```
# 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
```

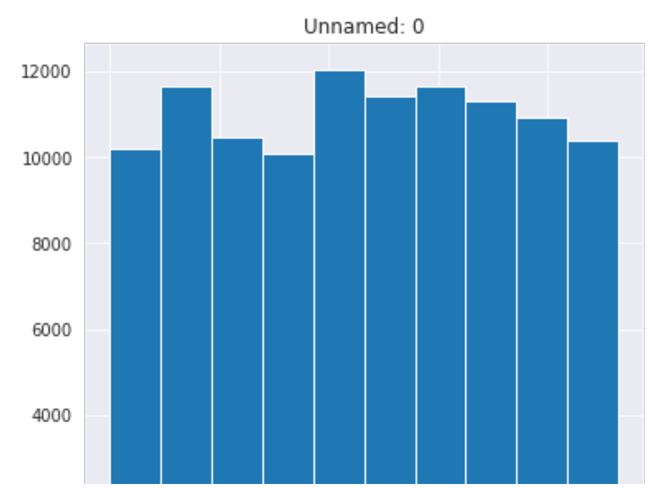
```
# load data
train_df = pd.read_csv('/content/drive/MyDrive/drugsComTrain_raw.tsv', sep = '\
test_df = pd.read_csv('/content/drive/MyDrive/drugsComTest_raw.tsv', sep = '\t'
```

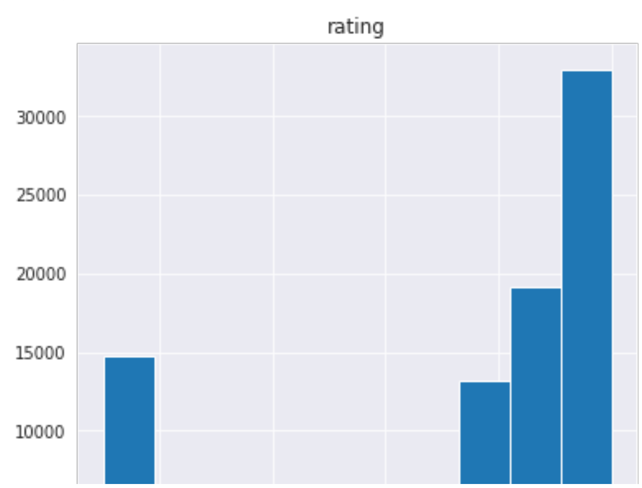
```
# remove na values from DataFrame
train_df = train_df.dropna()
test_df = test_df.dropna()
```

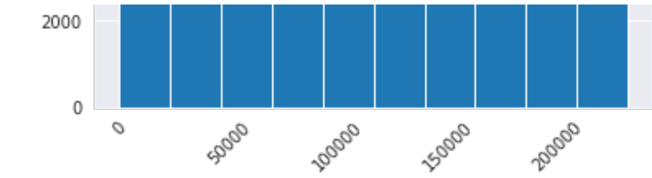
```
# exploration
def explore(df):
    print("Shape: ", df.shape, "/n")
    print(df.dtypes, "/n")
    print(df.head(), "/n")
    # numeric data statistics
    print(df.describe())
    df.hist(figsize=(14,14), xrot=45)
    plt.show()
    # categorical data statistics
    print(df.describe(include = 'object'))
    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)
```

