Intsagram Reach Prediction

Import Library as well as Dataset

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import os
%matplotlib inline
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv("Downloads/Instagram data.csv", encoding='latin-1')
```

In [3]:
```
df
```

Out[3]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	F
0	3920	2586	1028	619	56	98	9	5	162	35	
1	5394	2727	1838	1174	78	194	7	14	224	48	
2	4021	2085	1188	0	533	41	11	1	131	62	
3	4528	2700	621	932	73	172	10	7	213	23	
4	2518	1704	255	279	37	96	5	4	123	8	
...	...	...	...	...	...	...	...	...	...	...	
114	13700	5185	3041	5352	77	573	2	38	373	73	
115	5731	1923	1368	2266	65	135	4	1	148	20	
116	4139	1133	1538	1367	33	36	0	1	92	34	
117	32695	11815	3147	17414	170	1095	2	75	549	148	

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	F
118	36919	13473	4176	16444	2547	653	5	26	443	611	

119 rows × 13 columns

In [4]:
```
#Make a Copy of the Original dataset Which can help me in future
df1 = df.copy(deep=True)
df2 = df.copy(deep=True)
```

In [5]:
```
df.head()
```

Out[5]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Foll
0	3920	2586	1028	619	56	98	9	5	162	35	
1	5394	2727	1838	1174	78	194	7	14	224	48	
2	4021	2085	1188	0	533	41	11	1	131	62	
3	4528	2700	621	932	73	172	10	7	213	23	
4	2518	1704	255	279	37	96	5	4	123	8	

In [6]:
```
df.tail()
```

Out[6]:

