2192 lines (2192 sloc) | 143 KB

Raw Blame

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.cm as cm
import matplotlib.pyplot as plt

# Importing data
path = "/content/train.csv"
data_train = pd.read_csv(path)
```

In [2]:
```
data_train.head()
```

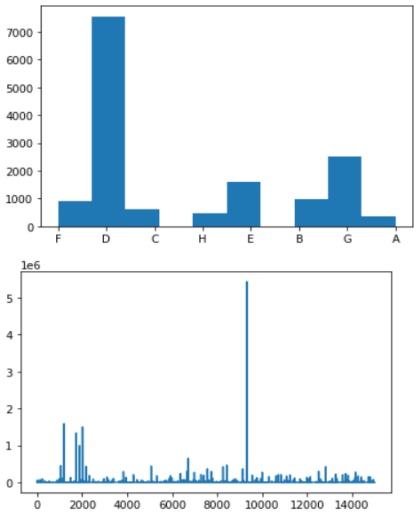
Out[2]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40	1031602	8523	363	1095	2016-09-14	PT7M37S	F
1	VID_14135	2	1707	56	2	6	2016-10-01	PT9M30S	D
2	VID_2187	1	2023	25	0	2	2016-07-02	PT2M16S	C
3	VID_23096	6	620860	777	161	153	2016-07-27	PT4M22S	H
4	VID_10175	1	666	1	0	0	2016-06-29	PT31S	D

In [3]:
```
data_train.shape
```

Out[3]:
```
(14999, 9)
```

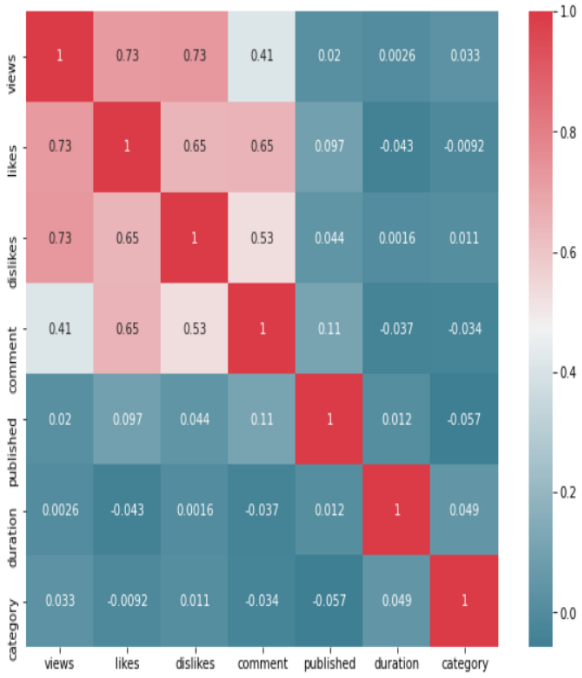
In [4]:
```
# Visualization
# Individual Plots
```



In [5]:
```
# Remove videos with adview greater than 2000000 as outlier
data_train = data_train[data_train["adview"] <2000000]
```

In [31]:

```
# Heatmap
import seaborn as sns
f, ax = plt.subplots(figsize=(10, 8))
corr = data_train.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
square=True, ax=ax,annot=True)
plt.show()
```



In [7]:

```
# Removing character "F" present in data
data_train=data_train[data_train.views!='F']
data_train=data_train[data_train.likes!='F']
data_train=data_train[data_train.dislikes!='F']
data_train=data_train[data_train.comment!='F']
```

In [8]:

```
data_train.head()
```

Out[8]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40	1031602	8523	363	1095	2016-09-14	PT7M37S	F
1	VID_14135	2	1707	56	2	6	2016-10-01	PT9M30S	D
2	VID_2187	1	2023	25	0	2	2016-07-02	PT2M16S	C
3	VID_23096	6	620860	777	161	153	2016-07-27	PT4M22S	H
4	VID_10175	1	666	1	0	0	2016-06-29	PT31S	D

In [9]:

