TikTok Data Analysis

Introduction:- This case is about a tiktok data made by the author. Majority of the tiktokers preferred to make the video of the duration 15 seconds and people preffered to share the video mostly. And we saw that tiktokers who verify their account also has used the original music get the more comments and like. Also reviewer preffered to like and comment the video on short video as compare to long duration video.

Objective:- Build a model that will help to identify the video duration preffered on the basis of comments, like, shares etc

Import Libraries

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

Read the Dataset

In [3]:
```
d=pd.read_csv(r'C:\Users\lxm\Desktop\trending.csv')
d
```

Out[3]:

	id	text	createTime	authorMeta/id	authorname	authornickName	author
0	6.907230e+18	Confidence went 📈	1608214517	6.825540e+18	ninakleij	Nina	
1	6.875470e+18	Quiet Zone... follow me on insta: joeysofo. Co...	1600819763	6.729290e+18	joeysofo	JoeySofo	
2	6.898700e+18	Iphone bend test😷 #tiktok #viral #fyp #iphone ...	1606228625	6.791900e+18	jackeyephone	JackJacko	
3	6.902820e+18	NaN	1607187987	6.574080e+18	naomivaneeren	Naomi van eeren ✨	
4	6.905640e+18	小技です👟✨#tiktok教室#tutorial	1607843600	6.586850e+18	io.dreamer_mk	io. Dreamer	
...	...	...	...	...	...	...	
995	6.877190e+18	#foryou #foryoupage	1601220970	6.788450e+18	artistmiranda	ArtistMiranda	
996	6.908070e+18	Stop eating 💩 #gttfg #gotothegym #swolefam #nu...	1608410366	6.718790e+18	papaswolio	Papa Swolio	
997	6.883480e+18	#fy #foryoupage #foryou	1602686079	6.792310e+18	sanaelfarah	Sana El Farah	
998	6.898720e+18	regretss 📈 #fyp #foryou #curls	1606233872	6.957010e+16	safae.kx	Safae	
999	6.899120e+18	The collab you didn't know you needed, myself ...	1606325682	6.798140e+18	erinwilliams_1	Erin Williams	

1000 rows × 17 columns

Drop the unwanted columns

In [3]:
```
for col in d.columns:
    #print(d1[col].dtypes)
    if d[col].dtypes=='object':
        d.drop(col,inplace=True, axis=1)
```

In [4]:
```
d.shape
```

Out[4]:
```
(1000, 11)
```

In [5]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              1000 non-null   float64
 1   createTime      1000 non-null   int64  
 2   authorMeta/id   1000 non-null   float64
 3   authorverified  1000 non-null   bool   
 4   musicId         1000 non-null   float64
 5   musicoriginal   1000 non-null   bool   
 6   videoduration   1000 non-null   int64  
 7   videolike       1000 non-null   int64  
 8   shareCount      1000 non-null   int64  
 9   playCount       1000 non-null   int64  
 10  commentCount    1000 non-null   int64  
dtypes: bool(2), float64(3), int64(6)
memory usage: 72.4 KB
```

In [6]:
```
d1=d
```

In [7]:
```
for col in d1.columns:
    if d[col].dtypes=='bool':
        d.drop(col,inplace=True,axis=1)
```

In [8]:
```
d1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             1000 non-null   float64
 1   createTime     1000 non-null   int64  
 2   authorMeta/id  1000 non-null   float64
 3   musicId        1000 non-null   float64
 4   videoduration  1000 non-null   int64  
 5   videolike      1000 non-null   int64  
 6   shareCount     1000 non-null   int64  
 7   playCount      1000 non-null   int64  
 8   commentCount   1000 non-null   int64  
dtypes: float64(3), int64(6)
memory usage: 70.4 KB
```

From the above Data Dictionary we observe that there are 8 columns listed in the dataset.

Y = Video Dration will be our Dependent/Target variable

X = Remaining features will be considered as Independent variables

In [10]:
```
x=d1.drop('videoduration',axis=1)
y=d1['videoduration']
```

StandardDise the dataset

In [26]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_transformed=sc.fit_transform(x_train)
```

In [27]:
```
x_train_transformed
```

Out[27]:
```
array([[-0.71253678, -0.71269507,  0.42647492, ..., -0.1263003 ,
        -0.10675005, -0.04717181],
       [ 1.42286536,  1.42315142,  0.38546399, ..., -0.16103025,
        -0.11207853, -0.05968028],
       [-1.78428217, -1.78442714,  0.35936037, ..., -0.02107492,
        -0.03561333, -0.02645466],
       ...,
       [ 0.338987  ,  0.33943531,  0.31455247, ..., -0.14828532,
         0.59895982, -0.05724808],
       [-0.1705976 , -0.17011485,  0.37612769, ..., -0.15529503,
        -0.11260074, -0.06102668],
       [-0.7165811 , -0.71619123,  0.37307292, ..., -0.14151457,
        -0.10088968, -0.03092818]])
```