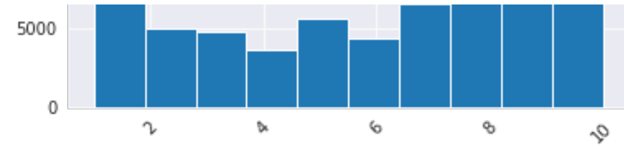
```
Shape:  (110121, 7) /n
```

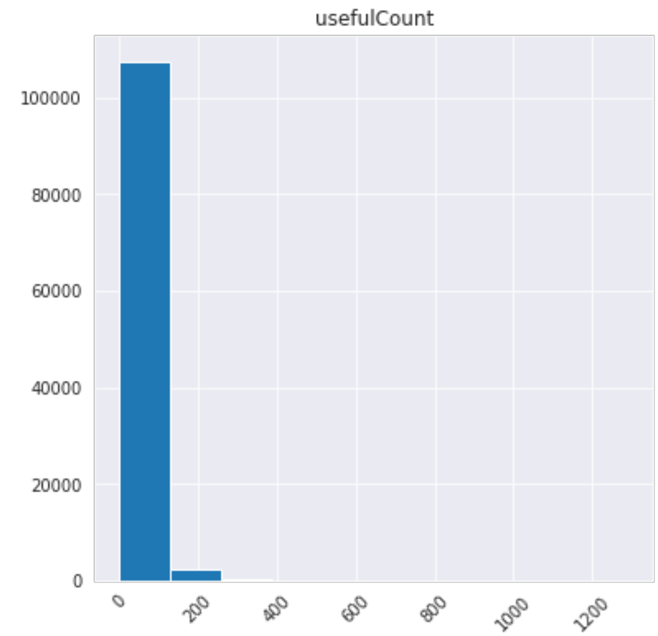
```
Unnamed: 0       int64
drugName        object
condition       object
review          object
rating         float64
date            object
usefulCount      int64
dtype: object /n
   Unnamed: 0                    drugName                  condition  \
1       95260                  Guanfacine                       ADHD
2       92703                      Lybrel              Birth Control
3      138000                  Ortho Evra              Birth Control
4       35696  Buprenorphine / naloxone          Opiate Dependence
6      165907              Levonorgestrel  Emergency Contraception

                                              review  rating  \
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
6  "He pulled out, but he cummed a bit in me. I t...     1.0

                date  usefulCount
1     April 27, 2010          192
2  December 14, 2009           17
3   November 3, 2015           10
4  November 27, 2016           37
6      March 7, 2017            5   /n
          Unnamed: 0         rating    usefulCount
count  110121.000000  110121.000000  110121.000000
mean   116603.495346       6.919924      28.008899
std     66249.766260       3.270902      38.251211
min         3.000000       1.000000       0.000000
25%     60483.000000       4.000000       6.000000
50%    117681.000000       8.000000      15.000000
75%    172110.000000      10.000000      36.000000
max    232289.000000      10.000000    1291.000000
```











	drugName	condition	review	date
count	110121	110121	110121	110121
unique	1324	32	75730	3576
top	Levonorgestrel	Birth Control	"Good"	March 1, 2016
freq	3626	28788	22	106

```
# 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
```

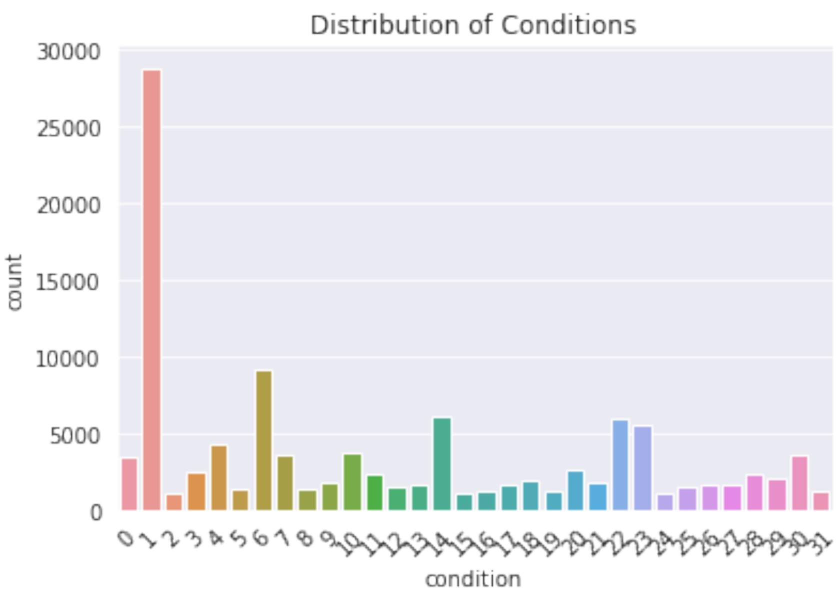
```
# reduce the number of classes to anything with more than 1000 reviews
condition_counts_1000 = condition_counts[condition_counts > 1000]

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)
  return y_train, y_test, y_test_one_hot, y_test_rev

y_train, y_test, y_test_one_hot, y_test_rev = prepare_targets(y_train, y_test)
```

```
# create weights for classes
class_weights = class_weight.compute_class_weight('balanced', np.unique(y_train

weights = dict(enumerate(class_weights))
print(weights)
```

```
72276825302986, 1: 1.6418326574427482, 2: 0.6158341535433071, 3: 0.5828728404
```

```
# 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 - a
Epoch 2/50
1762/1762 [==============================] - 34s 19ms/step - loss: 2.6470 - a
Epoch 3/50
1762/1762 [==============================] - 34s 19ms/step - loss: 2.3106 - a
Epoch 4/50
1762/1762 [==============================] - 34s 20ms/step - loss: 2.0920 - a
```