```
# Assigning each category a number for Category feature
category={'A': 1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7,'H':8}
data_train["category"]=data_train["category"].map(category)
data_train.head()
```

Out[9]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	VID_18655	40	1031602	8523	363	1095	2016-09-14	PT7M37S	6
1	VID_14135	2	1707	56	2	6	2016-10-01	PT9M30S	4
2	VID_2187	1	2023	25	0	2	2016-07-02	PT2M16S	3
3	VID_23096	6	620860	777	161	153	2016-07-27	PT4M22S	8
4	VID_10175	1	666	1	0	0	2016-06-29	PT31S	4

In [10]:

```
# Convert values to integers for views, likes, comments, dislikes and adview
data_train["views"] = pd.to_numeric(data_train["views"])
data_train["comment"] = pd.to_numeric(data_train["comment"])
data_train["likes"] = pd.to_numeric(data_train["likes"])
data_train["dislikes"] = pd.to_numeric(data_train["dislikes"])
data_train["adview"]=pd.to_numeric(data_train["adview"])
column_vidid=data_train['vidid']

# Endoding features like Category, Duration, Vidid
from sklearn.preprocessing import LabelEncoder
data_train['duration']=LabelEncoder().fit_transform(data_train['duration'])
data_train['vidid']=LabelEncoder().fit_transform(data_train['vidid'])
data_train['published']=LabelEncoder().fit_transform(data_train['published'])
data_train.head()

# Convert Time_in_sec for duration
import datetime
import time
def checki(x):
  y = x[2:]
  h = ''
  m = ''
  s = ''
  mm = ''
  P = ['H','M','S']
  for i in y:
    if i not in P:
      mm+=i
    else:
      if(i=="H"):
        h = mm
        mm = ''
      elif(i == "M"):
        m = mm
        mm = ''
      else:
        s = mm
        mm = ''
  if(h==''):
    h = '00'
  if(m == ''):
    m = '00'
  if(s==''):
    s='00'
  bp = h+':'+m+':'+s
  return bp

train=pd.read_csv("train.csv")
mp = pd.read_csv(path)["duration"]
time = mp.apply(checki)

def func_sec(time_string):
  h, m, s = time_string.split(':')
  return int(h) * 3600 + int(m) * 60 + int(s)

time1=time.apply(func_sec)

data_train["duration"]=time1
data_train.head()
```

Out[10]:

	vidid	adview	views	likes	dislikes	comment	published	duration	category
0	5912	40	1031602	8523	363	1095	2168	457	6
1	2741	2	1707	56	2	6	2185	570	4
2	8138	1	2023	25	0	2	2094	136	3
3	9004	6	620860	777	161	153	2119	262	8
4	122	1	666	1	0	0	2091	31	4

In [11]:

```
# Split Data
Y_train = pd.DataFrame(data = data_train.iloc[:, 1].values, columns = ['target'])
data_train=data_train.drop(["adview"],axis=1)
data_train=data_train.drop(["vidid"],axis=1)
data_train.head()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_train, Y_train, test_size=0.2, random_state=42)

X_train.shape

# Normalise Data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)

X_train.mean()
```

Out[11]:

```
0.1739096800320488
```

In [12]:

```
# Evaluation Metrics
from sklearn import metrics
def print_error(X_test, y_test, model_name):
  prediction = model_name.predict(X_test)
  print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, prediction))
  print('Mean Squared Error:', metrics.mean_squared_error(y_test, prediction))
  print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))

# Linear Regression
from sklearn import linear_model
linear_regression = linear_model.LinearRegression()
linear_regression.fit(X_train, y_train)
print_error(X_test,y_test, linear_regression)

# Support Vector Regressor
from sklearn.svm import SVR
supportvector_regressor = SVR()
supportvector_regressor.fit(X_train,y_train)
print_error(X_test,y_test, linear_regression)
```

```
Mean Absolute Error: 3707.378005824532
Mean Squared Error: 835663131.1210337
Root Mean Squared Error: 28907.83857573986
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Mean Absolute Error: 3707.378005824532
Mean Squared Error: 835663131.1210337
Root Mean Squared Error: 28907.83857573986
```

In [13]:

```
# Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
decision_tree = DecisionTreeRegressor()
decision_tree.fit(X_train, y_train)
print_error(X_test,y_test, decision_tree)

# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
n_estimators = 200
max_depth = 25
min_samples_split=15
min_samples_leaf=2
random_forest = RandomForestRegressor(n_estimators = n_estimators, max_depth = max_depth, min_samples_split=min_samples_split)
random_forest.fit(X_train,y_train)
print_error(X_test,y_test, random_forest)
```

```
Mean Absolute Error: 2881.942281420765
Mean Squared Error: 1200253501.0249317
Root Mean Squared Error: 34644.67493028231
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples,), for example using ravel().