In [28]:
```
x_test_transformed=sc.transform(x_test)
```

In [29]:
```
x_test_transformed
```

Out[29]:
```
array([[ 4.12795882e-01,  4.12839490e-01,  3.76643610e-01, ...,
        -8.14540560e-02, -1.01073423e-01, -3.11887692e-02],
       [ 6.83765472e-01,  6.84171722e-01,  2.97364266e-01, ...,
        -7.04615485e-02,  6.70784530e-02,  3.69997642e-02],
       [ 2.75013912e-04,  2.58618353e-04,  4.24461030e-01, ...,
        -1.65172356e-01, -1.16557553e-01, -6.69334590e-02],
       ...,
       [ 6.68599263e-01,  6.68341942e-01,  3.89641115e-01, ...,
        -1.65809603e-01, -1.15981864e-01, -6.61082475e-02],
       [-1.40512696e+00, -1.40481467e+00,  3.05600846e-01, ...,
        -1.65172356e-01, -1.14254410e-01, -6.45881209e-02],
       [-1.44455910e+00, -1.44409925e+00,  4.04964774e-01, ...,
        -1.63738551e-01, -1.13751541e-01, -6.55001969e-02]])
```

Fit Linear Regression into train test data

In [30]:
```
lr.fit(x_train_transformed,y_train)
```

Out[30]:
```
LinearRegression()
```

In [31]:
```
print("Test Score",lr.score(x_test_transformed,y_test))
```

```
Test Score 0.004584399356110658
```

In [32]:
```
print("Training Score",lr.score(x_train_transformed,y_train))
```

```
Training Score 0.012463926246669699
```

In [33]:
```
pred=lr.predict(x_test_transformed)
```

In [34]:
```
pred
```

Out[34]:
```
array([19.34340829, 18.38001819, 19.45547779, 18.7275252 , 19.13704491,
       16.22195856, 19.9245826 , 18.75520253, 16.75175035, 19.32915724,
       18.9004599 , 16.62486007, 16.45380906, 19.91134284, 18.67315576,
       19.88162449, 18.55150669, 18.81064666, 19.01940455, 19.80924824,
       18.48439889, 15.60456524, 20.30342456, 12.93378539, 18.74155347,
       18.99151612, 18.54816918, 19.93271175, 19.05953119, 19.1316123 ,
       17.05011564, 19.01303663, 19.75917573, 19.00854331, 20.26868863,
       19.77885511, 18.98022613, 19.19800319, 18.95022676, 16.16870667,
       18.89572459, 18.78545503, 20.19180395, 18.99573669, 15.00123902,
       16.99532805, 19.71821068, 19.73668985, 18.97651223, 19.96247372,
       19.58645952, 19.38905864, 19.73559297, 18.59002342, 18.65501091,
       19.08135569, 18.69321869, 18.59966457, 18.85269146, 19.14950637,
       16.14195482, 19.15945159, 20.24663021, 18.96470558, 15.72613096,
       19.19138087, 20.24573449, 19.33895438, 19.06537514, 19.50815665,
       19.10710975, 19.00129228, 18.21446911, 18.58497035, 19.82353086,
       20.12531875, 18.69841595, 15.09312149, 19.56421066, 16.5893343 ,
       19.11364028, 19.97291362, 18.6613722 , 18.65358877, 20.02944273,
       19.40028301, 19.5201001 , 16.2494296 , 18.53547671, 18.48514016,
       19.13584582, 18.83155843, 19.38953291, 15.85798269, 20.0145949 ,
       19.09552099, 20.23459756, 19.82104766, 15.81878548, 19.06866223,
       18.41268939, 19.23988681, 19.86271637, 18.84615297, 19.94398819,
       17.49915114, 18.89261127, 19.02565278, 16.56367128, 19.33909716,
       19.88285795, 19.36548414, 19.35298107, 19.34096812, 18.51175083,
       18.83918091, 16.65473218, 15.35784313, 19.85342213, 18.80568656,
       19.72682487, 18.61022469, 19.74051792, 16.58452755, 19.09325496,
       20.02408575, 20.17255343, 18.47489545, 20.19664814, 19.32981159,
       20.073388  , 20.13049722, 16.52807919, 19.45526902, 19.42140836,
       18.80851085, 18.8141349 , 19.06372243, 18.95859238, 20.28383789,
       20.06282712, 19.20190924, 20.04232665, 19.66825294, 19.01757335,
       19.84207863, 19.00083221, 19.97501896, 15.99163701, 17.18335121,
       18.89504218, 18.88347971, 19.90998507, 18.76282601, 17.04230097,
       19.35584604, 19.69423807, 19.0504653 , 18.81262279, 20.01652801,
       18.79726455, 19.72438118, 18.49204389, 18.55510194, 19.54815386,
       19.55133077, 15.61136534, 20.09296309, 19.6552359 , 19.35995781,
       19.89172269, 18.98724896, 15.66130295, 18.63586615, 19.38686068,
       20.0730028 , 19.20524685, 19.13297034, 11.71074237, 19.00491751,
       20.05764435, 19.45587453, 19.6159998 , 16.92417213, 18.75725325,
       19.05208018, 15.28394972, 16.31986672, 19.91760399, 20.16059308,
       19.06103908, 18.70913247, 20.07530998, 19.44018893, 19.12849475,
       15.38681164, 19.99411969, 19.56935275, 19.715235  , 20.1444913 ,
       20.00462099, 19.94512096, 19.91651552, 19.53642946, 19.01973899,
       19.21668213, 19.52812857, 19.93255209, 18.68742541, 16.72274353,
       20.05956842, 18.97466833, 19.04073826, 18.83559772, 18.47536793,
       19.90739378, 19.29788018, 18.80911035, 19.04074472, 19.22467247,
       19.94852594, 19.35600806, 18.65089279, 19.03021267, 19.58786886,
       19.56173669, 18.86825326, 16.67971171, 16.94394901, 19.06801189,
       19.71437077, 19.19812425, 19.27037846, 18.79224462, 20.08882755,
       17.05990475, 18.6656301 , 16.35203591, 15.33781987, 19.6758351 ,
       18.76014472, 18.90402292, 18.70811483, 19.13513535, 16.12243565,
       19.6682    , 19.29638687, 19.84772113, 18.73074715, 18.57811123])
```