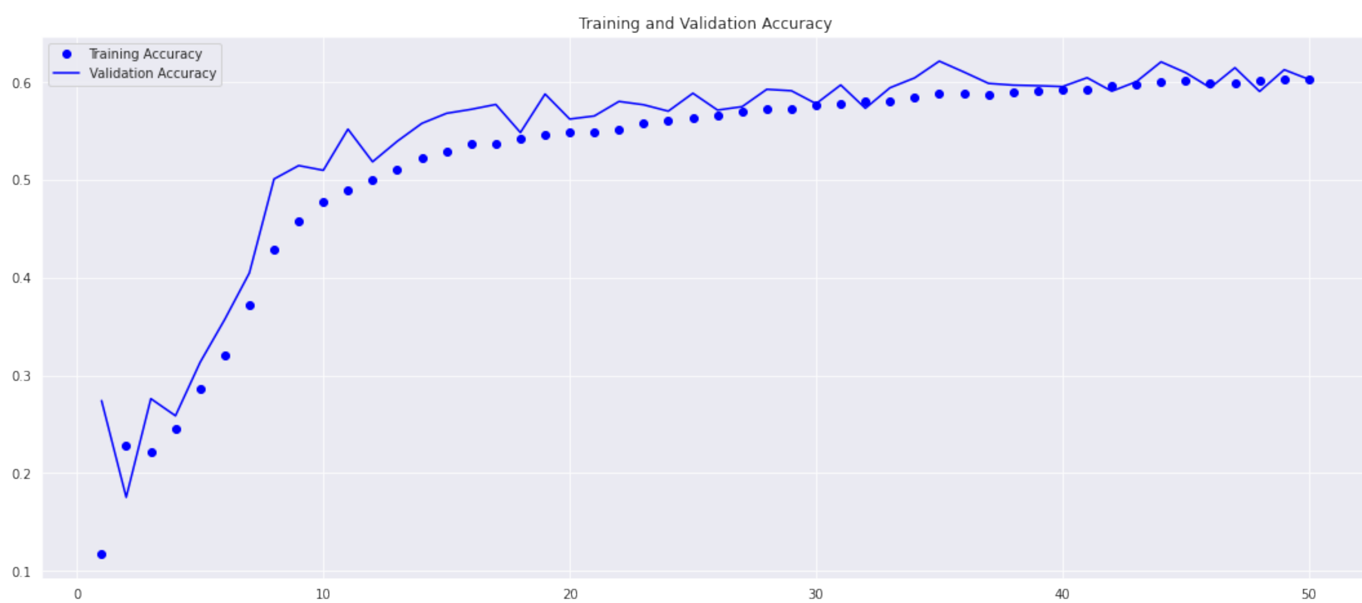
```
Epoch 5/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.9465 - a
Epoch 6/50
1762/1762 [==============================] - 34s 19ms/step - loss: 1.8344 - a
Epoch 7/50
1762/1762 [==============================] - 33s 19ms/step - loss: 1.7504 - a