Mean Absolute Error: 3287.306282933801
Mean Squared Error: 612645714.2800009
Root Mean Squared Error: 24751.681039476913
```

In [14]:

```
# Artificial Neural Network
import keras
from keras.layers import Dense
ann = keras.models.Sequential([
                                Dense(6, activation="relu",
                                input_shape=X_train.shape[1:]),
                                Dense(6,activation="relu"),
                                Dense(1)
                                ])

optimizer=keras.optimizers.Adam()
loss=keras.losses.mean_squared_error
ann.compile(optimizer=optimizer,loss=loss,metrics=["mean_squared_error"])

history=ann.fit(X_train,y_train,epochs=100)

ann.summary()

print_error(X_test,y_test,ann)
```

```
Epoch 1/100
366/366 [==============================] - 1s 1ms/step - loss: 1460605039.5555 - mean_squared_error: 1460605039.5555
Epoch 2/100
366/366 [==============================] - 0s 1ms/step - loss: 835850050.6294 - mean_squared_error: 835850050.6294
Epoch 3/100
366/366 [==============================] - 0s 1ms/step - loss: 652082777.9837 - mean_squared_error: 652082777.9837
Epoch 4/100
366/366 [==============================] - 0s 1ms/step - loss: 356579735.7527 - mean_squared_error: 356579735.7527
Epoch 5/100
366/366 [==============================] - 0s 1ms/step - loss: 508746306.1304 - mean_squared_error: 508746306.1304
Epoch 6/100
366/366 [==============================] - 0s 946us/step - loss: 353001179.8001 - mean_squared_error: 353001179.8001
Epoch 7/100
366/366 [==============================] - 0s 937us/step - loss: 1090564028.9414 - mean_squared_error: 1090564028.9414
Epoch 8/100
366/366 [==============================] - 0s 931us/step - loss: 890400791.0762 - mean_squared_error: 890400791.0762
Epoch 9/100
366/366 [==============================] - 0s 953us/step - loss: 1292254084.3379 - mean_squared_error: 1292254084.3379
Epoch 10/100
366/366 [==============================] - 0s 972us/step - loss: 901213933.3185 - mean_squared_error: 901213933.3185
Epoch 11/100
366/366 [==============================] - 0s 1ms/step - loss: 908548840.6676 - mean_squared_error: 908548840.6676
Epoch 12/100
Epoch 13/100
366/366 [==============================] - 0s 973us/step - loss: 706570693.7939 - mean_squared_error: 706570693.7939
Epoch 14/100
366/366 [==============================] - 0s 1ms/step - loss: 504222550.4387 - mean_squared_error: 504222550.4387
Epoch 15/100
366/366 [==============================] - 0s 928us/step - loss: 706137562.4968 - mean_squared_error: 706137562.4968
Epoch 16/100
366/366 [==============================] - 0s 1ms/step - loss: 609411887.3995 - mean_squared_error: 609411887.3995
Epoch 17/100
366/366 [==============================] - 0s 952us/step - loss: 378216780.9687 - mean_squared_error: 378216780.9687
Epoch 18/100
366/366 [==============================] - 0s 1ms/step - loss: 1651339645.2425 - mean_squared_error: 1651339645.2425
Epoch 19/100
366/366 [==============================] - 0s 1ms/step - loss: 565605821.0043 - mean_squared_error: 565605821.0043
Epoch 20/100
366/366 [==============================] - 0s 1ms/step - loss: 1409569655.0599 - mean_squared_error: 1409569655.0599
Epoch 21/100
366/366 [==============================] - 0s 1ms/step - loss: 904555647.1029 - mean_squared_error: 904555647.1029
Epoch 22/100
366/366 [==============================] - 0s 1ms/step - loss: 1046805510.8011 - mean_squared_error: 1046805510.8011
Epoch 23/100
366/366 [==============================] - 0s 1ms/step - loss: 1072898878.5443 - mean_squared_error: 1072898878.5443
Epoch 24/100
366/366 [==============================] - 0s 1ms/step - loss: 806679596.5186 - mean_squared_error: 806679596.5186
Epoch 25/100
366/366 [==============================] - 0s 1ms/step - loss: 847382207.1519 - mean_squared_error: 847382207.1519
Epoch 26/100
366/366 [==============================] - 0s 1ms/step - loss: 738223954.9646 - mean_squared_error: 738223954.9646
Epoch 27/100
366/366 [==============================] - 0s 921us/step - loss: 812650158.3852 - mean_squared_error: 812650158.3852
Epoch 28/100
366/366 [==============================] - 0s 1ms/step - loss: 488211690.7466 - mean_squared_error: 488211690.7466
Epoch 29/100
366/366 [==============================] - 0s 959us/step - loss: 1173729588.6196 - mean_squared_error: 1173729588.6196
Epoch 30/100
366/366 [==============================] - 0s 992us/step - loss: 1611139948.6431 - mean_squared_error: 1611139948.6431
Epoch 31/100
366/366 [==============================] - 0s 996us/step - loss: 598515927.3869 - mean_squared_error: 598515927.3869
Epoch 32/100
366/366 [==============================] - 0s 954us/step - loss: 363212669.3474 - mean_squared_error: 363212669.3474
Epoch 33/100
366/366 [==============================] - 0s 964us/step - loss: 977488116.7738 - mean_squared_error: 977488116.7738
```