	Impressions	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	F
114	13700	5185	3041	5352	77	573	2	38	373	73	
115	5731	1923	1368	2266	65	135	4	1	148	20	
116	4139	1133	1538	1367	33	36	0	1	92	34	
117	32695	11815	3147	17414	170	1095	2	75	549	148	
118	36919	13473	4176	16444	2547	653	5	26	443	611	

Data Preprocessing

In [7]:
```
#chaeking for the missing value
df.isnull().sum()
```

Out[7]:
```
Impressions       0
From Home         0
From Hashtags     0
From Explore      0
From Other        0
Saves             0
Comments          0
Shares            0
Likes             0
Profile Visits    0
Follows           0
Caption           0
Hashtags          0
dtype: int64
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119 entries, 0 to 118
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Impressions     119 non-null    int64 
 1   From Home       119 non-null    int64 
 2   From Hashtags   119 non-null    int64 
 3   From Explore    119 non-null    int64 
 4   From Other      119 non-null    int64 
 5   Saves           119 non-null    int64 
 6   Comments        119 non-null    int64 
 7   Shares          119 non-null    int64 
 8   Likes           119 non-null    int64 
 9   Profile Visits  119 non-null    int64 
 10  Follows         119 non-null    int64 
 11  Caption         119 non-null    object
 12  Hashtags        119 non-null    object
dtypes: int64(11), object(2)