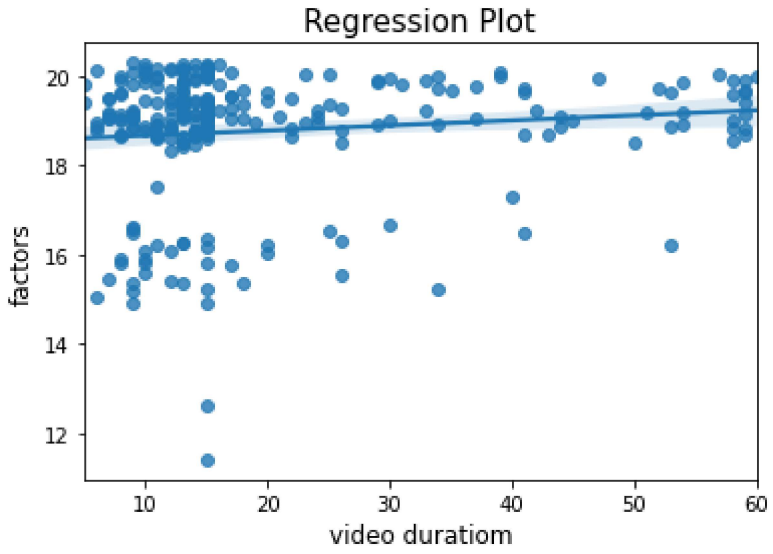
In [35]:
```
frame=pd.DataFrame({'Actual':y_test,'Predicted':pred})
frame
```

Out[35]:

	Actual	Predicted
539	18	19.343408
89	13	18.380018
40	15	19.455478
310	11	18.727525
695	59	19.137045
...	...	...
412	53	19.668200
339	15	19.296387
328	54	19.847721
745	59	18.730747
599	41	18.578111

250 rows × 2 columns

In [37]:
```
sns.regplot(x=y_test,y=predicted)
plt.title("Regression Plot",size=15)
plt.ylabel('factors',size=12)
plt.xlabel('video duratiom', size=12)
plt.show()
```



Conclusion

After standardize the model we are going to train and test the data and then we are fitting the linear regression into the model.

After analysing the data, machine give us the accuracy of Test Score 0.00459 and Training Score 0.0125 which is showing the lower accuracy.

In [ ]: