After the dismal performance of unnormalized features we will normalize the features based on their thread. This is mostly a copy of the previous notebook. The change comes in the normalization of the features.

To get all of this to run correctly we need to be in the correct python environment. Using Anaconda Here are the steps:

- conda create -n tf tensorflow
- conda activate tf
- conda install pandas
- conda install matplotlib
- conda install jupyter
- conda install scikit-learn==0.21.2 #this was used to not get an error on a mac system

Unfortunately environment files are not easily transferred between platforms. Hopefully this

In [1]:

```
import pandas as pd
import numpy as np
# import xml.etree.ElementTree as et
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
xml_file = 'stackexchange_data/diy.stackexchange.com/Posts_original.xml'
originaldf = pd.read_xml(xml_file,attrs_only=True,parser='etree')
originaldf.describe()
```

Out[2]:

	AcceptedAnswerId	AnswerCount	CommentCount	FavoriteCount	Id	LastE
count	22593.000000	64503.000000	173341.000000	7136.000000	173341.000000	60
mean	108373.832957	1.677674	1.950046	1.478840	118908.775829	34
std	70620.506794	1.453162	2.619226	2.210341	67767.548143	35
min	9.000000	0.000000	0.000000	0.000000	1.000000	
25%	41791.000000	1.000000	0.000000	1.000000	62355.000000	2
50%	106801.000000	1.000000	1.000000	1.000000	121874.000000	27
75%	170870.000000	2.000000	3.000000	1.000000	177914.000000	55
max	234205.000000	77.000000	48.000000	74.000000	234210.000000	141

In [3]:

```
originaldf.describe(exclude=[np.number])
```

Out[3]:

	Body	ContentLicense	CreationDate	LastActivityDate	L
count	173169	173341	173341	173341	
unique	173154	3	172934	137337	

	Body	ContentLicense	CreationDate	LastActivityDate	La
top	There's no need to use this tag. When ...	CC BY-SA 3.0	2011-10-16T21:46:14.993	2010-07-21T19:33:18.130	2020-06-16T

In [4]:

```
originaldf.isna().sum()
```

Out[4]:

```
AcceptedAnswerId          150748
AnswerCount               108838
Body                         172
CommentCount                   0
ContentLicense                 0
CreationDate                   0
FavoriteCount             166205
Id                             0
LastActivityDate               0
LastEditDate              112115
LastEditorUserId          112498
OwnerUserId                 1916
PostTypeId                     0
Score                          0
Tags                      108838
Title                     108838
ViewCount                 108838
ParentId                   65126
OwnerDisplayName          170670
CommunityOwnedDate        172872
LastEditorDisplayName     172946
ClosedDate                170901
dtype: int64
```

according to survey characteristics of good answers are :

- More varied vocabulary
- Answers referenced by other answers
- More comments from other users
- Earlier posted answers are likely to be better
- Answer most different from the rest
- Answer length (best)
- Forum specific easiest to look at are the answer length, time of posting and number of comments from other users. goal of this research is to find best answer. More interesting features are answers that are different from the rest. How to calculate answer similarity remains to be seen.. ###### start with comment count, answer length and time of posting? easy low hanging fruit

In [56]:

```
#a look at the columns that might help us to get to these
#body will give us word count
originaldf[['Body','CreationDate','CommentCount']]
```

Out[56]:

	Body	CreationDate	CommentCount
0	I'm looking to finish my basement and simply w...	2010-07-21 19:14:06	1
1	I would like to recaulk between the bathtub an...	2010-07-21 19:15:17	0
2	I'm going to be doing some drywalling shortly ...	2010-07-21 19:16:23	0
3	Other than looking up blue prints, which many ...	2010-07-21 19:16:23	1
4	I have a number of outlets that are old and wo...	2010-07-21 19:16:48	1
...	...	...	...
173336	I have an alcove I want to install some floati...	2021-09-05 01:27:37	1
173337	Summarize the problem\nMy 35 year-old home's w...	2021-09-05 02:31:01	0
173338	First, I'm going to try and describe the curre...	2021-09-05 02:32:28	0
173339	I need some help with confirming the wiring in...	2021-09-05 03:29:05	2
173340	To keep other gray water from backing up into ...	2021-09-05 04:35:47	0

173341 rows × 3 columns

In [5]:

```
#look at answers
originaldf.loc[originaldf['PostTypeId'] == 2].shape
```

Out[5]:

```
(108215, 22)
```

In [6]:

```
#look at number of questions
originaldf.loc[originaldf["PostTypeId"] == 1].shape
```

Out[6]:

```
(64503, 22)
```

In [7]:

```
#look at missing values
originaldf.loc[originaldf["PostTypeId"] == 1].isna().sum()
```

Out[7]:

```
AcceptedAnswerId        41910
AnswerCount                 0
Body                        0
CommentCount                0
ContentLicense              0
CreationDate                0
FavoriteCount           57367
Id                          0
LastActivityDate            0
LastEditDate            31375
LastEditorUserId        31502
OwnerUserId               656
PostTypeId                  0
Score                       0
Tags                        0
```

```
Title                        0
ViewCount                    0
ParentId                 64503
OwnerDisplayName         63386
CommunityOwnedDate       64475
LastEditorDisplayName    64372
ClosedDate               62063
dtype: int64
```

In [8]:

```
#look at missing values
originaldf.loc[originaldf["PostTypeId"] == 2].isna().sum()
```

Out[8]:

```
AcceptedAnswerId         108215
AnswerCount              108215
Body                          0
CommentCount                  0
ContentLicense                0
CreationDate                  0
FavoriteCount            108215
Id                            0
LastActivityDate              0
LastEditDate              80740
LastEditorUserId          80996
OwnerUserId                1260
PostTypeId                    0
Score                         0
Tags                     108215
Title                    108215
ViewCount                108215
ParentId                      0
OwnerDisplayName         106661
CommunityOwnedDate       107790
LastEditorDisplayName    107951
ClosedDate               108215
dtype: int64
```

In [9]:

```
#html tags in body columns with blank space
originaldf.Body = originaldf.Body.str.replace('<[^>]*>','', regex=True)
```

In [10]:

```
# Need a difference between answer posting time and question posting time

from datetime import datetime

datestrings = originaldf.CreationDate.str.slice_replace(start=-4)

dateObjects = []
for i in range(len(datestrings)):
    dateObjects.append(datetime.strptime(datestrings[i],'%Y-%m-%dT%H:%M:%S'))

originaldf.CreationDate = dateObjects
```

In [11]:

```
# want the question posting time for each answer
# so merge each answer with its question along with the body and creation dat
df = pd.merge(left=originaldf.loc[originaldf['PostTypeId'] == 2,
                    ['Id', 'CreationDate','Body','CommentCount','ParentId']],
                    right=originaldf[['Id','AcceptedAnswerId', 'Body',
                                      'CreationDate','AnswerCount']],
                    left_on="ParentId", right_on="Id", how="left",
                    suffixes=("_answer", "_question"))
```

In [12]:

```
#Assume that if there are no AcceptedAnswerId for the question then it is not
df.dropna(subset=["AcceptedAnswerId"],inplace=True)
df.reset_index(drop=True, inplace=True)
```

In [13]:

```
# if the id of the accepted answer for a question is the row's answer id
# then that row is accepted answer

df['is_accepted_answer'] = df.Id_answer == df.AcceptedAnswerId
```

In [14]:

```
#the count of unique accepted answers should be equal to the sum of "is_accep
len(df.AcceptedAnswerId.unique()) == df.is_accepted_answer.sum()
```

Out[14]: True

In [15]:

```
#the count of unique questions should also be equal to the sum of "is_accepte
len(df.Id_question.unique()) == df.is_accepted_answer.sum()
```

Out[15]: True

In [16]:

```
# calculate the difference between when the question and answers were posted
df['time_difference'] = df.CreationDate_answer - df.CreationDate_question

time_difference_in_seconds = []

for i in range(len(df.time_difference)):
        time_difference_in_seconds.append(df.time_difference[i].total_seconds

df.time_difference = time_difference_in_seconds
```

In [17]:

```
df.describe()
```

Out[17]:

	Id_answer	CommentCount	ParentId	Id_question	AcceptedAnswerId	An
count	46189.000000	46189.000000	46189.000000	46189.000000	46189.000000	461
mean	105381.628808	1.630821	97270.021196	97270.021196	98457.055619	
std	71227.945336	2.304975	72819.688767	72819.688767	72947.542389	
min	9.000000	0.000000	1.000000	1.000000	9.000000	
25%	38520.000000	0.000000	26454.000000	26454.000000	27136.000000	

	Id_answer	CommentCount	ParentId	Id_question	AcceptedAnswerId	Ans
50%	102446.000000	1.000000	89652.000000	89652.000000	91110.000000	
75%	169178.000000	2.000000	162114.000000	162114.000000	164504.000000	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

In [18]:

```
len(df.Id_question.unique())
```

Out[18]:

```
22593
```

So it looks like only ~22000 of the ~64000 questions have chosen answers. As there won't be reliable examples of chosen answers for the remaining 42000 we have removed them from the training set. (above)

In [19]:

```
df.shape
```

Out[19]:

```
(46189, 12)
```

In [20]:

```
df.drop(['ParentId'], axis=1, inplace=True)
```

In [21]:

```
answer_lengths = []
for body in df.Body_answer:
    answer_lengths.append(len(body.split()))
df['answer_length'] = answer_lengths
```

In [22]:

```
df.head()
```

Out[22]:

	Id_answer	CreationDate_answer	Body_answer	CommentCount	Id_question	AcceptedAnsw
0	9	2010-07-21 19:19:02	I've found that it works OK, but it's more dif...	1	3	
1	12	2010-07-21 19:20:53	I have used it for patching areas, but not for...	0	3	
2	13	2010-07-21 19:21:15	I just caulked my shower last night. I used GE...	3	2	
3	14	2010-07-21 19:21:41	It's just an ornamental wall it sounds like, s...	3	1	
4	15	2010-07-21 19:22:00	I just bought a permanent silicone product by ...	3	2	

In [53]:
```
df[['CommentCount', 'time_difference','answer_length']].describe()
```

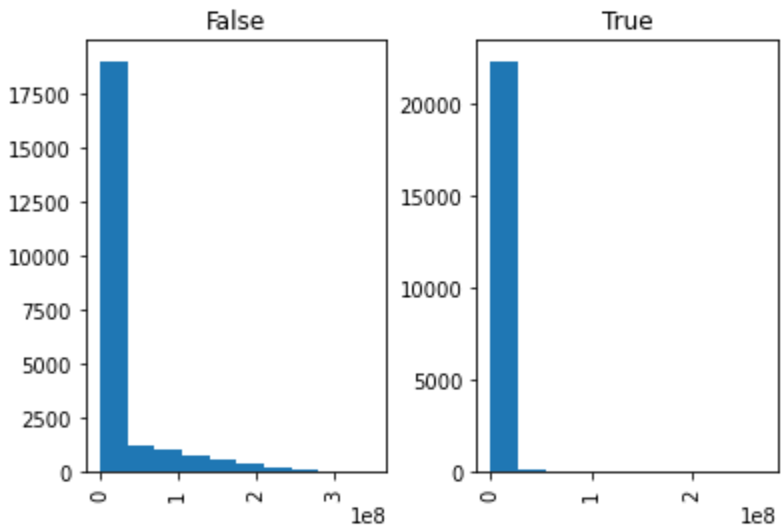
Out[53]:

	CommentCount	time_difference	answer_length
count	46189.000000	4.618900e+04	46189.000000
mean	1.630821	1.289551e+07	139.833986
std	2.304975	4.103870e+07	139.520765
min	0.000000	0.000000e+00	2.000000
25%	0.000000	3.382000e+03	59.000000
50%	1.000000	1.651600e+04	102.000000
75%	2.000000	1.013340e+05	173.000000
max	45.000000	3.504101e+08	4935.000000

Normalization of features by thread

In [23]:
```
df['time_difference'].hist(by=df.is_accepted_answer)
```

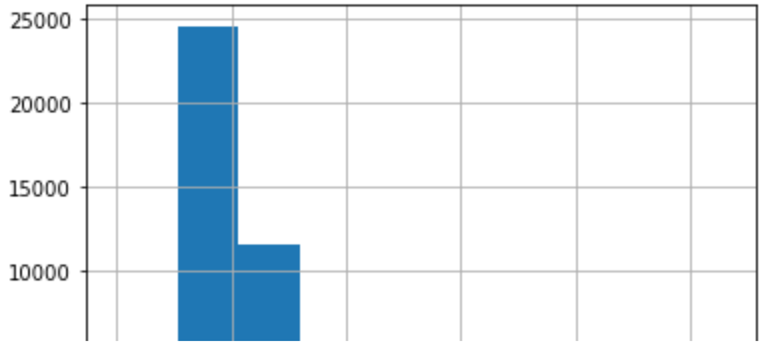
Out[23]:
```
array([<AxesSubplot:title={'center':'False'}>,
       <AxesSubplot:title={'center':'True'}>], dtype=object)
```



Due to the left skewness of the data we could apply a log or a root to scale it better. Somehow a few of the answers were posted in the same second as the question, so a log will not work, however an even powered root can work here.

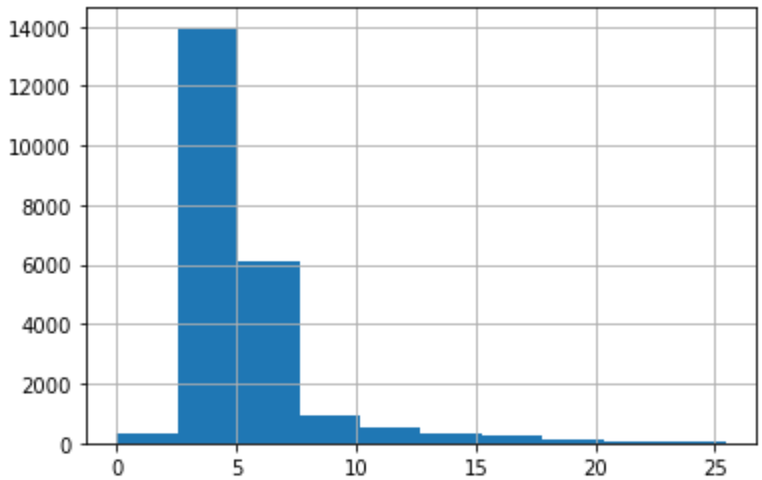
In [24]:
```
(df['time_difference']**(1/6)).hist()
```

Out[24]:
```
<AxesSubplot:>
```



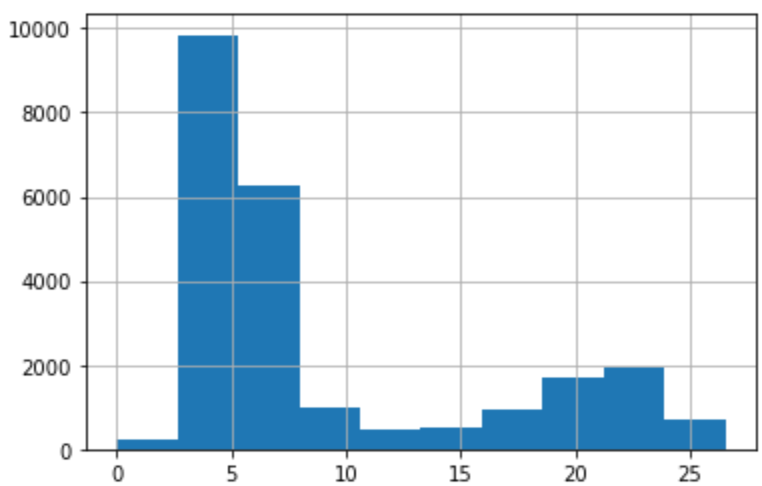
In [25]:
```
(df.loc[df.is_accepted_answer == 1]['time_difference']**(1/6)).hist()
```

Out[25]: <AxesSubplot:>



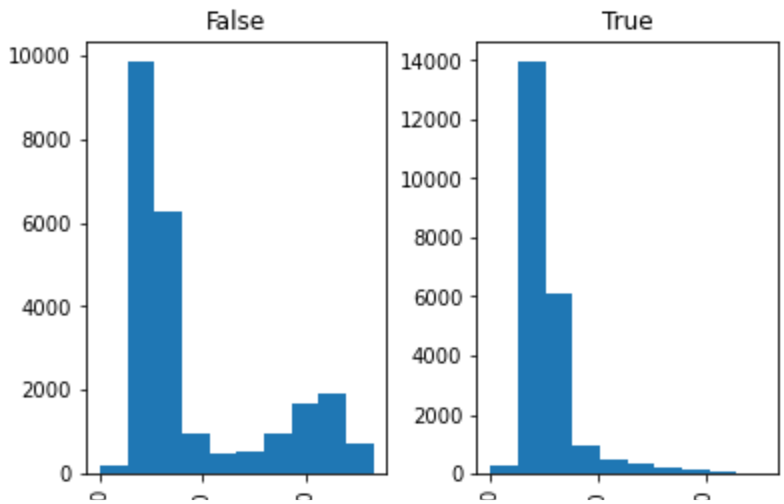
In [26]:
```
(df.loc[df.is_accepted_answer == 0]['time_difference']**(1/6)).hist()
```

Out[26]: <AxesSubplot:>



In [57]:
```
(df['time_difference']**(1/6)).hist(by=df.is_accepted_answer)
```

Out[57]:
```
array([<AxesSubplot:title={'center':'False'}>,
       <AxesSubplot:title={'center':'True'}>], dtype=object)
```



In [27]:

```
df.head()
```

Out[27]:

	Id_answer	CreationDate_answer	Body_answer	CommentCount	Id_question	AcceptedAnsw
0	9	2010-07-21 19:19:02	I've found that it works OK, but it's more dif...	1	3	
1	12	2010-07-21 19:20:53	I have used it for patching areas, but not for...	0	3	
2	13	2010-07-21 19:21:15	I just caulked my shower last night. I used GE...	3	2	
3	14	2010-07-21 19:21:41	It's just an ornamental wall it sounds like, s...	3	1	
4	15	2010-07-21 19:22:00	I just bought a permanent silicone product by ...	3	2	

In [28]:
```
# it looks like this will do not too badly for scaling.
#we can apply a by thread normalization now.
#write a function to do a maxmin scaling by thread
def maxminByThread(df,column_name):
    grouped_df = df.groupby(['Id_question'])[column_name]
    max_df = grouped_df.max().to_frame(name=column_name + '_max')
    min_df = grouped_df.min().to_frame(name=column_name + '_min')
#     max_series = grouped_df.max()
#     min_series = grouped_df.min()
    diff_df = pd.merge(left=max_df,right=min_df, on="Id_question")
    diff_df[column_name + '_difference'] = max_df[column_name + '_max'] - min

    df = pd.merge(left=df, right=diff_df, on="Id_question", how="left")
    minmax = (df[column_name] - df[column_name + '_min']) / df[column_name +
    minmax[minmax.isna()] = 0
#     diff = max_series - min_series
    # subtract the min from the root_time_difference and divide by difference
    # max and min.
#     maxmin_scaled = (df[column_name] - min_series) / diff
#     maxmin_scaled
    return minmax
```

In [29]:
```
#first calculate the max grouped by thread (question_id)
df['root_time_difference'] = df.time_difference**(1/6)
```

In [30]:
```
df['root_time_difference']
```

Out[30]:
```
0        2.327553
1        2.542303
2        2.664693
3        2.773332
4        2.717800
           ...
46184    3.854009
46185    6.603523
46186    4.637790
46187    6.512638
46188    5.182244
Name: root_time_difference, Length: 46189, dtype: float64
```

In [31]:
```
feature_names = []
for each in ['CommentCount', 'root_time_difference', 'answer_length']:
    feature_names.append('minmax_scaled_'+ each)
    df['minmax_scaled_' + each] = maxminByThread(df,each)
```

In [32]:
```
df.head()
```

Out[32]:

	Id_answer	CreationDate_answer	Body_answer	CommentCount	Id_question	AcceptedAnsw
0	9	2010-07-21 19:19:02	I've found that it works OK, but it's more dif...	1	3	

	Id_answer	CreationDate_answer	Body_answer	CommentCount	Id_question	AcceptedAnsw
1	12	2010-07-21 19:20:53	I have used it for patching areas, but not for...	0	3	
2	13	2010-07-21 19:21:15	I just caulked my shower last night. I used GE...	3	2	
3	14	2010-07-21 19:21:41	It's just an ornamental wall it sounds like, s...	3	1	
4	15	2010-07-21 19:22:00	I just bought a permanent silicone	3	2	

In [69]:

```
df[['minmax_scaled_CommentCount', 'minmax_scaled_root_time_difference', 'minm
```

Out[69]:

	minmax_scaled_CommentCount	minmax_scaled_root_time_difference	minmax_scaled_ar
count	46189.000000	46189.000000	4
mean	0.271149	0.372580	
std	0.419422	0.450733	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.000000	0.009168	
75%	0.555556	1.000000	
max	1.000000	1.000000	

Training The Neural Network

In previous notebooks we have done some feature generation. As before, it stands right now that there is no association between different answers that are in the same thread. The features now have been scaled relative to their thread, however, and a better result has been obtained than without the scaling. A random search of network parameters is performed in this section. After a few iterations we will see if we can get better performance from the network.

In [33]:

```
import tensorflow as tf
from tensorflow import keras
from sklearn.pipeline import Pipeline
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
```

In [34]:
```
print(tf.__version__)
print(keras.__version__)
```

```
2.0.0
2.2.4-tf
```

In [35]:
```
# do a train test split on the data
test_size = 0.2
train_full_size = 1-test_size
dev_size = test_size/train_full_size
# get the features discussed above
necessary_to_calculate_features = df[feature_names]
labels = df.is_accepted_answer
```

In [36]:
```
from sklearn.model_selection import train_test_split
```

In [37]:
```
# From: https://stackoverflow.com/questions/34842405/parameter-stratify-from-
X_train_full, X_test, y_train_full, y_test = train_test_split(necessary_to_ca
```

```
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/utils/
__init__.py:806: DeprecationWarning: `np.int` is a deprecated alias for the b
uiltin `int`. To silence this warning, use `int` by itself. Doing this will n
ot modify any behavior and is safe. When replacing `np.int`, you may wish to
use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re
view your current use, check the release note link for additional informatio
n.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  return floored.astype(np.int)
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/utils/
__init__.py:806: DeprecationWarning: `np.int` is a deprecated alias for the b
uiltin `int`. To silence this warning, use `int` by itself. Doing this will n
ot modify any behavior and is safe. When replacing `np.int`, you may wish to
use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re
view your current use, check the release note link for additional informatio
n.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  return floored.astype(np.int)
```

In [38]:
```
# also create a dev set
X_train, X_dev, y_train, y_dev = train_test_split(X_train_full, y_train_full,
```

```
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/utils/
__init__.py:806: DeprecationWarning: `np.int` is a deprecated alias for the b
uiltin `int`. To silence this warning, use `int` by itself. Doing this will n
ot modify any behavior and is safe. When replacing `np.int`, you may wish to
use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re
view your current use, check the release note link for additional informatio
n.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  return floored.astype(np.int)
```

```
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/utils/
__init__.py:806: DeprecationWarning: `np.int` is a deprecated alias for the b
uiltin `int`. To silence this warning, use `int` by itself. Doing this will n
ot modify any behavior and is safe. When replacing `np.int`, you may wish to
use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to re
view your current use, check the release note link for additional informatio
n.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  return floored.astype(np.int)
```

In [77]:

```
y_train.to_csv('y_train.csv', index = False)
```

In [58]:

```
#make a tunable model factory
def build_model(n_hidden=1, n_neurons=30, learning_rate=3e-3, input_shape=X_t
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=input_shape))
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="relu"))
    model.add(keras.layers.Dense(1, activation="sigmoid"))
    optimizer = keras.optimizers.SGD(lr=learning_rate)
    model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["
    return model
```

In [59]:

```
import os
root_logdir = os.path.join(os.curdir, "my_logs") #'./my_logs/' in MacOS

# this function creates a time for the log # e.g., './my_logs/run_2019_06_07-
def get_run_logdir():
    import time
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)

#create the callback for early stopping
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
                                                  restore_best_weights=True)

#create the tensorboard callback
tensorboard_cb = keras.callbacks.TensorBoard(get_run_logdir())
```