```
Epoch 34/100
366/366 [==============================] - 0s 1ms/step - loss: 569122776.1090 - mean_squared_error: 569122776.1090
Epoch 35/100
366/366 [==============================] - 0s 1ms/step - loss: 737600101.0232 - mean_squared_error: 737600101.0232
Epoch 36/100
366/366 [==============================] - 0s 987us/step - loss: 468048275.3678 - mean_squared_error: 468048275.3678
Epoch 37/100
366/366 [==============================] - 0s 951us/step - loss: 1258881131.0354 - mean_squared_error: 1258881131.0354
Epoch 38/100
366/366 [==============================] - 0s 1ms/step - loss: 796349626.1431 - mean_squared_error: 796349626.1431
Epoch 39/100
366/366 [==============================] - 0s 960us/step - loss: 1812555960.3270 - mean_squared_error: 1812555960.3270
Epoch 40/100
366/366 [==============================] - 0s 996us/step - loss: 386320909.5490 - mean_squared_error: 386320909.5490
Epoch 41/100
366/366 [==============================] - 0s 956us/step - loss: 746808678.7943 - mean_squared_error: 746808678.7943
Epoch 42/100
366/366 [==============================] - 0s 985us/step - loss: 932800315.6662 - mean_squared_error: 932800315.6662
Epoch 43/100
366/366 [==============================] - 0s 986us/step - loss: 688097976.8719 - mean_squared_error: 688097976.8719
Epoch 44/100
366/366 [==============================] - 0s 1ms/step - loss: 617887684.4046 - mean_squared_error: 617887684.4046
Epoch 45/100
366/366 [==============================] - 0s 1ms/step - loss: 982219261.9414 - mean_squared_error: 982219261.9414
Epoch 46/100
366/366 [==============================] - 0s 955us/step - loss: 1165807201.3161 - mean_squared_error: 1165807201.3161
Epoch 47/100
366/366 [==============================] - 0s 1ms/step - loss: 1113356810.1989 - mean_squared_error: 1113356810.1989
Epoch 48/100
366/366 [==============================] - 0s 966us/step - loss: 797795102.8229 - mean_squared_error: 797795102.8229
Epoch 49/100
366/366 [==============================] - 0s 1ms/step - loss: 1131739096.4292 - mean_squared_error: 1131739096.4292
Epoch 50/100
366/366 [==============================] - 0s 1ms/step - loss: 811985718.1907 - mean_squared_error: 811985718.1907
Epoch 51/100
366/366 [==============================] - 0s 1ms/step - loss: 605468188.6485 - mean_squared_error: 605468188.6485
Epoch 52/100
366/366 [==============================] - 0s 1ms/step - loss: 1249707812.8733 - mean_squared_error: 1249707812.8733
Epoch 53/100
366/366 [==============================] - 0s 1ms/step - loss: 725363425.9959 - mean_squared_error: 725363425.9959
Epoch 54/100
366/366 [==============================] - 0s 921us/step - loss: 638884470.8174 - mean_squared_error: 638884470.8174
```

