

```
1 # load required packages
2 import numpy as np
3 import pandas as pd
4 pd.set_option('display.max_columns',100)
5 import matplotlib.pyplot as plt
6 %matplotlib inline
7 import seaborn as sns
8 sns.set_style('darkgrid')
9 import tensorflow as tf
10 from sklearn.metrics import confusion_matrix
11 from sklearn.preprocessing import LabelBinarizer
12 from sklearn.utils import class_weight
13 from sklearn.metrics import accuracy_score

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

1 # remove na values from DataFrame
2 train_df = train_df.dropna()
3 test_df = test_df.dropna()

1 # exploration
2 def explore(df):
3     print("Shape: ", df.shape, "\n")
4     print(df.dtypes, "\n")
5     print(df.head(), "\n")
6     # numeric data statistics
7     print(df.describe())
8     df.hist(figsize=(14,14), xrot=45)
9     plt.show()
10    # categorical data statistics
11    print(df.describe(include = 'object'))
12    for column in df.select_dtypes(include = 'object'):
13        if df[column].nunique() < 10:
14            sns.countplot(y = column, data = df)
15            plt.show()
16            plt.savefig('{}_dist.png'.format(column))
17
18
19 explore(train_df)
```

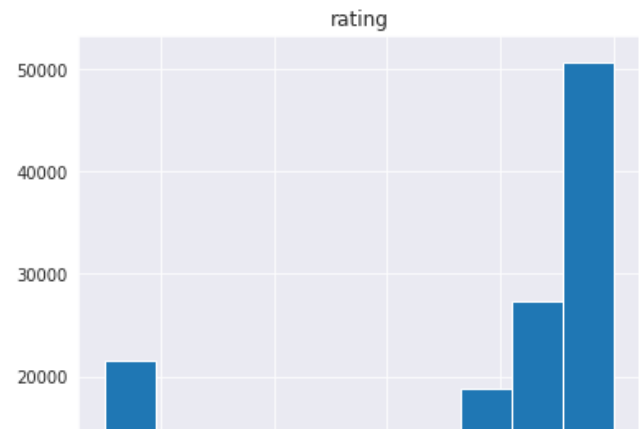
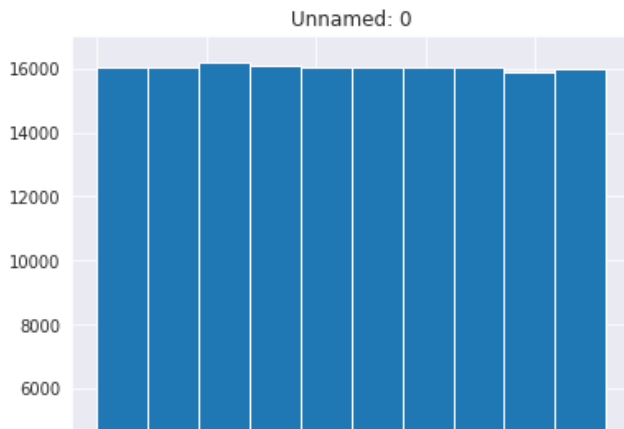
```
Shape: (160398, 7) /n
Unnamed: 0      int64
drugName        object
condition        object
review          object
rating          float64
date            object
usefulCount     int64
dtype: object /n
```

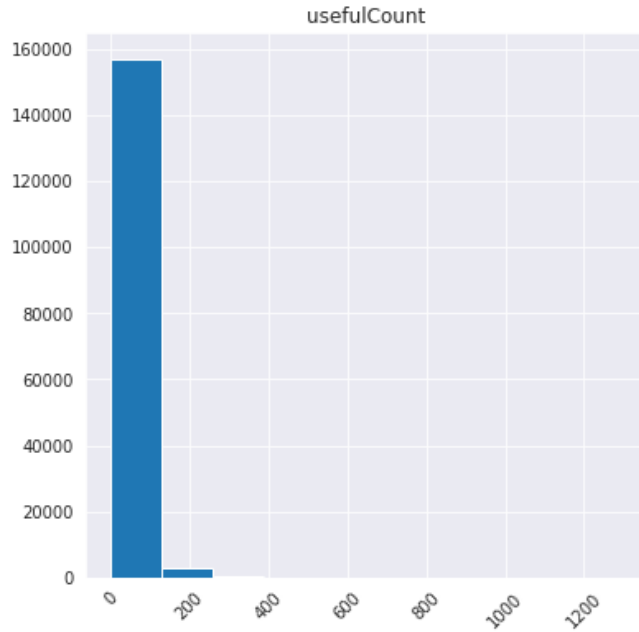
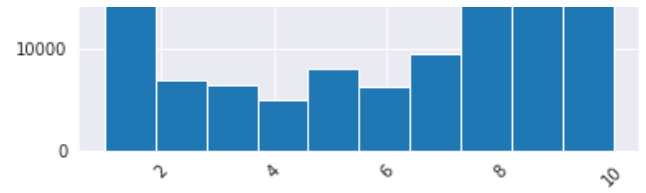
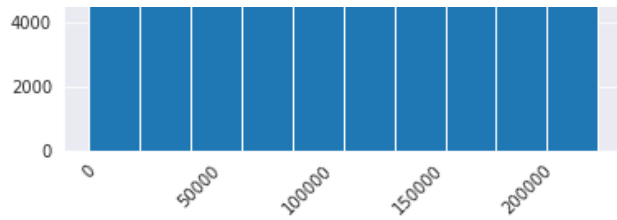
	Unnamed: 0	drugName	condition \
0	206461	Valsartan	Left Ventricular Dysfunction
1	95260	Guanfacine	ADHD
2	92703	Lybrel	Birth Control
3	138000	Ortho Evra	Birth Control
4	35696	Buprenorphine / naloxone	Opiate Dependence

	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

	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





	drugName	condition	review	date
count	160398	160398	160398	160398
unique	3431	884	112105	3579
top	Levonorgestrel	Birth Control	"Good"	March 1, 2016
freq	3631	28788	33	146

```
1 # obtain counts for conditions
2 condition_counts = train_df['condition'].value_counts()
3 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	

```
1 # reduce the number of classes to anything with more than 1000 reviews
2 condition_counts_1000 = condition_counts[condition_counts > 1000]
3
4 print(len(condition_counts_1000))
5
6 counts_1000_list = list()
7 for idx, name in enumerate(condition_counts_1000.index.tolist()):
8     counts_1000_list.append(name)
9
10 train_df = train_df[train_df['condition'].isin(counts_1000_list)]
11 test_df = test_df[test_df['condition'].isin(counts_1000_list)]
12
13 print(train_df['review'].shape)

32
(110121,)

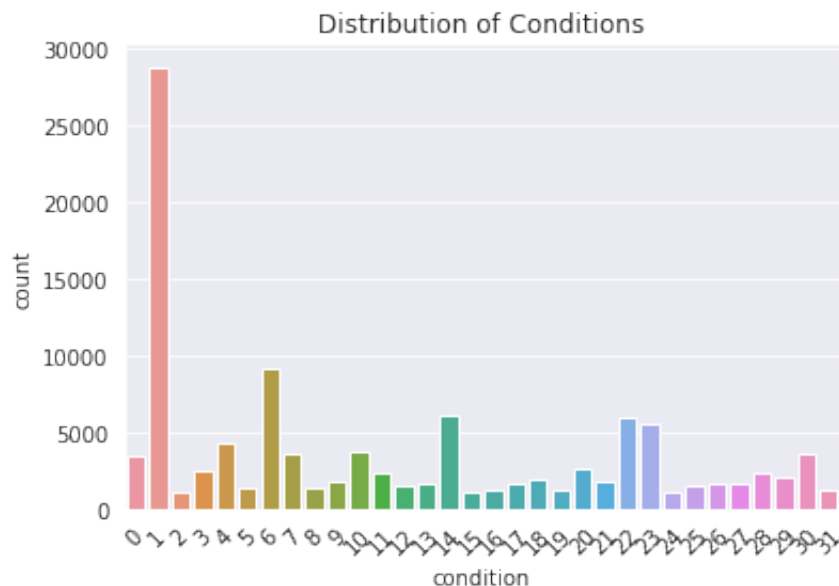
1 print(counts_1000_list)

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

```

1 # visualizations
2 fig, ax = plt.subplots(1, 1)
3 ax = sns.countplot(x = 'condition', data = train_df)
4 ax.set_title('Distribution of Conditions')
5 ax.set_xticklabels(ax.get_xticks(), rotation = 45)
6 fig.show()
7 fig.savefig('condition_dist.png')

```



```

1 # set data and labels
2 x_train = train_df['review']
3 x_test = test_df['review']
4 y_train = train_df['condition']
5 y_test = test_df['condition']

1 # labels to one hot encoded
2 def prepare_targets(y_train, y_test):
3     one_hot = LabelBinarizer()
4     one_hot.fit(y_train)
5     y_train = np.argmax(one_hot.transform(y_train), axis = 1)
6     y_test_one_hot = one_hot.transform(y_test)
7     y_test = np.argmax(one_hot.transform(y_test), axis = 1)
8     y_test_rev = one_hot.inverse_transform(y_test_one_hot)
9     return y_train, y_test, y_test_one_hot, y_test_rev
10
11 y_train, y_test, y_test_one_hot, y_test_rev = prepare_targets(y_train, y_test)

```

```
1 # create weights for classes
2 class_weights = class_weight.compute_class_weight('balanced', np.unique(y_train))
3
4 weights = dict(enumerate(class_weights))
5 print(weights)
```

```
72276825302986, 1: 1.6418326574427482, 2: 0.6158341535433071, 3: 0.58287284044
```

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

```
0.26142152722913886
```

```
1 # vectorize text
2 vocab_size = 1500
3 review_len_max = 200
4 encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(
5     max_tokens = vocab_size,
6     output_sequence_length = review_len_max)
7
8 # develop vocabulary
9 encoder.adapt(x_train.values)
10
11 # vectorize text
12 x_train = encoder(x_train)
13 x_test = encoder(x_test)
```

```
1 print('Training input shape: ', x_train.shape)
2 print(len(x_train))
3 print(len(x_train[0]))
4 print(x_train[0].shape)
5 print('Test input shape: ', x_test.shape)
6 print(len(x_test))
7 print(len(x_test[0]))
8 print(x_test[0].shape)
```

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

```
1 print(y_train.shape)

(110121,)
```



```

1 # classifiying with tf.keras RNN
2 model = tf.keras.models.Sequential([
3     tf.keras.layers.Embedding(input_dim = vocab_size + 1, output_dim = 16),
4     tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(units = 16, dropout = 0
5     tf.keras.layers.Dense(32, activation = 'softmax')
6 ]
7 )
8
9 # compile model
10 model.compile(loss = tf.keras.losses.SparseCategoricalCrossentropy(),
11               optimizer = 'Adam',
12               metrics = ['acc'])
13
14 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		

```

1 # test model
2 history = model.fit(x = x_train, y = y_train,
3                     epochs = 50,
4                     batch_size = 50,
5                     validation_split = 0.2,
6                     class_weight = weights,
7                     )

```

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

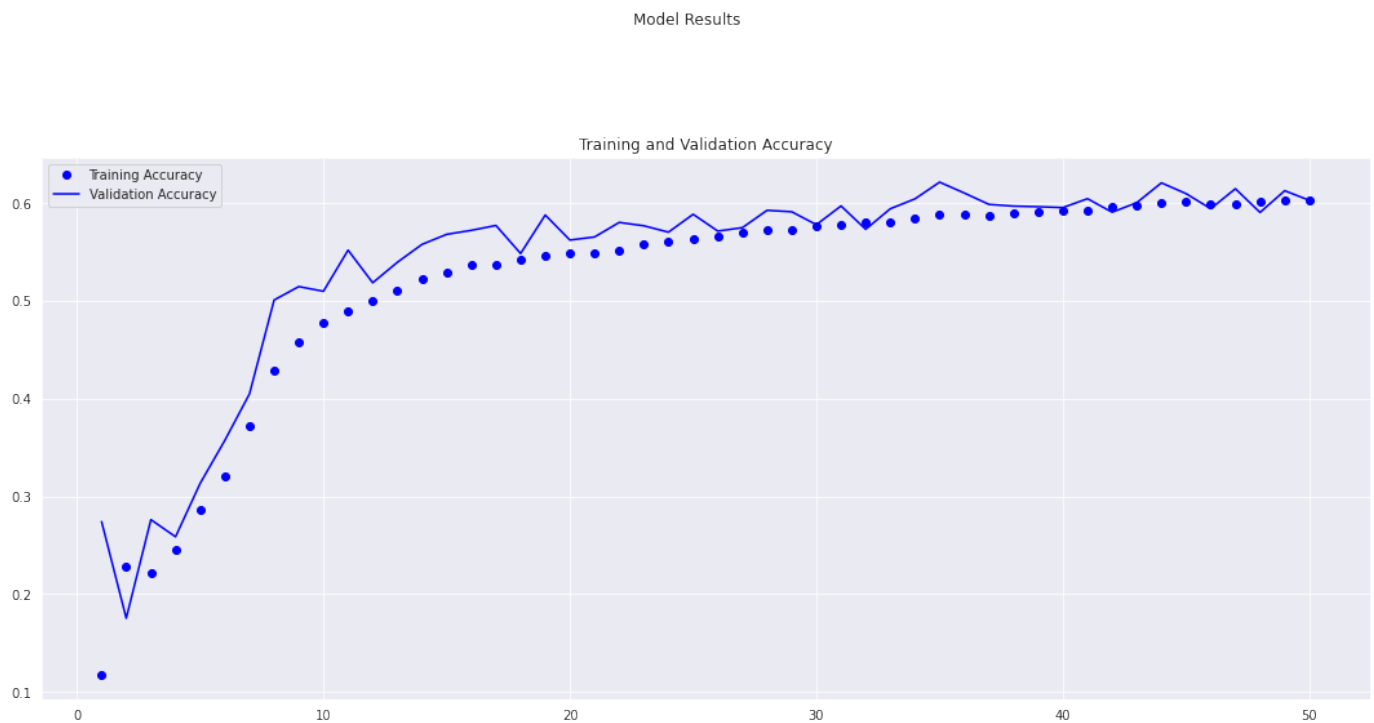
```
1762/1762 [=====] - 31s 17ms/step - loss: 1.2005 - a
Epoch 26/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.1817 - a
Epoch 27/50
1762/1762 [=====] - 32s 18ms/step - loss: 1.1740 - a
Epoch 28/50
1762/1762 [=====] - 33s 19ms/step - loss: 1.1762 - a
Epoch 29/50
1762/1762 [=====] - 32s 18ms/step - loss: 1.1684 - a
Epoch 30/50
1762/1762 [=====] - 34s 19ms/step - loss: 1.1671 - a
Epoch 31/50
1762/1762 [=====] - 34s 19ms/step - loss: 1.1591 - a
Epoch 32/50
1762/1762 [=====] - 35s 20ms/step - loss: 1.1555 - a
Epoch 33/50
1762/1762 [=====] - 35s 20ms/step - loss: 1.1334 - a
Epoch 34/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.1413 - a
Epoch 35/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.1393 - a
Epoch 36/50
1762/1762 [=====] - 35s 20ms/step - loss: 1.1348 - a
Epoch 37/50
1762/1762 [=====] - 32s 18ms/step - loss: 1.1417 - a
Epoch 38/50
1762/1762 [=====] - 35s 20ms/step - loss: 1.1259 - a
Epoch 39/50
1762/1762 [=====] - 33s 18ms/step - loss: 1.1276 - a
Epoch 40/50
1762/1762 [=====] - 32s 18ms/step - loss: 1.1097 - a
Epoch 41/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.1100 - a
Epoch 42/50
1762/1762 [=====] - 30s 17ms/step - loss: 1.1121 - a
Epoch 43/50
1762/1762 [=====] - 33s 19ms/step - loss: 1.1032 - a
Epoch 44/50
1762/1762 [=====] - 34s 19ms/step - loss: 1.0905 - a
Epoch 45/50
1762/1762 [=====] - 33s 19ms/step - loss: 1.0928 - a
Epoch 46/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.0922 - a
Epoch 47/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.0919 - a
Epoch 48/50
1762/1762 [=====] - 34s 19ms/step - loss: 1.0893 - a
Epoch 49/50
1762/1762 [=====] - 31s 18ms/step - loss: 1.0879 - a
Epoch 50/50
1762/1762 [=====] - 31s 17ms/step - loss: 1.0783 - a
```

```

1 # save model
2 model.save('drug_review_rnn_50.h5')

1 # model results
2 acc = history.history['acc']
3 val_acc = history.history['val_acc']
4 loss = history.history['loss']
5 val_loss = history.history['val_loss']
6 epochs = range(1, len(acc) + 1)
7
8 fig, (ax0, ax1) = plt.subplots(2, 1, figsize = (15,15))
9 fig.suptitle('Model Results')
10 ax0.plot(epochs, acc, 'bo', label = 'Training Accuracy')
11 ax0.plot(epochs, val_acc, 'b', label = 'Validation Accuracy')
12 ax0.set_title('Training and Validation Accuracy')
13 ax0.legend()
14
15 ax1.plot(epochs, loss, 'bo', label = 'Training Loss')
16 ax1.plot(epochs, val_loss, 'b', label = 'Validation Loss')
17 ax1.set_title('Training and Validation Loss')
18 ax1.legend()
19 fig.tight_layout(rect=[0, 0.03, 1, 0.9])
20
21 fig.savefig('model_results.png')

```





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

```

1 print(predictions[0])
2 print(y_test[0])
3 print(y_test_one_hot[0])
4 print(y_test_rev[0])

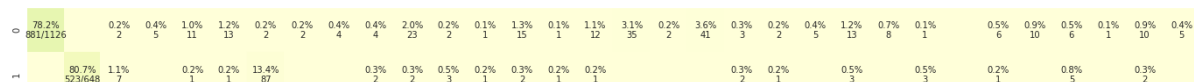
[4.1943714e-03 5.3141492e-05 1.1367062e-04 9.5583566e-02 2.2228974e-01
 1.9638129e-02 1.8428185e-03 7.2395586e-08 2.5014702e-05 5.0075510e-06
 2.4951330e-01 7.3006345e-06 3.9542447e-06 1.4613361e-06 1.5919979e-01
 1.8198247e-04 1.3924406e-03 4.0000530e-05 2.0378804e-01 6.9832919e-07
 7.5372423e-05 1.2797008e-06 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Depression