In [60]:

```
#set up the randomized search
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
    "n_hidden": tuple([0, 1, 2, 3]),
    "n_neurons": tuple(np.arange(1, 100))
#     "learning_rate": reciprocal(3e-4, 3e-2), # going to be choosing a rando
}

rnd_search_cv = RandomizedSearchCV(KerasClassifier(build_fn=build_model), par
rnd_search_cv.fit(X_train.values, y_train.values, epochs=100,
                  validation_data=(X_dev.values, y_dev.values),
                  callbacks=[early_stopping_cb, tensorboard_cb])
```

```
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/model_
selection/_search.py:269: DeprecationWarning: `np.int` is a deprecated alias
for the builtin `int`. To silence this warning, use `int` by itself. Doing th
is will not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wi
sh to review your current use, check the release note link for additional inf
ormation.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  random_state=rnd):
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/model_
selection/_split.py:442: DeprecationWarning: `np.int` is a deprecated alias f
or the builtin `int`. To silence this warning, use `int` by itself. Doing thi
s will not modify any behavior and is safe. When replacing `np.int`, you may
wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wi
sh to review your current use, check the release note link for additional inf
ormation.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/model_
selection/_split.py:102: DeprecationWarning: `np.bool` is a deprecated alias
for the builtin `bool`. To silence this warning, use `bool` by itself. Doing
this will not modify any behavior and is safe. If you specifically wanted the
numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/Users/chris/opt/anaconda3/envs/tf/lib/python3.7/site-packages/sklearn/model_
selection/_split.py:102: DeprecationWarning: `np.bool` is a deprecated alias
for the builtin `bool`. To silence this warning, use `bool` by itself. Doing
this will not modify any behavior and is safe. If you specifically wanted the
numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/de
vdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
Train on 13856 samples, validate on 9238 samples
Epoch 1/100
 1024/13856 [=>............................] - ETA: 16s - loss: 0.6858 - accu
racy: 0.6025

2021-12-08 17:07:30.671025: I tensorflow/core/profiler/lib/profiler_session.c
c:184] Profiler session started.
13856/13856 [==============================] - 5s 358us/sample - loss: 0.6786
- accuracy: 0.6827 - val_loss: 0.6698 - val_accuracy: 0.7151
Epoch 2/100
13856/13856 [==============================] - 2s 154us/sample - loss: 0.6632
- accuracy: 0.7101 - val_loss: 0.6568 - val_accuracy: 0.7042
Epoch 3/100
13856/13856 [==============================] - 2s 155us/sample - loss: 0.6515
- accuracy: 0.6969 - val_loss: 0.6459 - val_accuracy: 0.6910
Epoch 4/100
13856/13856 [==============================] - 2s 152us/sample - loss: 0.6416
- accuracy: 0.6983 - val_loss: 0.6364 - val_accuracy: 0.7094
Epoch 5/100
13856/13856 [==============================] - 2s 152us/sample - loss: 0.6329
- accuracy: 0.7084 - val_loss: 0.6283 - val_accuracy: 0.7033
Epoch 6/100
13856/13856 [==============================] - 3s 188us/sample - loss: 0.6255
```

```
- accuracy: 0.7070 - val_loss: 0.6212 - val_accuracy: 0.7057
Epoch 7/100
13856/13856 [==============================] - 2s 171us/sample - loss: 0.6191
- accuracy: 0.7049 - val_loss: 0.6152 - val_accuracy: 0.6994
Epoch 8/100
13856/13856 [==============================] - 2s 160us/sample - loss: 0.6137
- accuracy: 0.6997 - val_loss: 0.6103 - val_accuracy: 0.6972
Epoch 9/100
13856/13856 [==============================] - 2s 175us/sample - loss: 0.6094
- accuracy: 0.6967 - val_loss: 0.6063 - val_accuracy: 0.6956
Epoch 10/100
13856/13856 [==============================] - 2s 156us/sample - loss: 0.6058
- accuracy: 0.6941 - val_loss: 0.6029 - val_accuracy: 0.6940
Epoch 11/100
13856/13856 [==============================] - 2s 176us/sample - loss: 0.6028
- accuracy: 0.6911 - val_loss: 0.6002 - val_accuracy: 0.6928
Epoch 12/100
13856/13856 [==============================] - 3s 181us/sample - loss: 0.6002
- accuracy: 0.6904 - val_loss: 0.5978 - val_accuracy: 0.6921
Epoch 13/100
13856/13856 [==============================] - 2s 162us/sample - loss: 0.5980
- accuracy: 0.6900 - val_loss: 0.5958 - val_accuracy: 0.6915
Epoch 14/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5961
- accuracy: 0.6911 - val_loss: 0.5941 - val_accuracy: 0.6914
Epoch 15/100
13856/13856 [==============================] - 2s 152us/sample - loss: 0.5943
- accuracy: 0.6921 - val_loss: 0.5924 - val_accuracy: 0.6903
Epoch 16/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5926
- accuracy: 0.6911 - val_loss: 0.5908 - val_accuracy: 0.6906
Epoch 17/100
13856/13856 [==============================] - 2s 153us/sample - loss: 0.5910
- accuracy: 0.6915 - val_loss: 0.5894 - val_accuracy: 0.6902
Epoch 18/100
13856/13856 [==============================] - 2s 149us/sample - loss: 0.5896
- accuracy: 0.6916 - val_loss: 0.5882 - val_accuracy: 0.6903
Epoch 19/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5883
- accuracy: 0.6923 - val_loss: 0.5870 - val_accuracy: 0.6907
Epoch 20/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5871
- accuracy: 0.6926 - val_loss: 0.5859 - val_accuracy: 0.6903
Epoch 21/100
13856/13856 [==============================] - 2s 152us/sample - loss: 0.5859
- accuracy: 0.6926 - val_loss: 0.5848 - val_accuracy: 0.6903
Epoch 22/100
13856/13856 [==============================] - 2s 151us/sample - loss: 0.5848
- accuracy: 0.6928 - val_loss: 0.5837 - val_accuracy: 0.6901
Epoch 23/100
13856/13856 [==============================] - 2s 148us/sample - loss: 0.5837
- accuracy: 0.6931 - val_loss: 0.5827 - val_accuracy: 0.6907
Epoch 24/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5826
- accuracy: 0.6937 - val_loss: 0.5817 - val_accuracy: 0.6902
Epoch 25/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5815
- accuracy: 0.6933 - val_loss: 0.5809 - val_accuracy: 0.6912
Epoch 26/100
```

```
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5805
- accuracy: 0.6940 - val_loss: 0.5798 - val_accuracy: 0.6906
Epoch 27/100
13856/13856 [==============================] - 2s 154us/sample - loss: 0.5795
- accuracy: 0.6941 - val_loss: 0.5789 - val_accuracy: 0.6918
Epoch 28/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5785
- accuracy: 0.6940 - val_loss: 0.5781 - val_accuracy: 0.6916
Epoch 29/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5776
- accuracy: 0.6942 - val_loss: 0.5771 - val_accuracy: 0.6920
Epoch 30/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5767
- accuracy: 0.6944 - val_loss: 0.5763 - val_accuracy: 0.6924
Epoch 31/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5758
- accuracy: 0.6944 - val_loss: 0.5755 - val_accuracy: 0.6933
Epoch 32/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5750
- accuracy: 0.6957 - val_loss: 0.5748 - val_accuracy: 0.6933
Epoch 33/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5742
- accuracy: 0.6962 - val_loss: 0.5741 - val_accuracy: 0.6950
Epoch 34/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5735
- accuracy: 0.6970 - val_loss: 0.5734 - val_accuracy: 0.6963
Epoch 35/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5727
- accuracy: 0.6982 - val_loss: 0.5726 - val_accuracy: 0.6965
Epoch 36/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5720
- accuracy: 0.6993 - val_loss: 0.5720 - val_accuracy: 0.6969
Epoch 37/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5712
- accuracy: 0.6993 - val_loss: 0.5714 - val_accuracy: 0.7003
Epoch 38/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5706
- accuracy: 0.7009 - val_loss: 0.5707 - val_accuracy: 0.7000
Epoch 39/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5699
- accuracy: 0.7012 - val_loss: 0.5701 - val_accuracy: 0.7002
Epoch 40/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5692
- accuracy: 0.7008 - val_loss: 0.5696 - val_accuracy: 0.7024
Epoch 41/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5686
- accuracy: 0.7016 - val_loss: 0.5689 - val_accuracy: 0.7016
Epoch 42/100
13856/13856 [==============================] - 2s 156us/sample - loss: 0.5679
- accuracy: 0.6998 - val_loss: 0.5683 - val_accuracy: 0.6943
Epoch 43/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5673
- accuracy: 0.6944 - val_loss: 0.5676 - val_accuracy: 0.7019
Epoch 44/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5666
- accuracy: 0.6947 - val_loss: 0.5670 - val_accuracy: 0.6944
Epoch 45/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5660
- accuracy: 0.6955 - val_loss: 0.5666 - val_accuracy: 0.6953
```

```
Epoch 46/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5654
- accuracy: 0.6991 - val_loss: 0.5660 - val_accuracy: 0.7016
Epoch 47/100
13856/13856 [==============================] - 2s 147us/sample - loss: 0.5649
- accuracy: 0.7023 - val_loss: 0.5654 - val_accuracy: 0.7017
Epoch 48/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5643
- accuracy: 0.7067 - val_loss: 0.5649 - val_accuracy: 0.7020
Epoch 49/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5638
- accuracy: 0.7067 - val_loss: 0.5643 - val_accuracy: 0.7043
Epoch 50/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5632
- accuracy: 0.7071 - val_loss: 0.5639 - val_accuracy: 0.7056
Epoch 51/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5626
- accuracy: 0.7087 - val_loss: 0.5632 - val_accuracy: 0.7055
Epoch 52/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5621
- accuracy: 0.7092 - val_loss: 0.5627 - val_accuracy: 0.7058
Epoch 53/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5615
- accuracy: 0.7094 - val_loss: 0.5622 - val_accuracy: 0.7055
Epoch 54/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5610
- accuracy: 0.7099 - val_loss: 0.5617 - val_accuracy: 0.7053
Epoch 55/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5604
- accuracy: 0.7109 - val_loss: 0.5611 - val_accuracy: 0.7074
Epoch 56/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5598
- accuracy: 0.7118 - val_loss: 0.5605 - val_accuracy: 0.7088
Epoch 57/100
13856/13856 [==============================] - 2s 158us/sample - loss: 0.5593
- accuracy: 0.7130 - val_loss: 0.5600 - val_accuracy: 0.7091
Epoch 58/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5587
- accuracy: 0.7131 - val_loss: 0.5595 - val_accuracy: 0.7098
Epoch 59/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5581
- accuracy: 0.7140 - val_loss: 0.5590 - val_accuracy: 0.7111
Epoch 60/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5576
- accuracy: 0.7147 - val_loss: 0.5584 - val_accuracy: 0.7111
Epoch 61/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5571
- accuracy: 0.7151 - val_loss: 0.5580 - val_accuracy: 0.7115
Epoch 62/100
13856/13856 [==============================] - 2s 147us/sample - loss: 0.5566
- accuracy: 0.7171 - val_loss: 0.5575 - val_accuracy: 0.7117
Epoch 63/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5561
- accuracy: 0.7168 - val_loss: 0.5570 - val_accuracy: 0.7120
Epoch 64/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5556
- accuracy: 0.7184 - val_loss: 0.5565 - val_accuracy: 0.7126
Epoch 65/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5551
```

```
- accuracy: 0.7187 - val_loss: 0.5560 - val_accuracy: 0.7137
Epoch 66/100
13856/13856 [==============================] - 2s 146us/sample - loss: 0.5546
- accuracy: 0.7191 - val_loss: 0.5555 - val_accuracy: 0.7135
Epoch 67/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5541
- accuracy: 0.7199 - val_loss: 0.5551 - val_accuracy: 0.7139
Epoch 68/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5537
- accuracy: 0.7216 - val_loss: 0.5547 - val_accuracy: 0.7139
Epoch 69/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5532
- accuracy: 0.7212 - val_loss: 0.5541 - val_accuracy: 0.7147
Epoch 70/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5527
- accuracy: 0.7217 - val_loss: 0.5537 - val_accuracy: 0.7148
Epoch 71/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5522
- accuracy: 0.7228 - val_loss: 0.5533 - val_accuracy: 0.7152
Epoch 72/100
13856/13856 [==============================] - 2s 153us/sample - loss: 0.5518
- accuracy: 0.7231 - val_loss: 0.5528 - val_accuracy: 0.7160
Epoch 73/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5513
- accuracy: 0.7234 - val_loss: 0.5523 - val_accuracy: 0.7162
Epoch 74/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5508
- accuracy: 0.7237 - val_loss: 0.5518 - val_accuracy: 0.7169
Epoch 75/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5503
- accuracy: 0.7233 - val_loss: 0.5513 - val_accuracy: 0.7170
Epoch 76/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5498
- accuracy: 0.7244 - val_loss: 0.5508 - val_accuracy: 0.7179
Epoch 77/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5494
- accuracy: 0.7244 - val_loss: 0.5504 - val_accuracy: 0.7175
Epoch 78/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5489
- accuracy: 0.7249 - val_loss: 0.5499 - val_accuracy: 0.7179
Epoch 79/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5484
- accuracy: 0.7252 - val_loss: 0.5495 - val_accuracy: 0.7199
Epoch 80/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5480
- accuracy: 0.7252 - val_loss: 0.5490 - val_accuracy: 0.7193
Epoch 81/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5475
- accuracy: 0.7252 - val_loss: 0.5485 - val_accuracy: 0.7197
Epoch 82/100
13856/13856 [==============================] - 2s 142us/sample - loss: 0.5470
- accuracy: 0.7257 - val_loss: 0.5480 - val_accuracy: 0.7201
Epoch 83/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5465
- accuracy: 0.7260 - val_loss: 0.5475 - val_accuracy: 0.7209
Epoch 84/100
13856/13856 [==============================] - 2s 150us/sample - loss: 0.5460
- accuracy: 0.7260 - val_loss: 0.5470 - val_accuracy: 0.7220
Epoch 85/100
```

```
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5456
- accuracy: 0.7268 - val_loss: 0.5466 - val_accuracy: 0.7223
Epoch 86/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5451
- accuracy: 0.7268 - val_loss: 0.5462 - val_accuracy: 0.7234
Epoch 87/100
13856/13856 [==============================] - 2s 152us/sample - loss: 0.5446
- accuracy: 0.7276 - val_loss: 0.5457 - val_accuracy: 0.7223
Epoch 88/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5441
- accuracy: 0.7283 - val_loss: 0.5452 - val_accuracy: 0.7229
Epoch 89/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5436
- accuracy: 0.7281 - val_loss: 0.5447 - val_accuracy: 0.7229
Epoch 90/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5431
- accuracy: 0.7290 - val_loss: 0.5442 - val_accuracy: 0.7239
Epoch 91/100
13856/13856 [==============================] - 2s 143us/sample - loss: 0.5427
- accuracy: 0.7294 - val_loss: 0.5438 - val_accuracy: 0.7237
Epoch 92/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5422
- accuracy: 0.7305 - val_loss: 0.5433 - val_accuracy: 0.7226
Epoch 93/100
13856/13856 [==============================] - 2s 147us/sample - loss: 0.5417
- accuracy: 0.7307 - val_loss: 0.5428 - val_accuracy: 0.7242
Epoch 94/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5412
- accuracy: 0.7316 - val_loss: 0.5425 - val_accuracy: 0.7221
Epoch 95/100
13856/13856 [==============================] - 2s 145us/sample - loss: 0.5407
- accuracy: 0.7309 - val_loss: 0.5421 - val_accuracy: 0.7229
Epoch 96/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5404
- accuracy: 0.7306 - val_loss: 0.5414 - val_accuracy: 0.7244
Epoch 97/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5399
- accuracy: 0.7323 - val_loss: 0.5410 - val_accuracy: 0.7244
Epoch 98/100
13856/13856 [==============================] - 2s 144us/sample - loss: 0.5394
- accuracy: 0.7322 - val_loss: 0.5405 - val_accuracy: 0.7257
Epoch 99/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5390
- accuracy: 0.7332 - val_loss: 0.5400 - val_accuracy: 0.7255
Epoch 100/100
13856/13856 [==============================] - 2s 141us/sample - loss: 0.5385
- accuracy: 0.7333 - val_loss: 0.5395 - val_accuracy: 0.7265
13857/1 [===================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
============================================================================
```

```
erasClassifier object at 0x7f84acfc95d0>,
                   iid='warn', n_iter=1, n_jobs=None,
                   param_distributions={'n_hidden': (0, 1, 2, 3),
                                        'n_neurons': (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, ...)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [42]:

```
rnd_search_cv
```

Out[42]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
                   estimator=<tensorflow.python.keras.wrappers.scikit_learn.K
erasClassifier object at 0x7f84ae735810>,
                   iid='warn', n_iter=1, n_jobs=None,
                   param_distributions={'n_hidden': (0, 1, 2, 3),
                                        'n_neurons': (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, ...)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [61]:

```
%load_ext tensorboard
%tensorboard --logdir=./my_logs --port=6006
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 24117), started 1:32:21 ago. (Use '!kil
l 24117' to kill it.)
```

Index of file:///

☑ Show hidden objects

Name	Size	Last Modified	
.VolumeIcon.icns		1969-12-31	December 31, 1969
.file		2020-01-01	January 1, 2020
.vol		2020-01-01	January 1, 2020
Applications		2022-02-07	February 7, 2022
Library		2022-02-09	February 9, 2022
System		2020-01-01	January 1, 2020
Users		2020-01-01	January 1, 2020
Volumes		2022-02-07	February 7, 2022
bin		2020-01-01	January 1, 2020
cores		2019-11-09	November 9, 2019
dev		2022-02-01	February 1, 2022
etc		2022-02-01	February 1, 2022

In [62]:
```
rnd_search_cv.best_params_
# since this searches a random subset of possibilities then
# it will return a different best option every time.
# at one point it returned n_neurons: 94 and n_hidden: 2, so we will stick wi
```

Out[62]: {'n_neurons': 94, 'n_hidden': 2}

In [63]:
```
model = build_model(n_neurons=94, n_hidden=2)
```

In [65]:
```
history = model.fit(X_train.values, y_train.values, epochs=100,validation_dat
```

```
Train on 27713 samples, validate on 9238 samples
Epoch 1/100
27713/27713 [==============================] - 5s 163us/sample - loss: 0.5812
- accuracy: 0.6883 - val_loss: 0.5788 - val_accuracy: 0.6904
Epoch 2/100
27713/27713 [==============================] - 4s 139us/sample - loss: 0.5790
- accuracy: 0.6901 - val_loss: 0.5769 - val_accuracy: 0.6903
Epoch 3/100
27713/27713 [==============================] - 4s 138us/sample - loss: 0.5769
- accuracy: 0.6912 - val_loss: 0.5749 - val_accuracy: 0.6901
Epoch 4/100
27713/27713 [==============================] - 4s 136us/sample - loss: 0.5750
- accuracy: 0.6922 - val_loss: 0.5735 - val_accuracy: 0.6898
Epoch 5/100
27713/27713 [==============================] - 4s 137us/sample - loss: 0.5732
- accuracy: 0.6931 - val_loss: 0.5715 - val_accuracy: 0.6913
Epoch 6/100
```