```
Epoch 55/100
366/366 [==============================] - 0s 961us/step - loss: 545977076.8774 - mean_squared_error: 545977076.8774
Epoch 56/100
366/366 [==============================] - 0s 984us/step - loss: 910450082.7561 - mean_squared_error: 910450082.7561
Epoch 57/100
366/366 [==============================] - 0s 982us/step - loss: 989222513.1553 - mean_squared_error: 989222513.1553
Epoch 58/100
366/366 [==============================] - 0s 991us/step - loss: 452498758.7629 - mean_squared_error: 452498758.7629
Epoch 59/100
366/366 [==============================] - 0s 1ms/step - loss: 804936829.7330 - mean_squared_error: 804936829.7330
Epoch 60/100
366/366 [==============================] - 0s 1ms/step - loss: 395076323.9435 - mean_squared_error: 395076323.9435
Epoch 61/100
366/366 [==============================] - 0s 1ms/step - loss: 923249197.3515 - mean_squared_error: 923249197.3515
Epoch 62/100
366/366 [==============================] - 0s 986us/step - loss: 504851548.4448 - mean_squared_error: 504851548.4448
Epoch 63/100
366/366 [==============================] - 0s 989us/step - loss: 479592494.8134 - mean_squared_error: 479592494.8134
Epoch 64/100
366/366 [==============================] - 0s 965us/step - loss: 704535451.0674 - mean_squared_error: 704535451.0674
Epoch 65/100
366/366 [==============================] - 0s 1ms/step - loss: 1044386611.5000 - mean_squared_error: 1044386611.5000
Epoch 66/100
366/366 [==============================] - 0s 945us/step - loss: 544004539.8685 - mean_squared_error: 544004539.8685
Epoch 67/100
366/366 [==============================] - 0s 1ms/step - loss: 395188105.1403 - mean_squared_error: 395188105.1403
Epoch 68/100
366/366 [==============================] - 0s 1ms/step - loss: 534050981.0572 - mean_squared_error: 534050981.0572
Epoch 69/100
366/366 [==============================] - 0s 1ms/step - loss: 483199330.7411 - mean_squared_error: 483199330.7411
Epoch 70/100
366/366 [==============================] - 0s 1ms/step - loss: 1139259874.1267 - mean_squared_error: 1139259874.1267
Epoch 71/100
366/366 [==============================] - 0s 1ms/step - loss: 671393806.7357 - mean_squared_error: 671393806.7357
Epoch 72/100
366/366 [==============================] - 0s 1ms/step - loss: 565650317.4114 - mean_squared_error: 565650317.4114
Epoch 73/100
366/366 [==============================] - 0s 1ms/step - loss: 695331619.5967 - mean_squared_error: 695331619.5967
Epoch 74/100
366/366 [==============================] - 0s 1ms/step - loss: 617950373.1662 - mean_squared_error: 617950373.1662
Epoch 75/100
366/366 [==============================] - 0s 1ms/step - loss: 543015322.0129 - mean_squared_error: 543015322.0129
Epoch 76/100
366/366 [==============================] - 0s 1ms/step - loss: 562785632.9741 - mean_squared_error: 562785632.9741
```