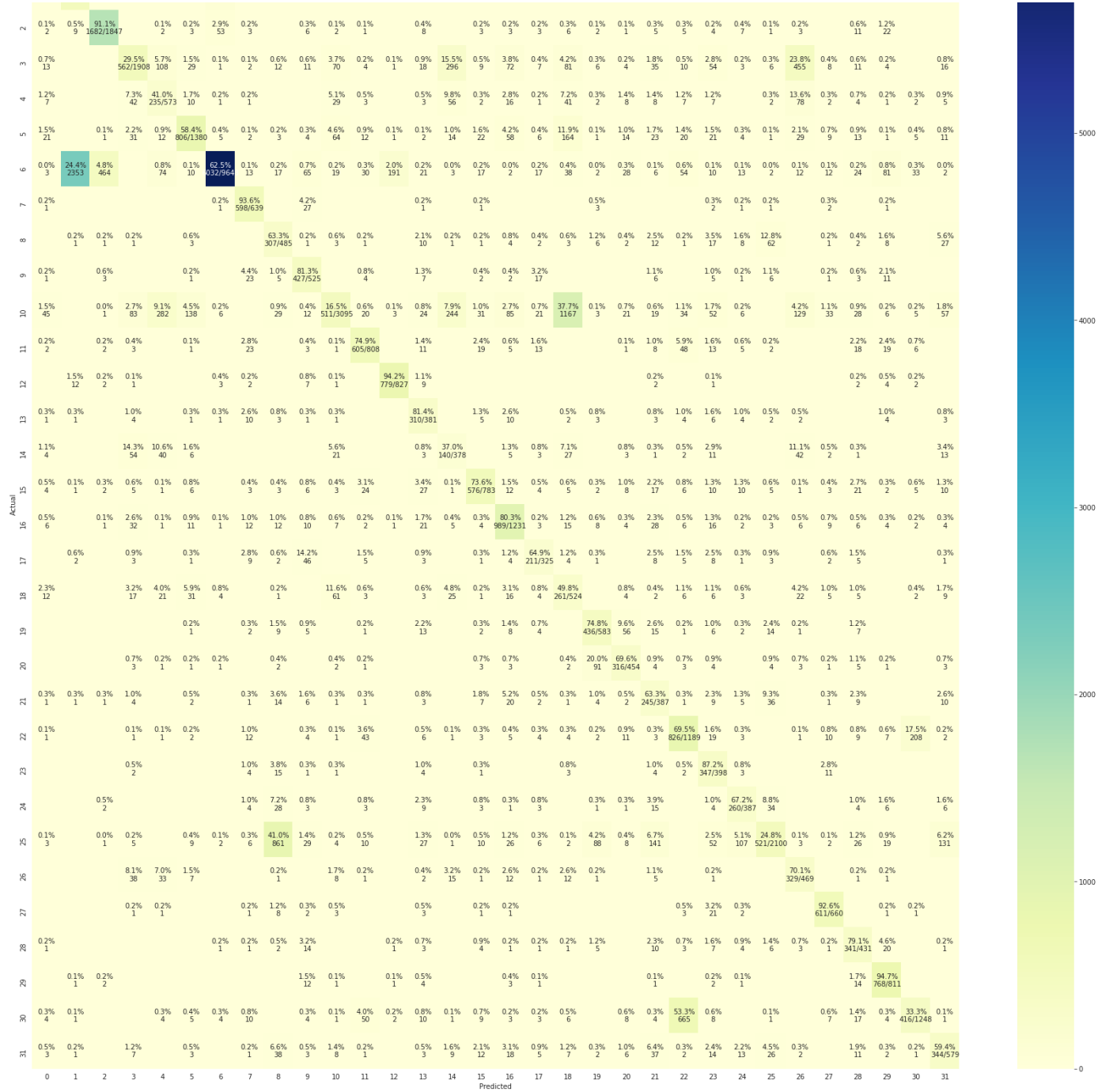
```

```

1 # build confusion matrix
2 def plot_cm(y_true, y_pred, figsize = (30, 30)):
3     cm = confusion_matrix(y_true, y_pred, labels = np.unique(y_true))
4     cm_sum = np.sum(cm, axis = 1, keepdims = True)
5     cm_perc = cm / cm_sum.astype(float) * 100
6     annot = np.empty_like(cm).astype(str)
7     nrows, ncols = cm.shape
8     for i in range(nrows):
9         for j in range(ncols):
10             c = cm[i, j]
11             p = cm_perc[i, j]
12             if i == j:
13                 s = cm_sum[i]
14                 annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
15             elif c == 0:
16                 annot[i, j] = ''
17             else:
18                 annot[i, j] = '%.1f%%\n%d' % (p, c)
19     cm = pd.DataFrame(cm, index = np.unique(y_true), columns = np.unique(y_true))
20     cm.index.name = 'Actual'
21     cm.columns.name = 'Predicted'
22     fig, ax = plt.subplots(figsize = figsize)
23     sns.heatmap(cm, cmap= "YlGnBu", annot = annot, fmt = '', ax = ax)
24     fig.savefig('confusion_matrix.png')
25
26 plot_cm(y_test_one_hot.argmax(axis = 1), predictions.argmax(axis = 1))

```





```
1 # load saved model
2 new_model = tf.keras.models.load_model('/content/drug_review_rnn_50.h5')
3
4 # make new predictions
5 new_predictions = new_model.predict(x_test)

1 print(y_test[0])
2 print(y_test_one_hot[0])
3 print(y_test_rev[0])

10
[0 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]
Depression

1 # model accuracy
2 accuracy_metric = accuracy_score(y_test, new_predictions.argmax(axis = 1))
3
4 print('Accuracy: ', accuracy_metric)

Accuracy:  0.6026828142395525
```

---

✓ 0s completed at 11:43 AM

