0.2% 1 2.6% 12 0.2% 1 1.1% 5 0.2% 1 70.1% 329/469 0.2% 1 0.2% 1
27 0.2% 1 0.2% 1 0.2% 1 1.2% 8 0.3% 2 0.5% 3 0.5% 3 0.2% 1 0.2% 1 0.5% 3 0.3% 2 92.6% 611/660 0.2% 1 0.2% 1
28 0.2% 1 0.2% 1 0.5% 2 3.2% 14 0.2% 1 0.7% 3 0.9% 4 0.2% 1 0.2% 1 0.2% 1 1.2% 5 2.3% 10 0.7% 3 1.6% 7 0.9% 4 1.4% 6 0.7% 3 0.2% 1 79.1% 341/431 4.6% 20 0.2% 1
29 0.1% 1 0.2% 2 1.5% 12 0.1% 1 0.1% 1 0.5% 4 0.4% 3 0.1% 1 0.1% 1 0.2% 2 0.1% 1 1.7% 14 94.7% 768/811
30 0.3% 4 0.1% 1 0.3% 4 0.4% 5 0.3% 4 0.8% 10 0.3% 4 0.1% 1 4.0% 50 0.2% 2 0.8% 10 0.1% 1 0.7% 9 0.2% 3 0.2% 3 0.5% 6 0.6% 8 0.3% 4 53.3% 665 0.6% 8 0.1% 1 0.6% 7 1.4% 17 0.3% 4 33.3% 416/1248 0.1% 1
31 0.5% 3 0.2% 1 1.2% 7 0.5% 3 0.2% 1 6.6% 38 0.5% 3 1.4% 8 0.2% 1 0.5% 3 1.6% 9 2.1% 12 3.1% 18 0.9% 5 1.2% 7 0.3% 2 1.0% 6 6.4% 37 0.3% 2 2.4% 14 2.2% 13 4.5% 26 0.3% 2 1.9% 11 0.3% 2 0.2% 1 59.4% 344/579
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
5000
4000
3000
2000
1000
0

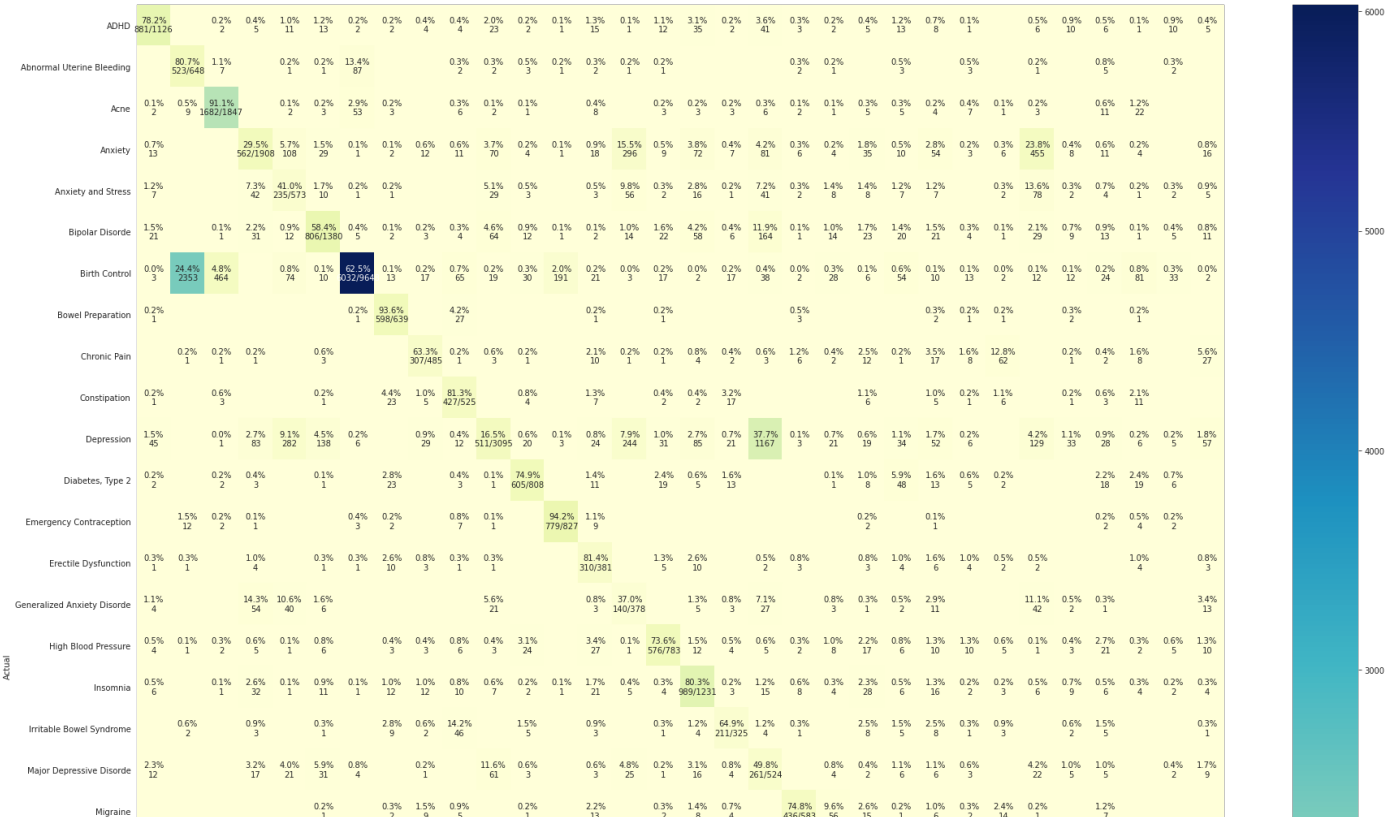
```
# load saved model
new_model = tf.keras.models.load_model('/content/drug_review_rnn_50.h5')

# make new predictions
new_predictions = new_model.predict(x_test)
```