Epoch 8/50
1762/1762 [==============================]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

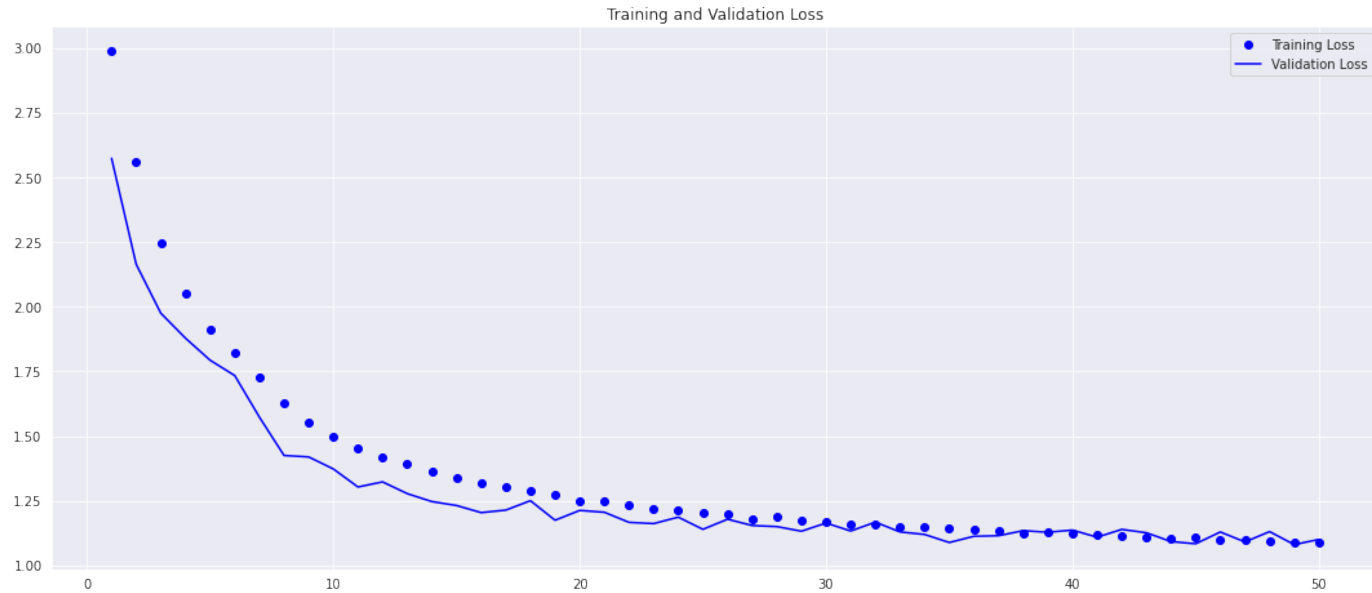
```
Epoch 50/50
1763/1763 [==============================] - 34s 19ms/step - loss: 1.1671 - a
```

```
# 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 Results



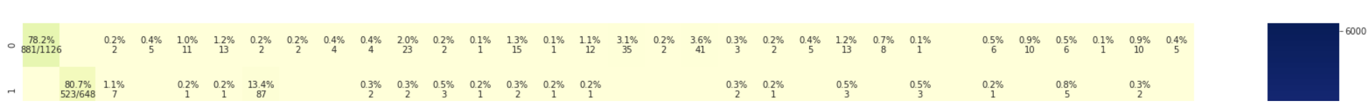


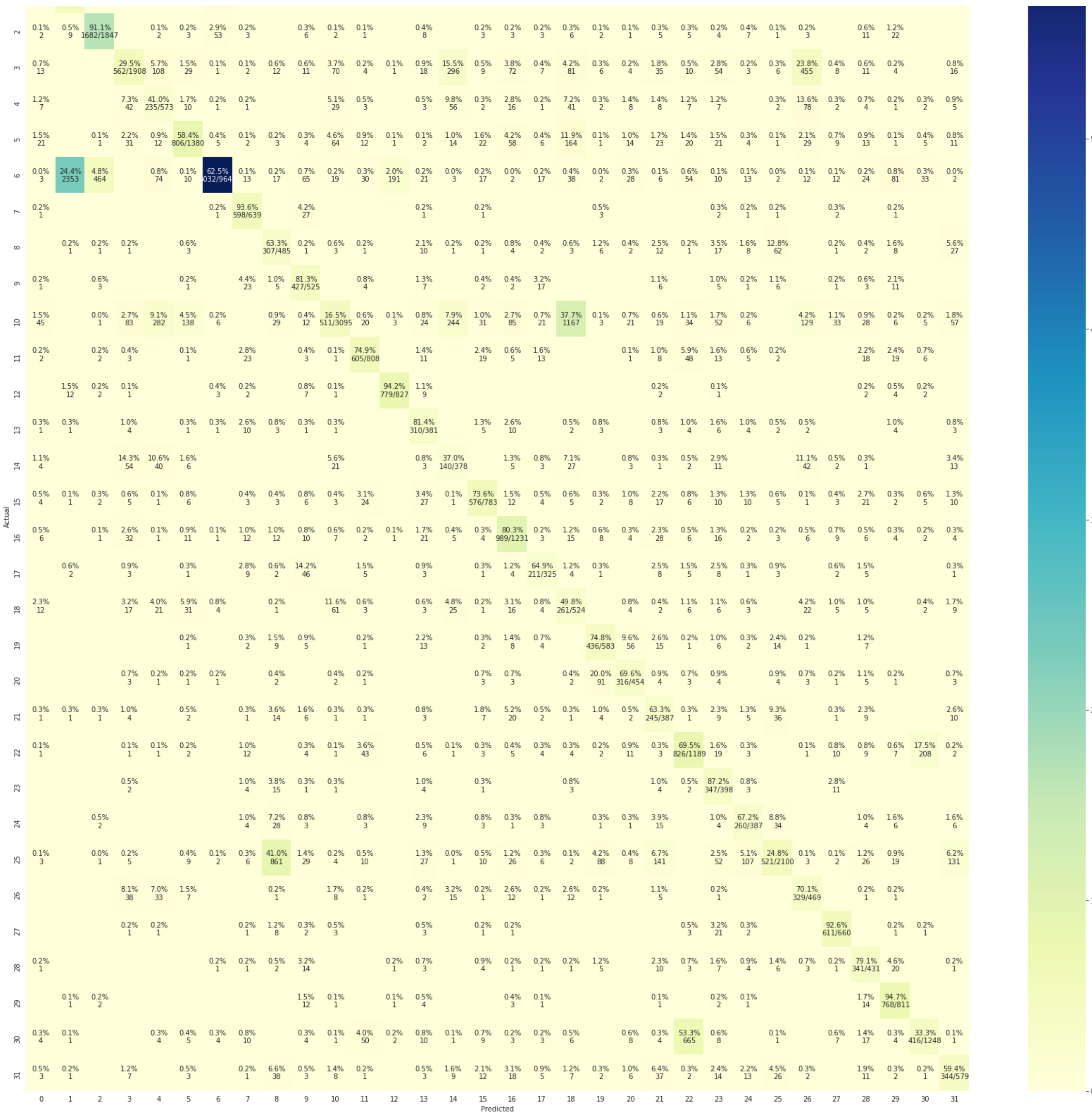

```
# model performance
predictions = model.predict(x_test)
```

```
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]
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_true
    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
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

```
1
```

10s completed at 9:36 PM