```
27713/27713 [==============================] - 4s 132us/sample - loss: 0.5715
- accuracy: 0.6931 - val_loss: 0.5699 - val_accuracy: 0.6938
Epoch 7/100
27713/27713 [==============================] - 4s 133us/sample - loss: 0.5700
- accuracy: 0.6956 - val_loss: 0.5688 - val_accuracy: 0.6930
Epoch 8/100
27713/27713 [==============================] - 4s 147us/sample - loss: 0.5686
- accuracy: 0.6956 - val_loss: 0.5674 - val_accuracy: 0.6944
Epoch 9/100
27713/27713 [==============================] - 4s 135us/sample - loss: 0.5672
- accuracy: 0.6979 - val_loss: 0.5658 - val_accuracy: 0.6997
Epoch 10/100
27713/27713 [==============================] - 4s 133us/sample - loss: 0.5658
- accuracy: 0.6973 - val_loss: 0.5646 - val_accuracy: 0.7006
Epoch 11/100
27713/27713 [==============================] - 4s 134us/sample - loss: 0.5645
- accuracy: 0.6972 - val_loss: 0.5635 - val_accuracy: 0.7083
Epoch 12/100
27713/27713 [==============================] - 4s 138us/sample - loss: 0.5633
- accuracy: 0.6988 - val_loss: 0.5627 - val_accuracy: 0.7006
Epoch 13/100
27713/27713 [==============================] - 4s 126us/sample - loss: 0.5622
- accuracy: 0.7045 - val_loss: 0.5611 - val_accuracy: 0.7032
Epoch 14/100
27713/27713 [==============================] - 4s 147us/sample - loss: 0.5610
- accuracy: 0.7074 - val_loss: 0.5601 - val_accuracy: 0.7055
Epoch 15/100
27713/27713 [==============================] - 4s 132us/sample - loss: 0.5600
- accuracy: 0.7091 - val_loss: 0.5591 - val_accuracy: 0.7063
Epoch 16/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5589
- accuracy: 0.7092 - val_loss: 0.5582 - val_accuracy: 0.7117
Epoch 17/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5580
- accuracy: 0.7119 - val_loss: 0.5572 - val_accuracy: 0.7110
Epoch 18/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5571
- accuracy: 0.7125 - val_loss: 0.5562 - val_accuracy: 0.7126
Epoch 19/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5561
- accuracy: 0.7137 - val_loss: 0.5553 - val_accuracy: 0.7123
Epoch 20/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5552
- accuracy: 0.7144 - val_loss: 0.5544 - val_accuracy: 0.7137
Epoch 21/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5542
- accuracy: 0.7157 - val_loss: 0.5534 - val_accuracy: 0.7153
Epoch 22/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5533
- accuracy: 0.7170 - val_loss: 0.5525 - val_accuracy: 0.7162
Epoch 23/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5524
- accuracy: 0.7183 - val_loss: 0.5519 - val_accuracy: 0.7160
Epoch 24/100
27713/27713 [==============================] - 4s 130us/sample - loss: 0.5516
- accuracy: 0.7184 - val_loss: 0.5508 - val_accuracy: 0.7171
Epoch 25/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5507
- accuracy: 0.7191 - val_loss: 0.5498 - val_accuracy: 0.7193
```

```
Epoch 26/100
27713/27713 [==============================] - 4s 135us/sample - loss: 0.5498
- accuracy: 0.7216 - val_loss: 0.5490 - val_accuracy: 0.7183
Epoch 27/100
27713/27713 [==============================] - 4s 160us/sample - loss: 0.5489
- accuracy: 0.7222 - val_loss: 0.5482 - val_accuracy: 0.7186
Epoch 28/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5480
- accuracy: 0.7242 - val_loss: 0.5471 - val_accuracy: 0.7219
Epoch 29/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5471
- accuracy: 0.7252 - val_loss: 0.5462 - val_accuracy: 0.7216
Epoch 30/100
27713/27713 [==============================] - 4s 128us/sample - loss: 0.5462
- accuracy: 0.7263 - val_loss: 0.5456 - val_accuracy: 0.7225
Epoch 31/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5454
- accuracy: 0.7273 - val_loss: 0.5447 - val_accuracy: 0.7231
Epoch 32/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5445
- accuracy: 0.7278 - val_loss: 0.5436 - val_accuracy: 0.7248
Epoch 33/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5437
- accuracy: 0.7287 - val_loss: 0.5430 - val_accuracy: 0.7243
Epoch 34/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5428
- accuracy: 0.7293 - val_loss: 0.5419 - val_accuracy: 0.7246
Epoch 35/100
27713/27713 [==============================] - 4s 128us/sample - loss: 0.5420
- accuracy: 0.7298 - val_loss: 0.5412 - val_accuracy: 0.7250
Epoch 36/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5412
- accuracy: 0.7305 - val_loss: 0.5402 - val_accuracy: 0.7254
Epoch 37/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5404
- accuracy: 0.7315 - val_loss: 0.5395 - val_accuracy: 0.7257
Epoch 38/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5395
- accuracy: 0.7319 - val_loss: 0.5386 - val_accuracy: 0.7263
Epoch 39/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5387
- accuracy: 0.7324 - val_loss: 0.5379 - val_accuracy: 0.7276
Epoch 40/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5379
- accuracy: 0.7331 - val_loss: 0.5370 - val_accuracy: 0.7276
Epoch 41/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5372
- accuracy: 0.7337 - val_loss: 0.5362 - val_accuracy: 0.7283
Epoch 42/100
27713/27713 [==============================] - 4s 126us/sample - loss: 0.5364
- accuracy: 0.7338 - val_loss: 0.5356 - val_accuracy: 0.7294
Epoch 43/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5356
- accuracy: 0.7349 - val_loss: 0.5352 - val_accuracy: 0.7313
Epoch 44/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5348
- accuracy: 0.7360 - val_loss: 0.5338 - val_accuracy: 0.7293
Epoch 45/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5339
```

```
- accuracy: 0.7355 - val_loss: 0.5334 - val_accuracy: 0.7293
Epoch 46/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5332
- accuracy: 0.7365 - val_loss: 0.5324 - val_accuracy: 0.7295
Epoch 47/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5325
- accuracy: 0.7363 - val_loss: 0.5316 - val_accuracy: 0.7310
Epoch 48/100
27713/27713 [==============================] - 4s 126us/sample - loss: 0.5317
- accuracy: 0.7369 - val_loss: 0.5309 - val_accuracy: 0.7311
Epoch 49/100
27713/27713 [==============================] - 4s 130us/sample - loss: 0.5311
- accuracy: 0.7374 - val_loss: 0.5303 - val_accuracy: 0.7322
Epoch 50/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5303
- accuracy: 0.7382 - val_loss: 0.5295 - val_accuracy: 0.7325
Epoch 51/100
27713/27713 [==============================] - 4s 128us/sample - loss: 0.5296
- accuracy: 0.7384 - val_loss: 0.5291 - val_accuracy: 0.7320
Epoch 52/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5289
- accuracy: 0.7388 - val_loss: 0.5286 - val_accuracy: 0.7301
Epoch 53/100
27713/27713 [==============================] - 4s 128us/sample - loss: 0.5283
- accuracy: 0.7387 - val_loss: 0.5276 - val_accuracy: 0.7345
Epoch 54/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5277
- accuracy: 0.7396 - val_loss: 0.5280 - val_accuracy: 0.7363
Epoch 55/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5271
- accuracy: 0.7394 - val_loss: 0.5270 - val_accuracy: 0.7361
Epoch 56/100
27713/27713 [==============================] - 3s 124us/sample - loss: 0.5264
- accuracy: 0.7405 - val_loss: 0.5258 - val_accuracy: 0.7334
Epoch 57/100
27713/27713 [==============================] - 3s 124us/sample - loss: 0.5258
- accuracy: 0.7409 - val_loss: 0.5250 - val_accuracy: 0.7351
Epoch 58/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5252
- accuracy: 0.7418 - val_loss: 0.5252 - val_accuracy: 0.7416
Epoch 59/100
27713/27713 [==============================] - 3s 124us/sample - loss: 0.5246
- accuracy: 0.7428 - val_loss: 0.5240 - val_accuracy: 0.7367
Epoch 60/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5240
- accuracy: 0.7434 - val_loss: 0.5234 - val_accuracy: 0.7357
Epoch 61/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5234
- accuracy: 0.7447 - val_loss: 0.5229 - val_accuracy: 0.7354
Epoch 62/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5230
- accuracy: 0.7440 - val_loss: 0.5224 - val_accuracy: 0.7373
Epoch 63/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5224
- accuracy: 0.7450 - val_loss: 0.5218 - val_accuracy: 0.7462
Epoch 64/100
27713/27713 [==============================] - 4s 131us/sample - loss: 0.5219
- accuracy: 0.7469 - val_loss: 0.5212 - val_accuracy: 0.7463
Epoch 65/100
```

```
27713/27713 [==============================] - 3s 124us/sample - loss: 0.5214
- accuracy: 0.7474 - val_loss: 0.5208 - val_accuracy: 0.7454
Epoch 66/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5209
- accuracy: 0.7490 - val_loss: 0.5205 - val_accuracy: 0.7383
Epoch 67/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5204
- accuracy: 0.7494 - val_loss: 0.5202 - val_accuracy: 0.7386
Epoch 68/100
27713/27713 [==============================] - 4s 126us/sample - loss: 0.5200
- accuracy: 0.7496 - val_loss: 0.5197 - val_accuracy: 0.7476
Epoch 69/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5196
- accuracy: 0.7497 - val_loss: 0.5190 - val_accuracy: 0.7459
Epoch 70/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5190
- accuracy: 0.7499 - val_loss: 0.5186 - val_accuracy: 0.7457
Epoch 71/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5187
- accuracy: 0.7503 - val_loss: 0.5189 - val_accuracy: 0.7455
Epoch 72/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5182
- accuracy: 0.7500 - val_loss: 0.5179 - val_accuracy: 0.7468
Epoch 73/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5179
- accuracy: 0.7512 - val_loss: 0.5174 - val_accuracy: 0.7483
Epoch 74/100
27713/27713 [==============================] - 4s 130us/sample - loss: 0.5175
- accuracy: 0.7524 - val_loss: 0.5171 - val_accuracy: 0.7497
Epoch 75/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5171
- accuracy: 0.7521 - val_loss: 0.5167 - val_accuracy: 0.7490
Epoch 76/100
27713/27713 [==============================] - 3s 124us/sample - loss: 0.5168
- accuracy: 0.7527 - val_loss: 0.5166 - val_accuracy: 0.7490
Epoch 77/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5164
- accuracy: 0.7527 - val_loss: 0.5159 - val_accuracy: 0.7480
Epoch 78/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5160
- accuracy: 0.7524 - val_loss: 0.5158 - val_accuracy: 0.7489
Epoch 79/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5156
- accuracy: 0.7524 - val_loss: 0.5154 - val_accuracy: 0.7476
Epoch 80/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5153
- accuracy: 0.7530 - val_loss: 0.5150 - val_accuracy: 0.7497
Epoch 81/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5151
- accuracy: 0.7533 - val_loss: 0.5160 - val_accuracy: 0.7462
Epoch 82/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5148
- accuracy: 0.7533 - val_loss: 0.5150 - val_accuracy: 0.7470
Epoch 83/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5145
- accuracy: 0.7531 - val_loss: 0.5162 - val_accuracy: 0.7514
Epoch 84/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5143
- accuracy: 0.7534 - val_loss: 0.5139 - val_accuracy: 0.7498
```

```
Epoch 85/100
27713/27713 [==============================] - 4s 131us/sample - loss: 0.5140
- accuracy: 0.7534 - val_loss: 0.5135 - val_accuracy: 0.7492
Epoch 86/100
27713/27713 [==============================] - 4s 129us/sample - loss: 0.5135
- accuracy: 0.7537 - val_loss: 0.5138 - val_accuracy: 0.7509
Epoch 87/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5134
- accuracy: 0.7547 - val_loss: 0.5140 - val_accuracy: 0.7476
Epoch 88/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5130
- accuracy: 0.7544 - val_loss: 0.5128 - val_accuracy: 0.7506
Epoch 89/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5129
- accuracy: 0.7547 - val_loss: 0.5127 - val_accuracy: 0.7505
Epoch 90/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5127
- accuracy: 0.7555 - val_loss: 0.5132 - val_accuracy: 0.7492
Epoch 91/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5123
- accuracy: 0.7550 - val_loss: 0.5128 - val_accuracy: 0.7502
Epoch 92/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5123
- accuracy: 0.7555 - val_loss: 0.5142 - val_accuracy: 0.7468
Epoch 93/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5120
- accuracy: 0.7553 - val_loss: 0.5119 - val_accuracy: 0.7514
Epoch 94/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5118
- accuracy: 0.7547 - val_loss: 0.5115 - val_accuracy: 0.7511
Epoch 95/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5116
- accuracy: 0.7557 - val_loss: 0.5120 - val_accuracy: 0.7529
Epoch 96/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5113
- accuracy: 0.7552 - val_loss: 0.5115 - val_accuracy: 0.7509
Epoch 97/100
27713/27713 [==============================] - 3s 126us/sample - loss: 0.5111
- accuracy: 0.7551 - val_loss: 0.5108 - val_accuracy: 0.7515
Epoch 98/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5110
- accuracy: 0.7561 - val_loss: 0.5106 - val_accuracy: 0.7517
Epoch 99/100
27713/27713 [==============================] - 3s 125us/sample - loss: 0.5108
- accuracy: 0.7562 - val_loss: 0.5105 - val_accuracy: 0.7510
Epoch 100/100
27713/27713 [==============================] - 4s 127us/sample - loss: 0.5107
```

In [66]:

```
y_test_pred = model.predict_classes(X_test.values)
```

In [67]:

```
from sklearn.metrics import classification_report
```

In [68]:

```
#trying the prediction on the test data.
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

       False       0.75      0.78      0.77      4719
        True       0.76      0.73      0.75      4519

    accuracy                           0.76      9238
   macro avg       0.76      0.76      0.76      9238
weighted avg       0.76      0.76      0.76      9238
```