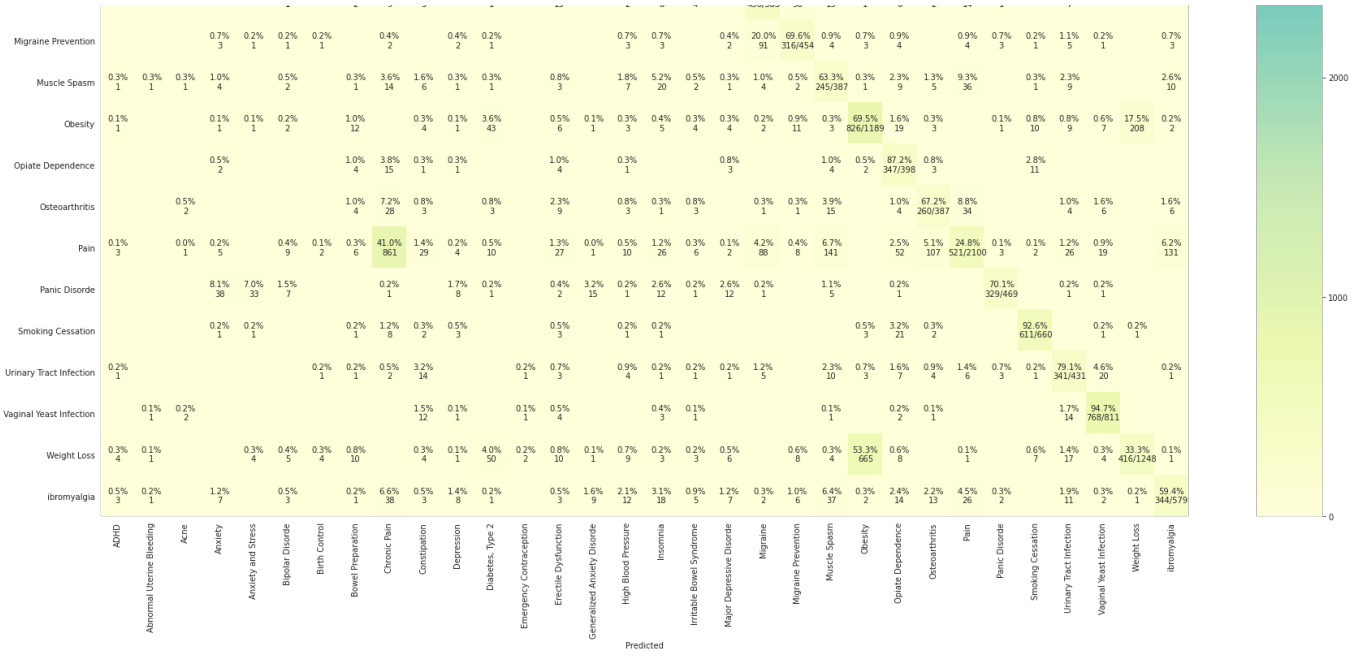
```
print(y_test[0])
print(y_test_one_hot[0])
print(y_test_rev[0])
```

```
10
[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Depression
```

```
# build confusion matrix
def plot_cm(y_true, y_pred, figsize = (30, 30)):
```

```
    cm = confusion_matrix(y_true, y_pred, labels = np.unique(y_true))
    cm_sum = np.sum(cm, axis = 1, keepdims = True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index = np.unique(y_true), columns = np.unique(y_tru
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    fig, ax = plt.subplots(figsize = figsize)
    sns.heatmap(cm, cmap= "YlGnBu", annot = annot, fmt = '', ax = ax, xticklab
    fig.savefig('confusion_matrix.png')

plot_cm(y_test_one_hot.argmax(axis = 1), new_predictions.argmax(axis = 1))
```




Migraine Prevention
Muscle Spasm
Obesity
Opiate Dependence
Osteoarthritis
Pain
Panic Disorde
Smoking Cessation
Urinary Tract Infection
Vaginal Yeast Infection
Weight Loss
ibromyalgia
ADHD
Abnormal Uterine Bleeding
Acne
Anxiety
Anxiety and Stress
Bipolar Disorde
Birth Control
Bowel Preparation
Chronic Pain
Constipation
Depression
Diabetes, Type 2
Emergency Contraception
Erectile Dysfunction
Generalized Anxiety Disorde
High Blood Pressure
Insomnia
Irritable Bowel Syndrome
Major Depressive Disorde
Migraine
Migraine Prevention
Muscle Spasm
Obesity
Opiate Dependence
Osteoarthritis
Pain
Panic Disorde
Smoking Cessation
Urinary Tract Infection
Vaginal Yeast Infection
Weight Loss
ibromyalgia
Predicted
2000
1000
0

```
# model accuracy
accuracy_metric = accuracy_score(y_test, new_predictions.argmax(axis = 1))

print('Accuracy: ', accuracy_metric)
```

```
Accuracy:  0.6026828142395525
```