```
Epoch 77/100
366/366 [==============================] - 0s 984us/step - loss: 489405410.3324 - mean_squared_error: 489405410.3324
Epoch 78/100
366/366 [==============================] - 0s 1ms/step - loss: 1141145029.1948 - mean_squared_error: 1141145029.1948
Epoch 79/100
366/366 [==============================] - 0s 981us/step - loss: 448873860.3045 - mean_squared_error: 448873860.3045
Epoch 80/100
366/366 [==============================] - 0s 1ms/step - loss: 551809568.1737 - mean_squared_error: 551809568.1737
Epoch 81/100
366/366 [==============================] - 0s 1ms/step - loss: 865533840.0899 - mean_squared_error: 865533840.0899
Epoch 82/100
366/366 [==============================] - 0s 1ms/step - loss: 557034259.4687 - mean_squared_error: 557034259.4687
Epoch 83/100
366/366 [==============================] - 0s 1ms/step - loss: 586631298.7868 - mean_squared_error: 586631298.7868
Epoch 84/100
366/366 [==============================] - 0s 1ms/step - loss: 491762905.0525 - mean_squared_error: 491762905.0525
Epoch 85/100
366/366 [==============================] - 0s 1ms/step - loss: 702715349.9857 - mean_squared_error: 702715349.9857
Epoch 86/100
366/366 [==============================] - 0s 1ms/step - loss: 708843149.0327 - mean_squared_error: 708843149.0327
Epoch 87/100
366/366 [==============================] - 0s 1ms/step - loss: 421068681.3283 - mean_squared_error: 421068681.3283
Epoch 88/100
366/366 [==============================] - 0s 1ms/step - loss: 1548456990.3971 - mean_squared_error: 1548456990.3971
Epoch 89/100
366/366 [==============================] - 0s 1ms/step - loss: 764566685.4530 - mean_squared_error: 764566685.4530
Epoch 90/100
366/366 [==============================] - 0s 1ms/step - loss: 234536564.5238 - mean_squared_error: 234536564.5238
Epoch 91/100
366/366 [==============================] - 0s 1ms/step - loss: 676000595.1717 - mean_squared_error: 676000595.1717
Epoch 92/100
366/366 [==============================] - 0s 1ms/step - loss: 561079088.6049 - mean_squared_error: 561079088.6049
Epoch 93/100
366/366 [==============================] - 0s 1ms/step - loss: 1026387713.3597 - mean_squared_error: 1026387713.3597
Epoch 94/100
366/366 [==============================] - 0s 1ms/step - loss: 1227671829.2752 - mean_squared_error: 1227671829.2752
Epoch 95/100
366/366 [==============================] - 0s 1ms/step - loss: 724290412.8161 - mean_squared_error: 724290412.8161
Epoch 96/100
366/366 [==============================] - 0s 1ms/step - loss: 1120691016.2916 - mean_squared_error: 1120691016.2916
Epoch 97/100
366/366 [==============================] - 0s 1ms/step - loss: 859769490.1771 - mean_squared_error: 859769490.1771
Epoch 98/100
366/366 [==============================] - 0s 1ms/step - loss: 1146916211.4537 - mean_squared_error: 1146916211.4537
Epoch 99/100
366/366 [==============================] - 0s 1ms/step - loss: 698116653.8747 - mean_squared_error: 698116653.8747
Epoch 100/100
366/366 [==============================] - 0s 957us/step - loss: 1597935968.0872 - mean_squared_error: 1597935968.0872
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 48
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 7
=================================================================
Total params: 97
Trainable params: 97
Non-trainable params: 0
_________________________________________________________________
Mean Absolute Error: 3213.11569803204
Mean Squared Error: 829403936.3214768
Root Mean Squared Error: 28799.37388766424
```

In [15]:

```
#Saving Scikitlearn models
import joblib
joblib.dump(decision_tree, "decisiontree_youtubeadview.pkl")

# Saving Keras Artificial Neural Network model
ann.save("ann_youtubeadview.h5")
```

Testing

In [16]:

```
data_test = pd.read_csv("/content/test.csv")
```

In [17]:

```
data_test.head()
```

Out[17]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	VID_1054	440238	6153	218	1377	2017-02-18	PT7M29S	B
1	VID_18629	1040132	8171	340	1047	2016-06-28	PT6M29S	F
2	VID_13967	28534	31	11	1	2014-03-10	PT37M54S	D
3	VID_19442	1316715	2284	250	274	2010-06-05	PT9M55S	G
4	VID_770	1893173	2519	225	116	2016-09-03	PT3M8S	B

In [18]:
```
from keras.models import load_model
model = load_model("/content/ann_youtubeadview.h5")
```

In [19]:
```
# Removing character "F" present in data
data_test=data_test[data_test.views!='F']
data_test=data_test[data_test.likes!='F']
data_test=data_test[data_test.dislikes!='F']
data_test=data_test[data_test.comment!='F']
```

In [20]:
```
data_test.head()
```

Out[20]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	VID_1054	440238	6153	218	1377	2017-02-18	PT7M29S	B
1	VID_18629	1040132	8171	340	1047	2016-06-28	PT6M29S	F
2	VID_13967	28534	31	11	1	2014-03-10	PT37M54S	D
3	VID_19442	1316715	2284	250	274	2010-06-05	PT9M55S	G
4	VID_770	1893173	2519	225	116	2016-09-03	PT3M8S	B

In [21]:
```
# Assigning each category a number for Category feature
category={'A': 1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7,'H':8}
data_test["category"]=data_test["category"].map(category)
data_test.head()
```

Out[21]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	VID_1054	440238	6153	218	1377	2017-02-18	PT7M29S	2
1	VID_18629	1040132	8171	340	1047	2016-06-28	PT6M29S	6
2	VID_13967	28534	31	11	1	2014-03-10	PT37M54S	4
3	VID_19442	1316715	2284	250	274	2010-06-05	PT9M55S	7
4	VID_770	1893173	2519	225	116	2016-09-03	PT3M8S	2

In [22]:
```
# Convert values to integers for views, likes, comments, dislikes and adview
data_test["views"] = pd.to_numeric(data_test["views"])
data_test["comment"] = pd.to_numeric(data_test["comment"])
data_test["likes"] = pd.to_numeric(data_test["likes"])
data_test["dislikes"] = pd.to_numeric(data_test["dislikes"])
column_vidid=data_test['vidid']

# Endoding features like Category, Duration, Vidid
from sklearn.preprocessing import LabelEncoder
data_test['duration']=LabelEncoder().fit_transform(data_test['duration'])
data_test['vidid']=LabelEncoder().fit_transform(data_test['vidid'])
data_test['published']=LabelEncoder().fit_transform(data_test['published'])
data_test.head()
```

Out[22]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	231	440238	6153	218	1377	2053	2115	2
1	3444	1040132	8171	340	1047	1825	2055	6
2	1593	28534	31	11	1	1009	1506	4
3	3775	1316715	2284	250	274	116	2265	7
4	7644	1893173	2519	225	116	1892	1625	2

In [23]:

```
# Convert Time_in_sec for duration
import datetime
import time
def checki(x):
  y = x[2:]
  h = ''
  m = ''
  s = ''
  mm = ''
  P = ['H','M','S']
  for i in y:
    if i not in P:
      mm+=i
    else:
      if(i=="H"):
        h = mm
        mm = ''
      elif(i == "M"):
        m = mm
        mm = ''
      else:
        s = mm
        mm = ''
  if(h==''):
    h = '00'
  if(m == ''):
    m = '00'
  if(s==''):
    s='00'
  bp = h+':'+m+':'+s
  return bp

train=pd.read_csv("test.csv")
mp = pd.read_csv(path)["duration"]
time = mp.apply(checki)

def func_sec(time_string):
  h, m, s = time_string.split(':')
  return int(h) * 3600 + int(m) * 60 + int(s)

time1=time.apply(func_sec)

data_test["duration"]=time1
```

Out[23]:

	vidid	views	likes	dislikes	comment	published	duration	category
0	231	440238	6153	218	1377	2053	457	2
1	3444	1040132	8171	340	1047	1825	570	6
2	1593	28534	31	11	1	1009	136	4
3	3775	1316715	2284	250	274	116	262	7
4	7644	1893173	2519	225	116	1892	31	2

In [24]:
```
data_test=data_test.drop(["vidid"],axis=1)
data_test.head()
```

Out[24]:

	views	likes	dislikes	comment	published	duration	category
0	440238	6153	218	1377	2053	457	2
1	1040132	8171	340	1047	1825	570	6
2	28534	31	11	1	1009	136	4
3	1316715	2284	250	274	116	262	7
4	1893173	2519	225	116	1892	31	2

In [25]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_test = data_test
X_test=scaler.fit_transform(X_test)
```

In [26]:
```
prediction = model.predict(X_test)
```

In [27]:
```
prediction=pd.DataFrame(prediction)
prediction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8549 entries, 0 to 8548
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       8549 non-null   float32
dtypes: float32(1)
memory usage: 33.5 KB
```

In [28]:
```
prediction = prediction.rename(columns={0: "Adview"})
```

In [29]:
```
prediction.head()
```

Out[29]:

	Adview
0	2167.144043
1	2330.150879
2	1572.538940
3	1154.258423
4	2101.321289

In [30]:
```
prediction.to_csv('predictions.csv')
```