memory usage: 12.2+ KB
```

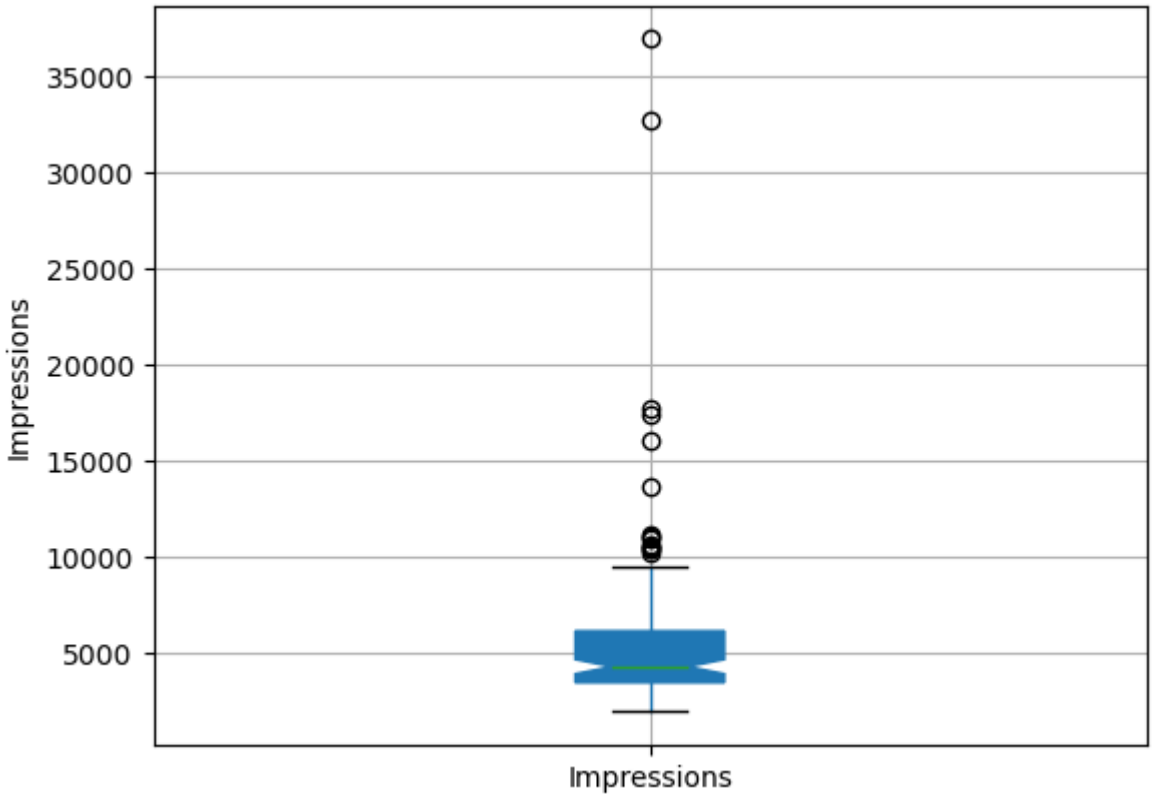
In [9]:
```
df.columns
```

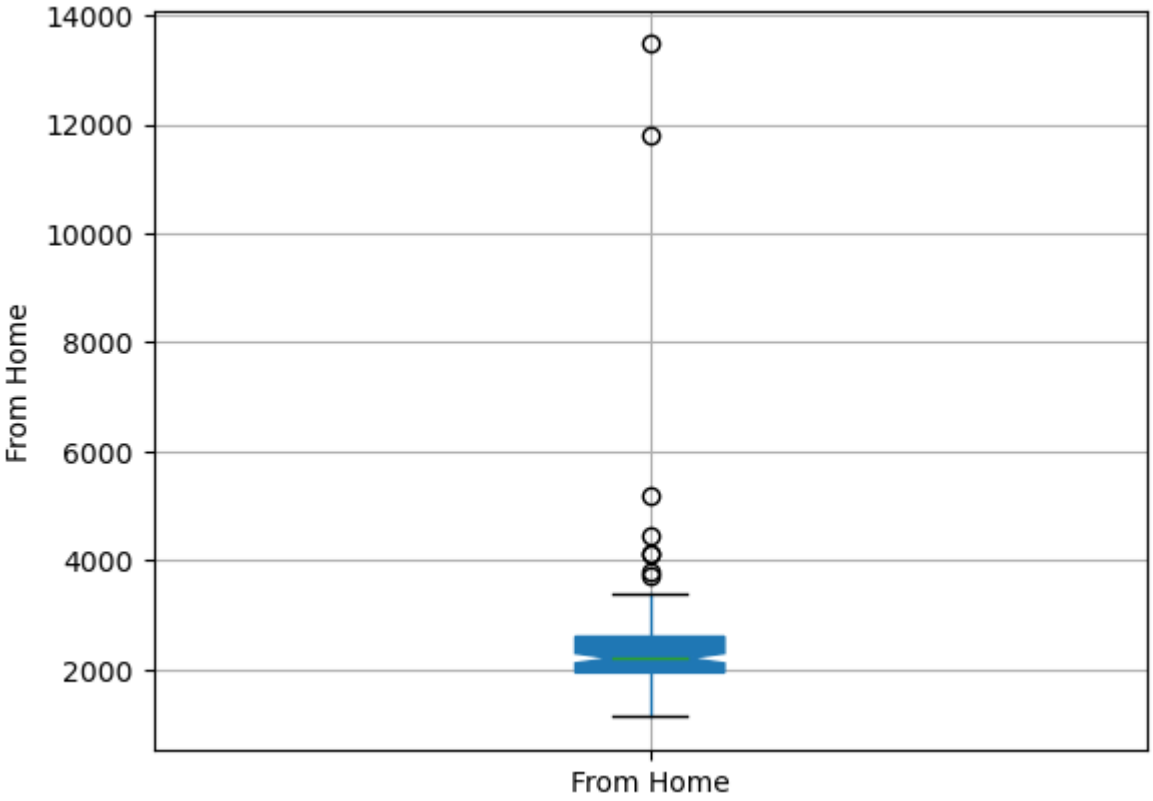
Out[9]:
```
Index(['Impressions', 'From Home', 'From Hashtags', 'From Explore',
       'From Other', 'Saves', 'Comments', 'Shares', 'Likes', 'Profile Visit
s',
       'Follows', 'Caption', 'Hashtags'],
      dtype='object')
```

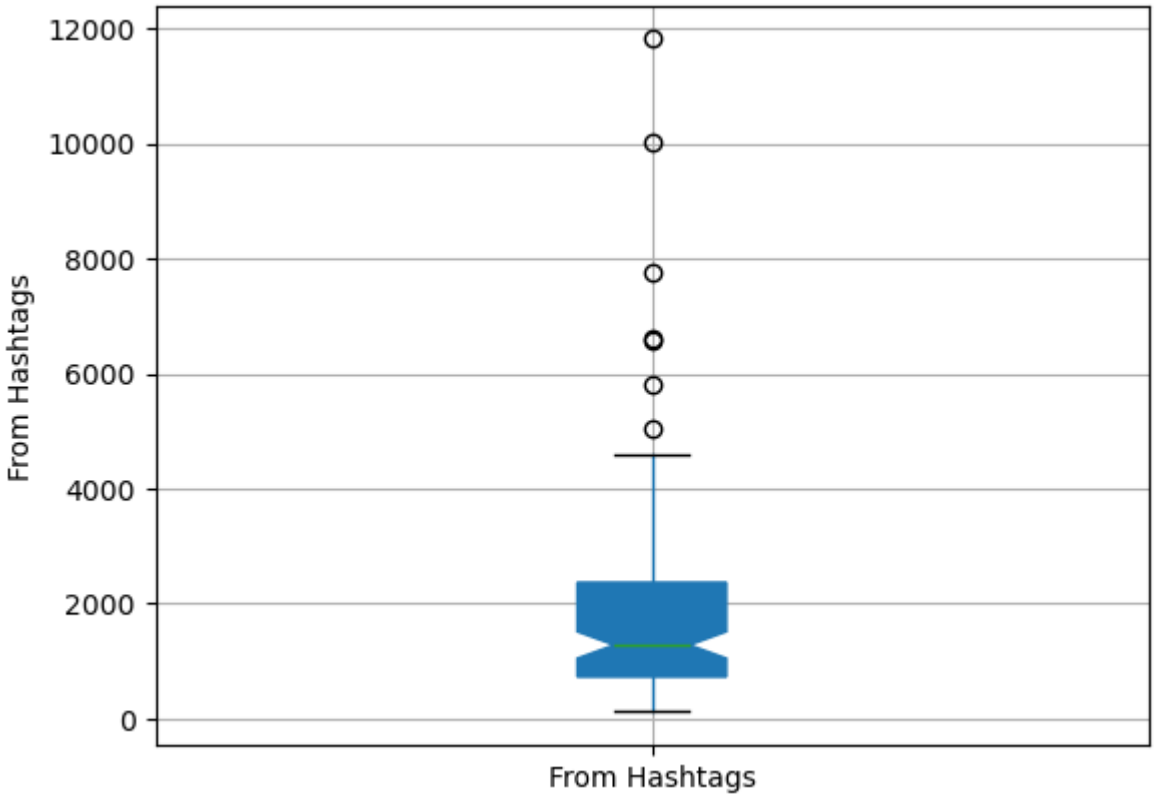
EXPLORATORY DATA ANALYSIS

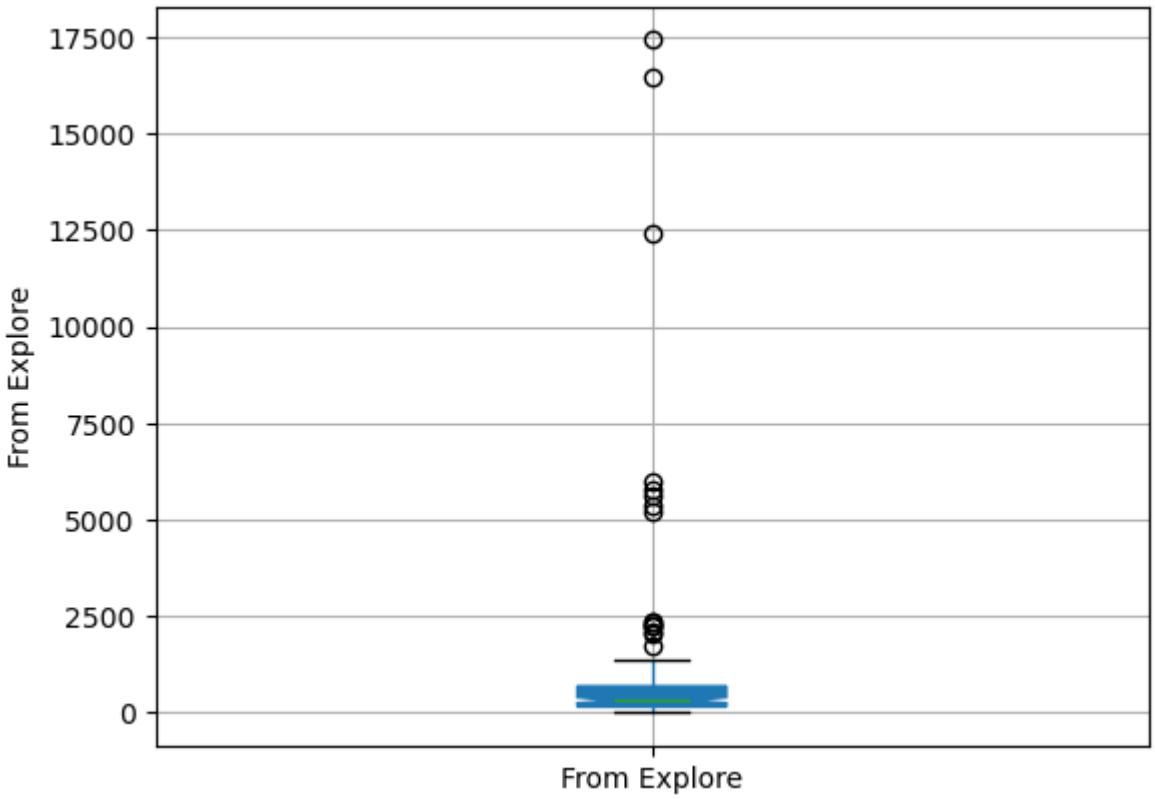
In [10]:
```
#First Of all we seperate categorical and numerical data
df_num = df.select_dtypes(include="number")
df_cat = df.select_dtypes(include="object")
```

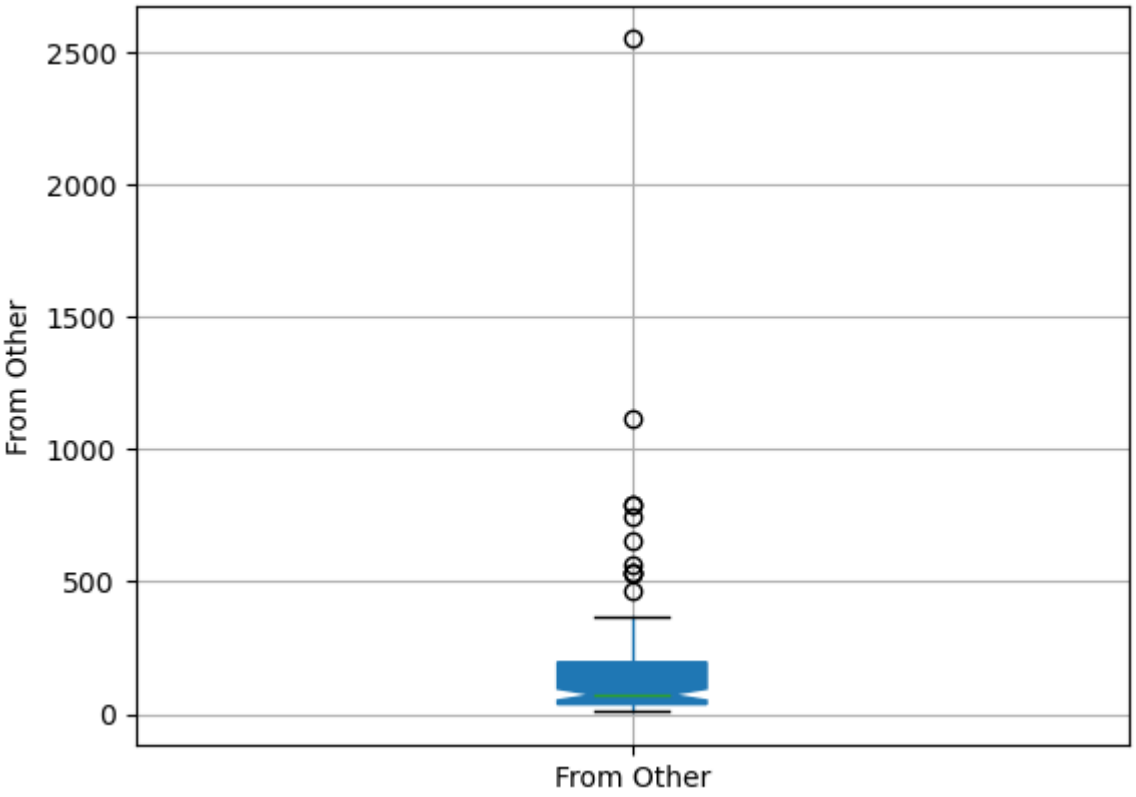
In [11]:
```
#for numerical distrubition
for i in df_num:
    df_num.boxplot(column=i,patch_artist = True, notch ='True')
    plt.ylabel(i)
    plt.show()
```

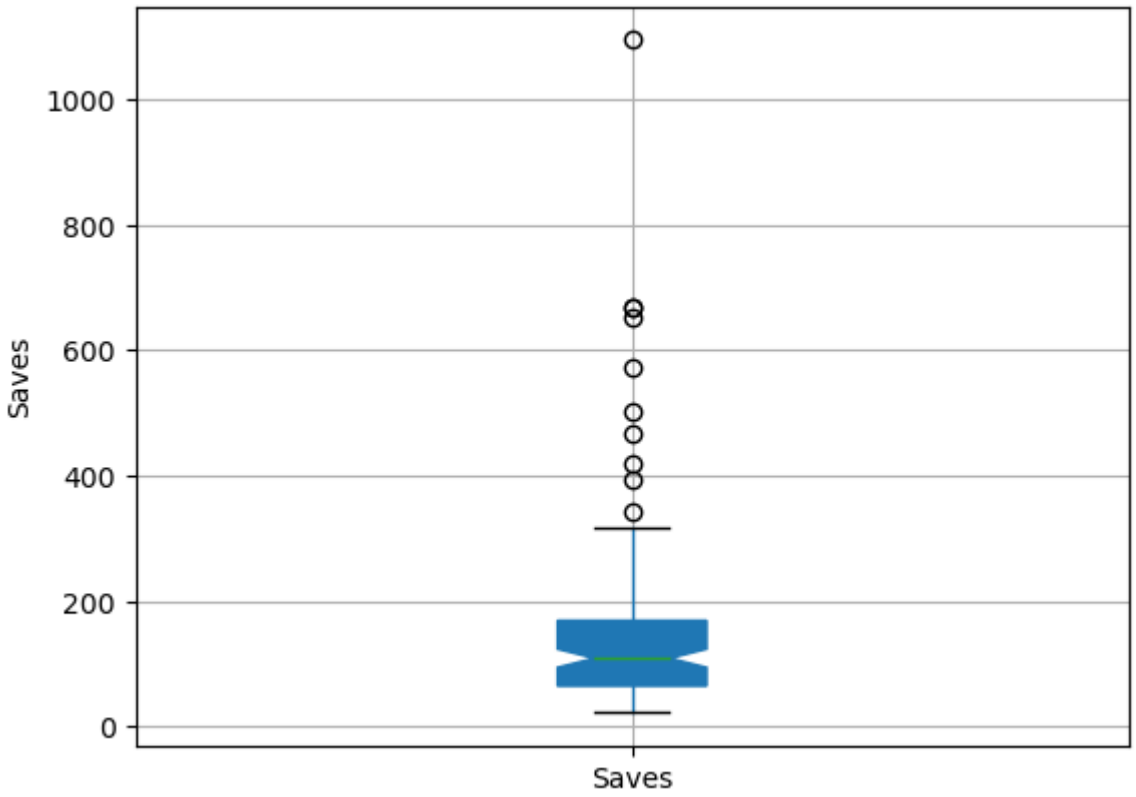

35000
30000
25000
20000
15000
10000
5000
Impressions
Impressions

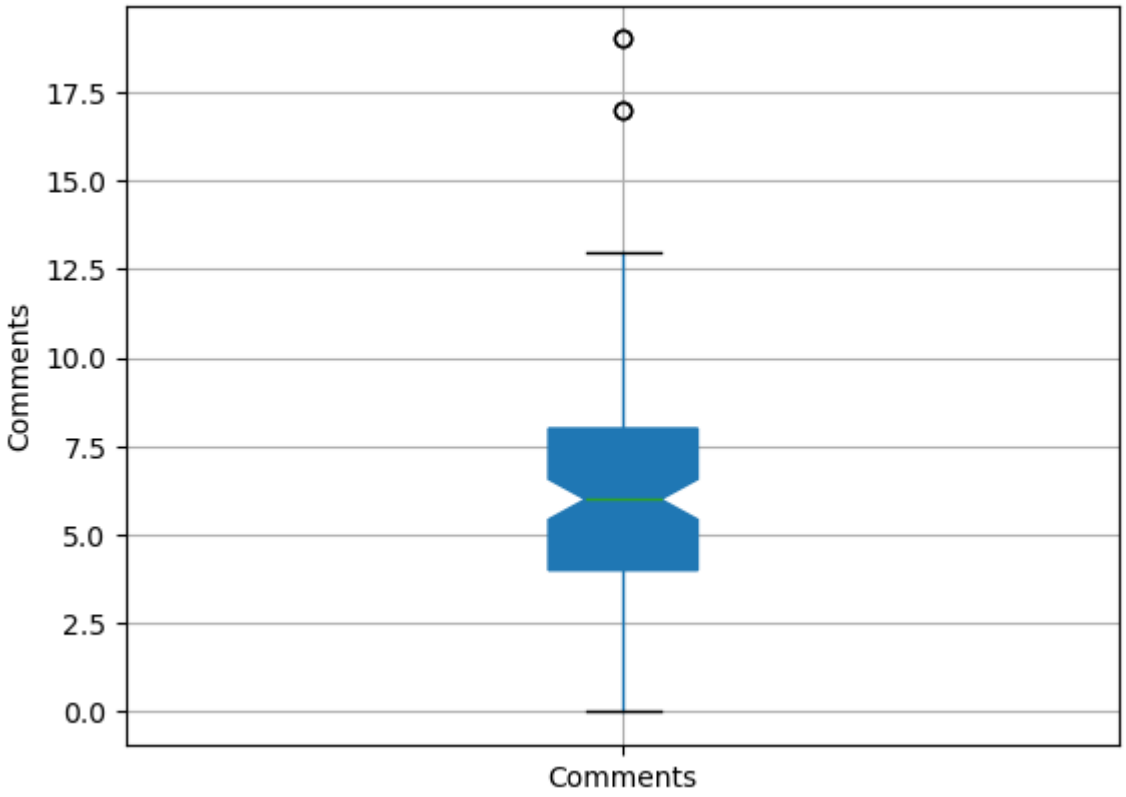

14000
12000
10000
8000
6000
4000
2000
From Home
From Home

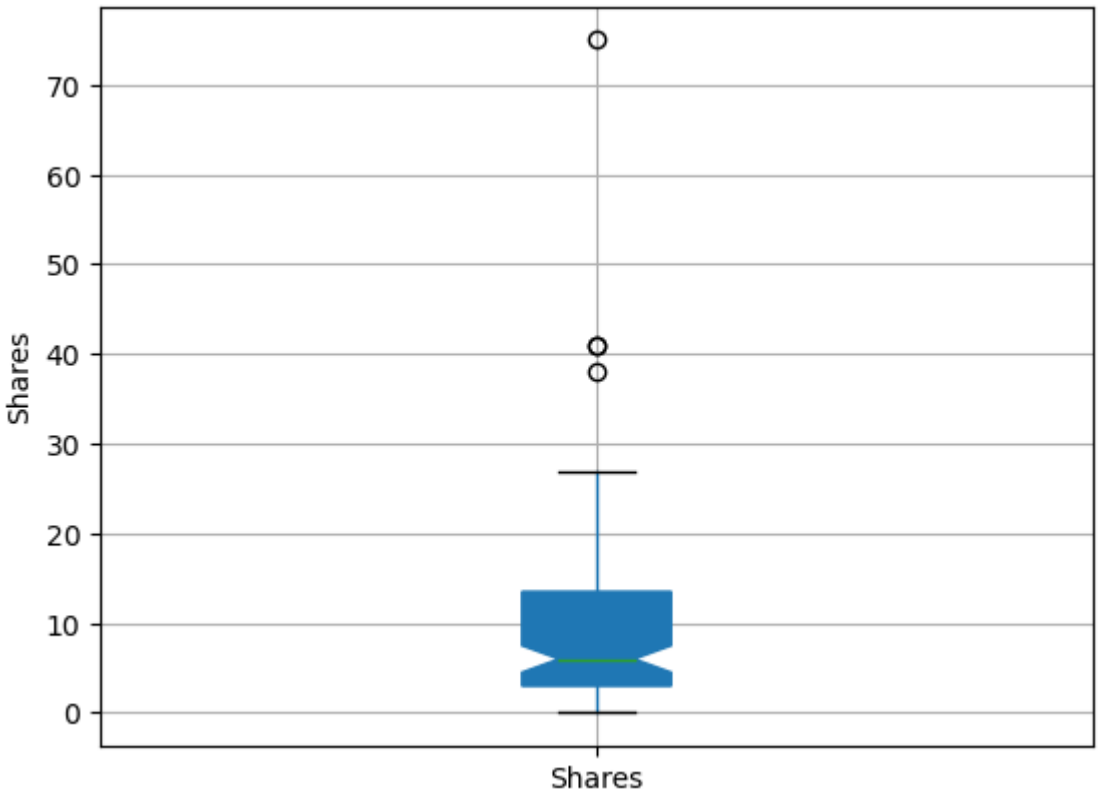

12000
10000
8000
6000
4000
2000
0
From Hashtags
From Hashtags

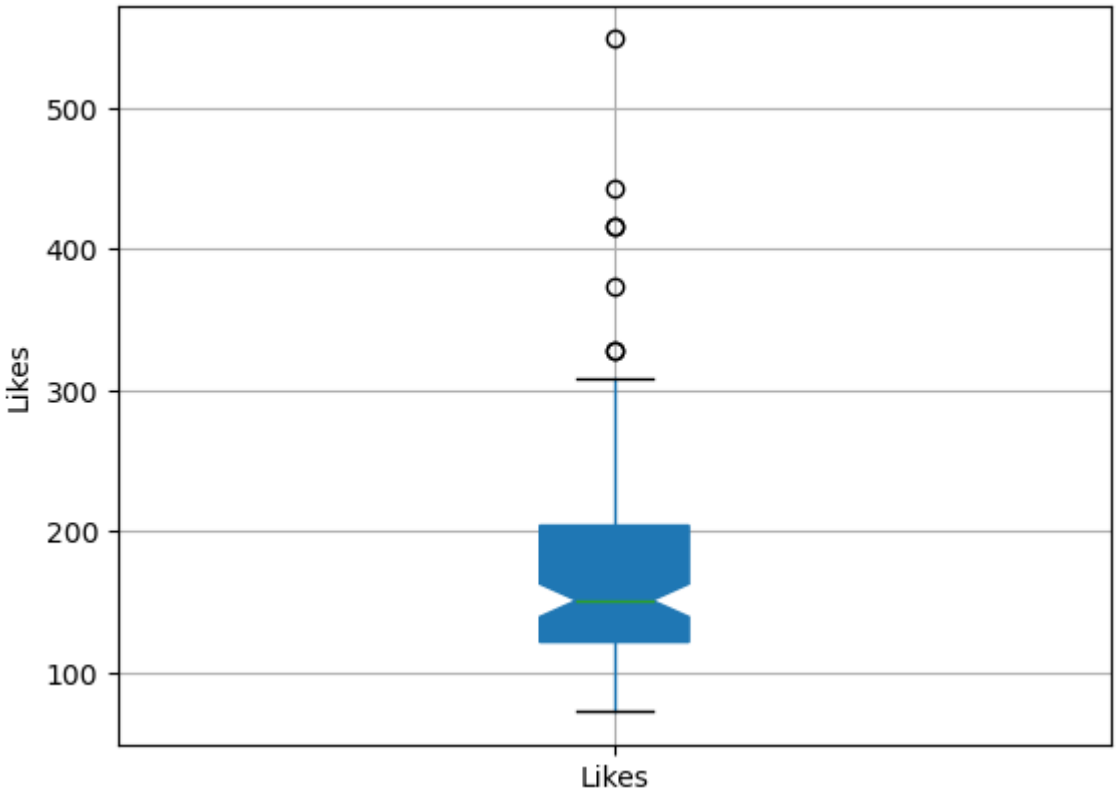

17500
15000
12500
10000
7500
5000
2500
0
From Explore
From Explore

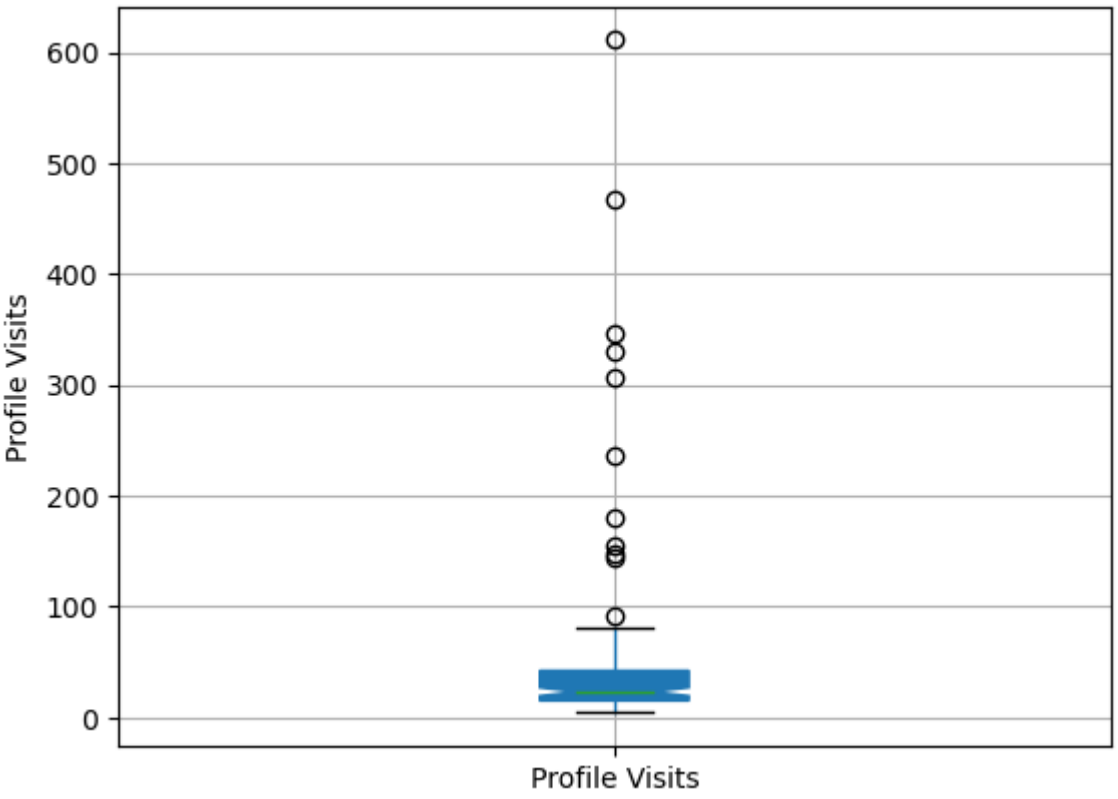

2500
2000
1500
1000
500
0
From Other
From Other

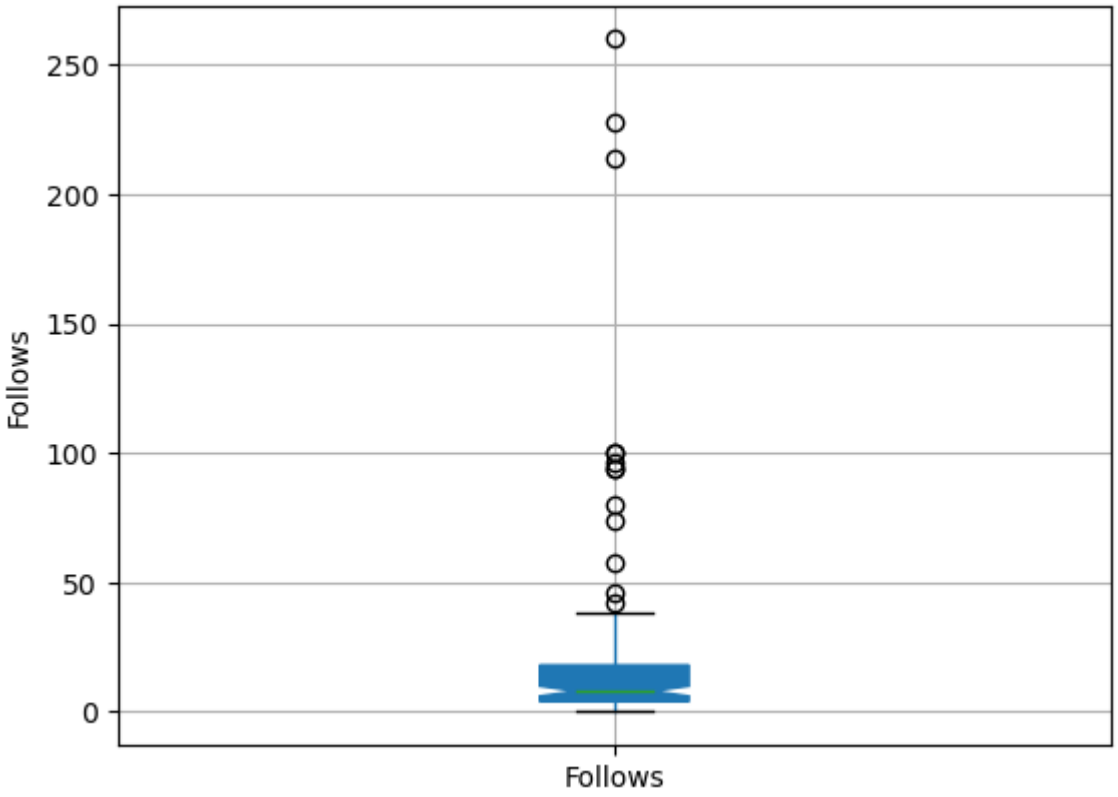

1000
800
600
400
200
0
Saves
Saves


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Comments
Comments


70
60
50
40
30
20
10
0
Shares
Shares

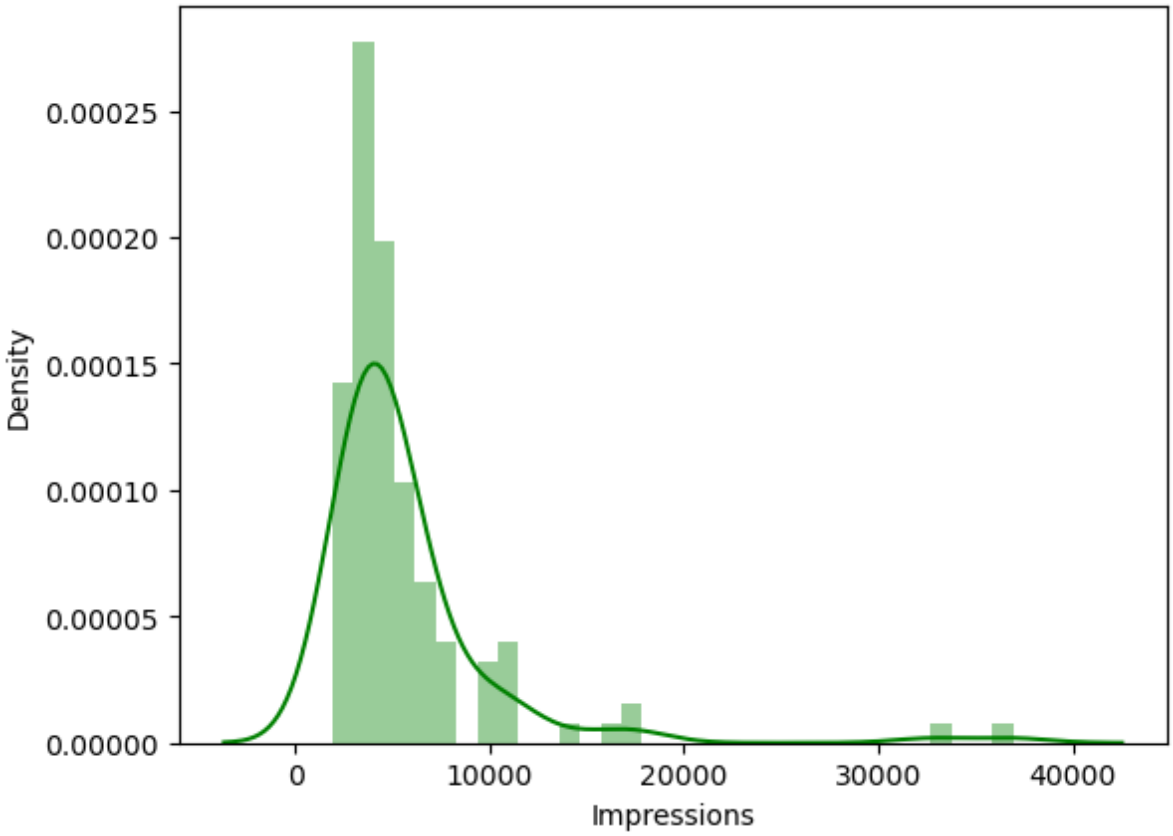

500
400
300
200
100
Likes
Likes

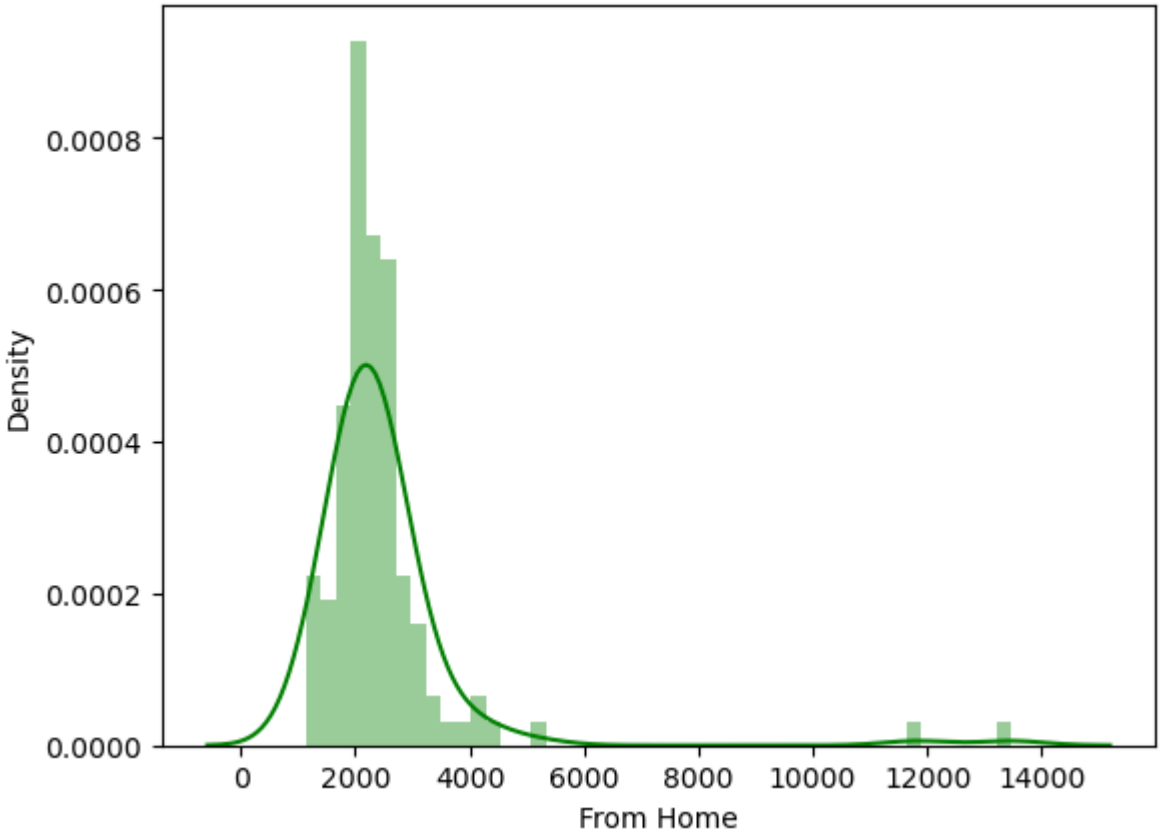

600
500
400
300
200
100
0
Profile Visits
Profile Visits

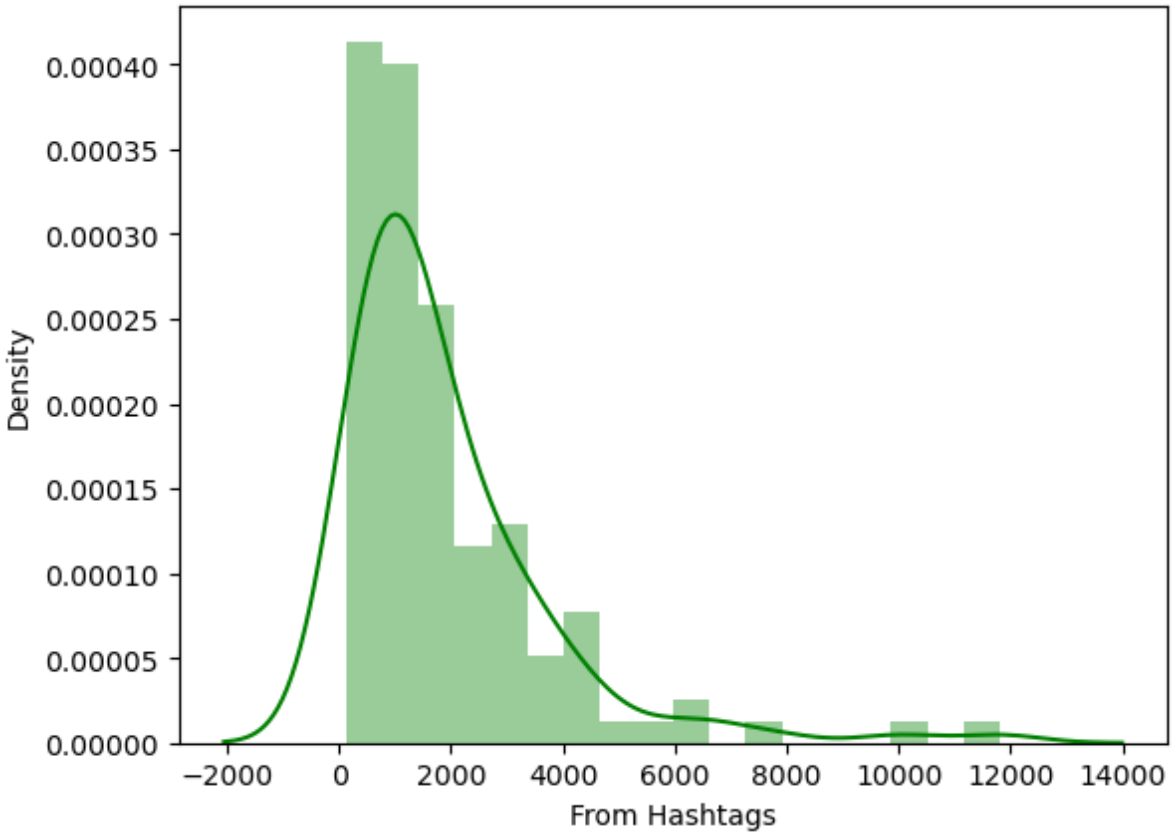


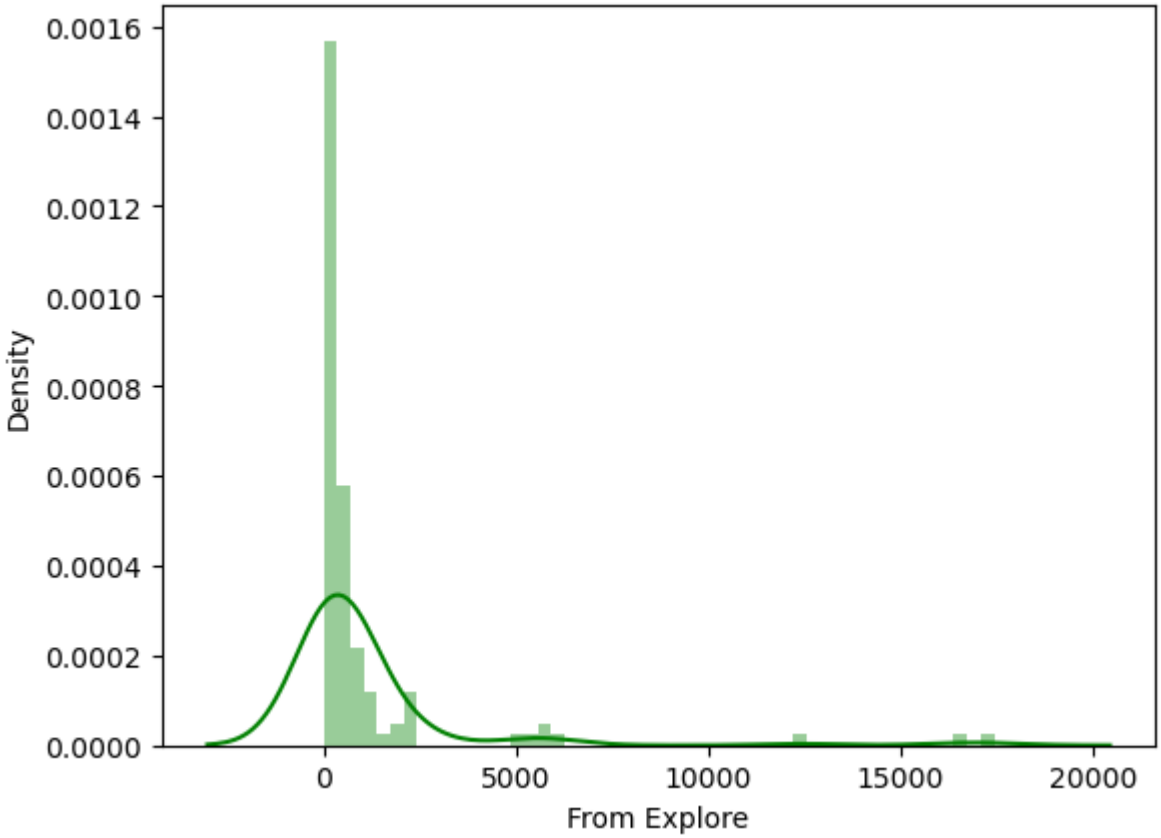
In [12]:
```
#we check distrubition of the numerical data
```

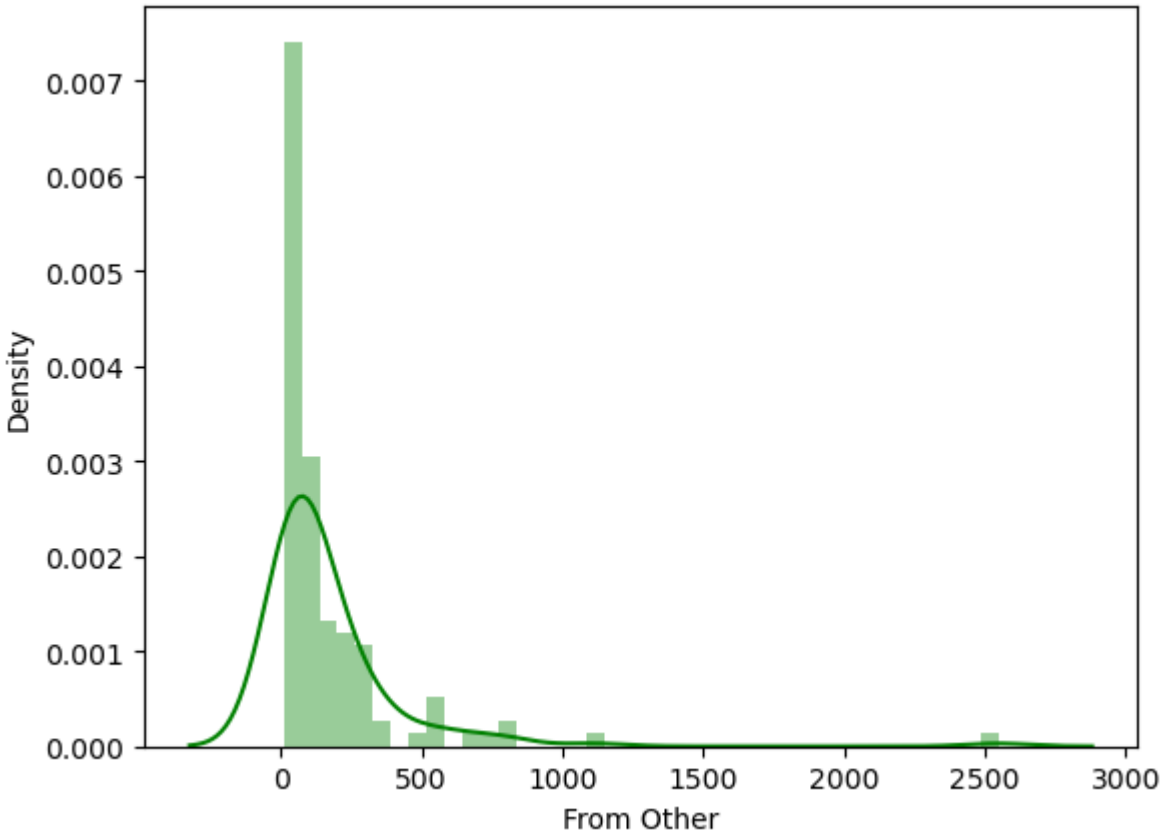
In [13]:
```
for i in df_num:
    sns.distplot(df[i], kde = True, color = 'green')
    plt.show()
```

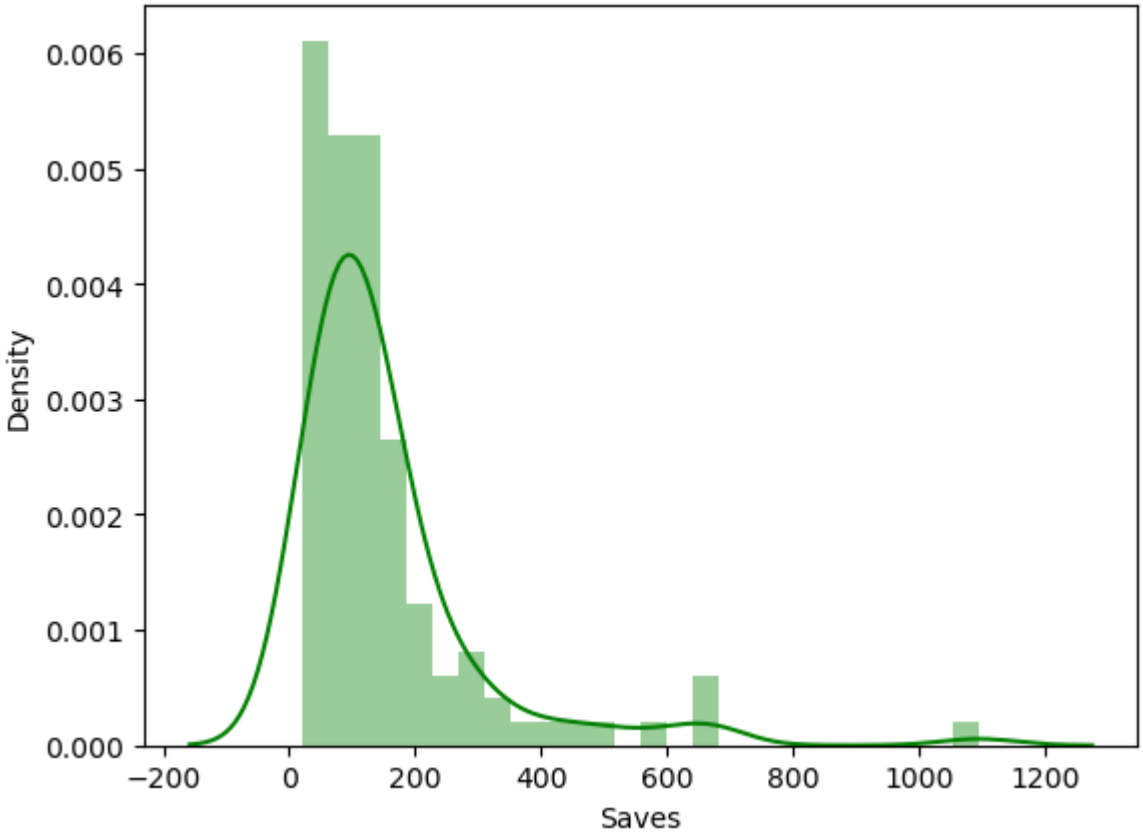

0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
10000
20000
30000
40000
Impressions

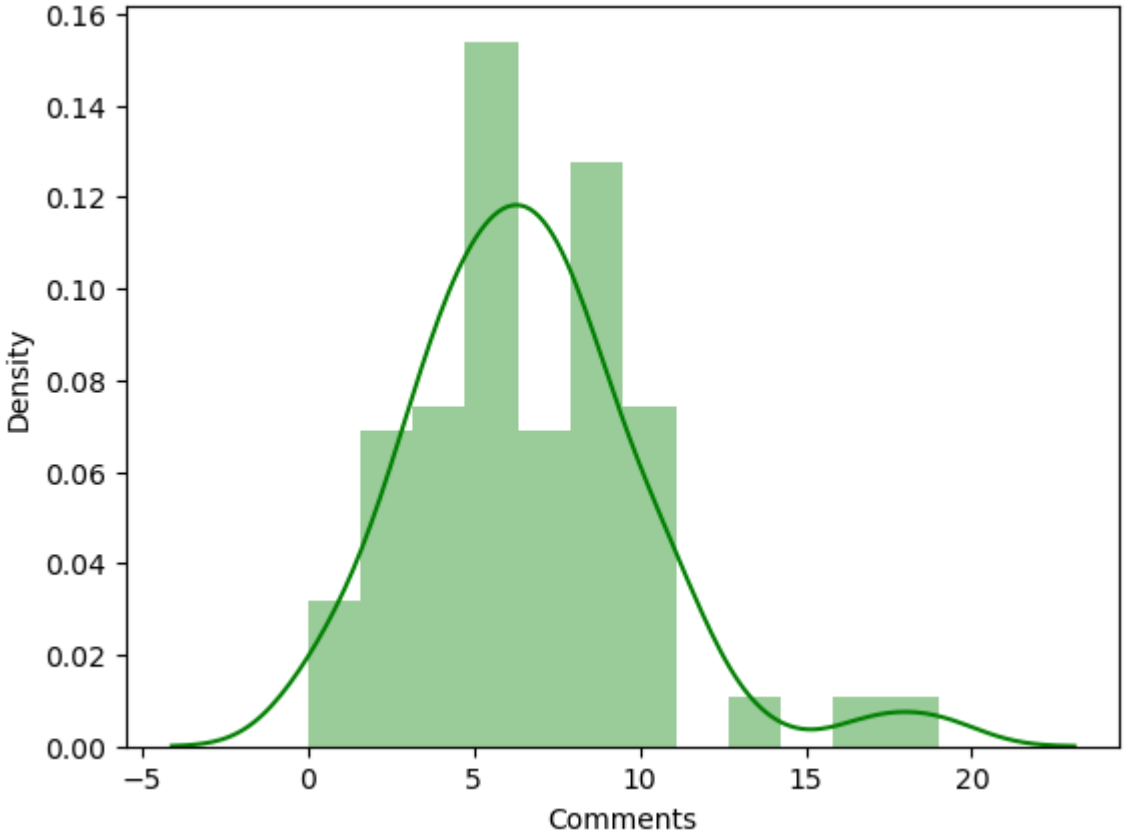

0.0008
0.0006
0.0004
0.0002
0.0000
Density
0
2000
4000
6000
8000
10000
12000
14000
From Home

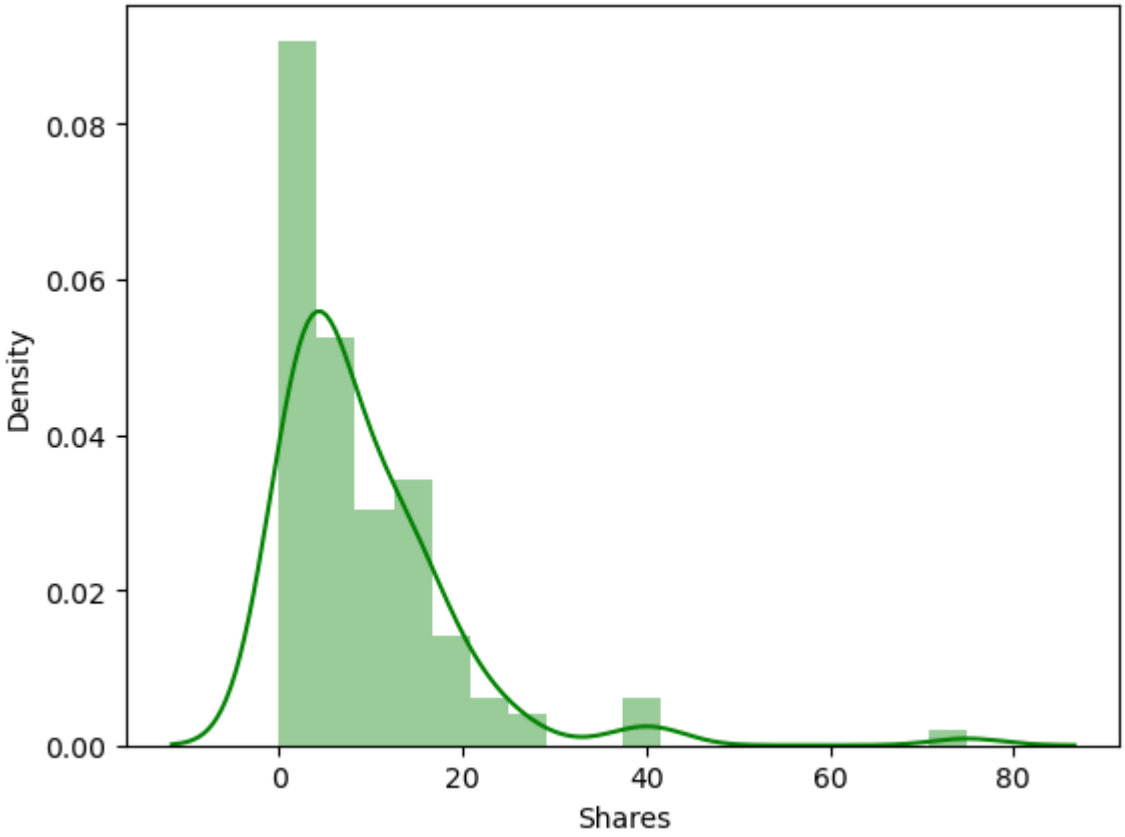

0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
−2000
0
2000
4000
6000
8000
10000
12000
14000
From Hashtags

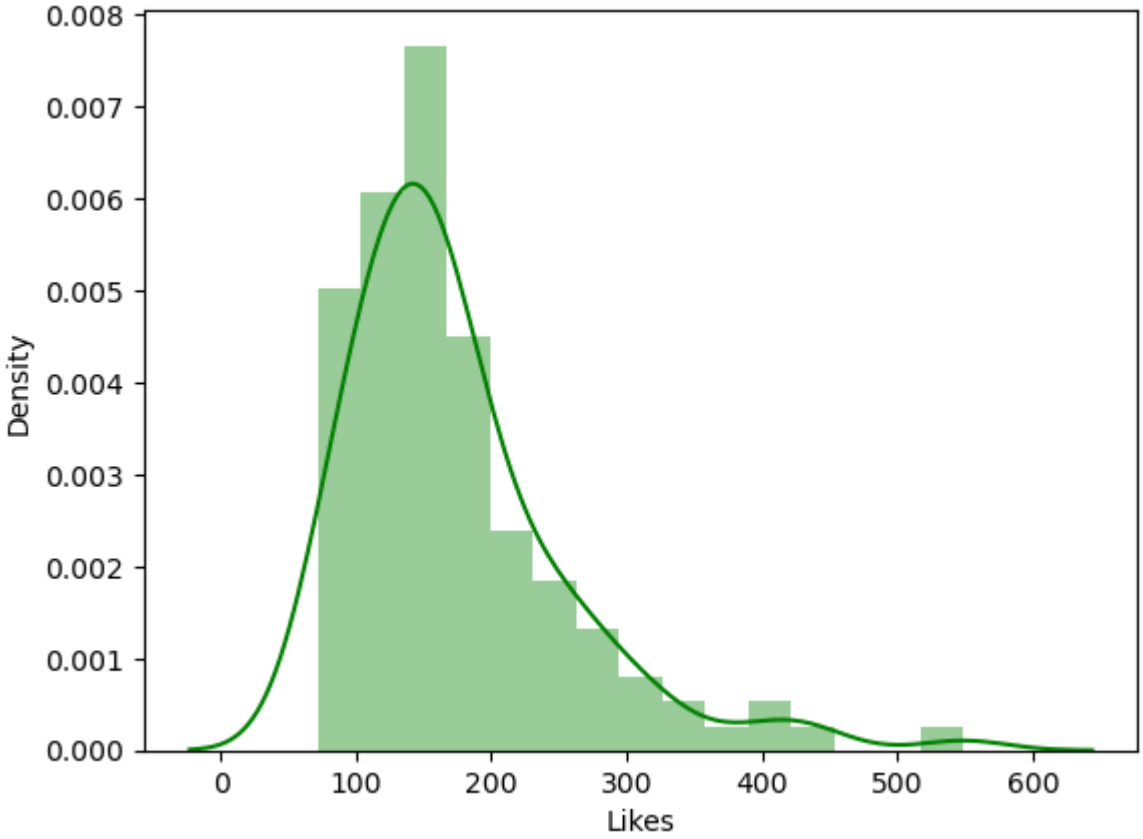

0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
Density
0
5000
10000
15000
20000
From Explore

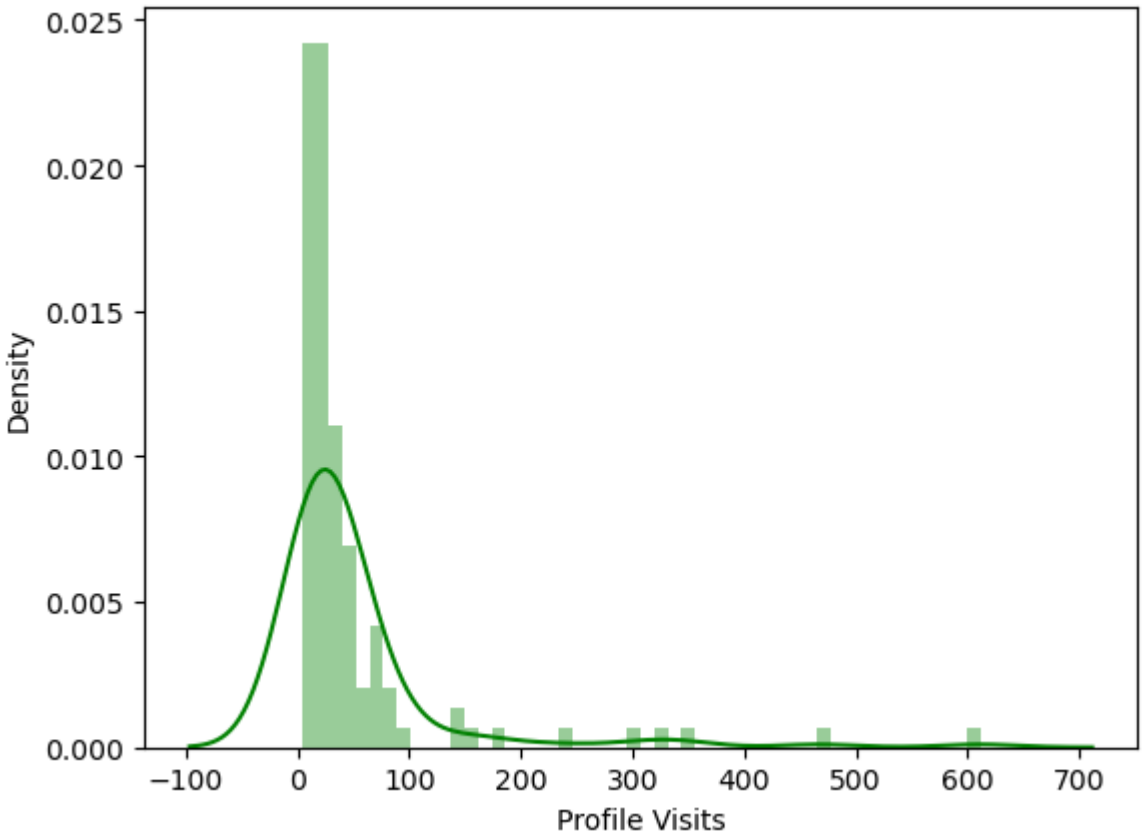

0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
0
500
1000
1500
2000
2500
3000
From Other

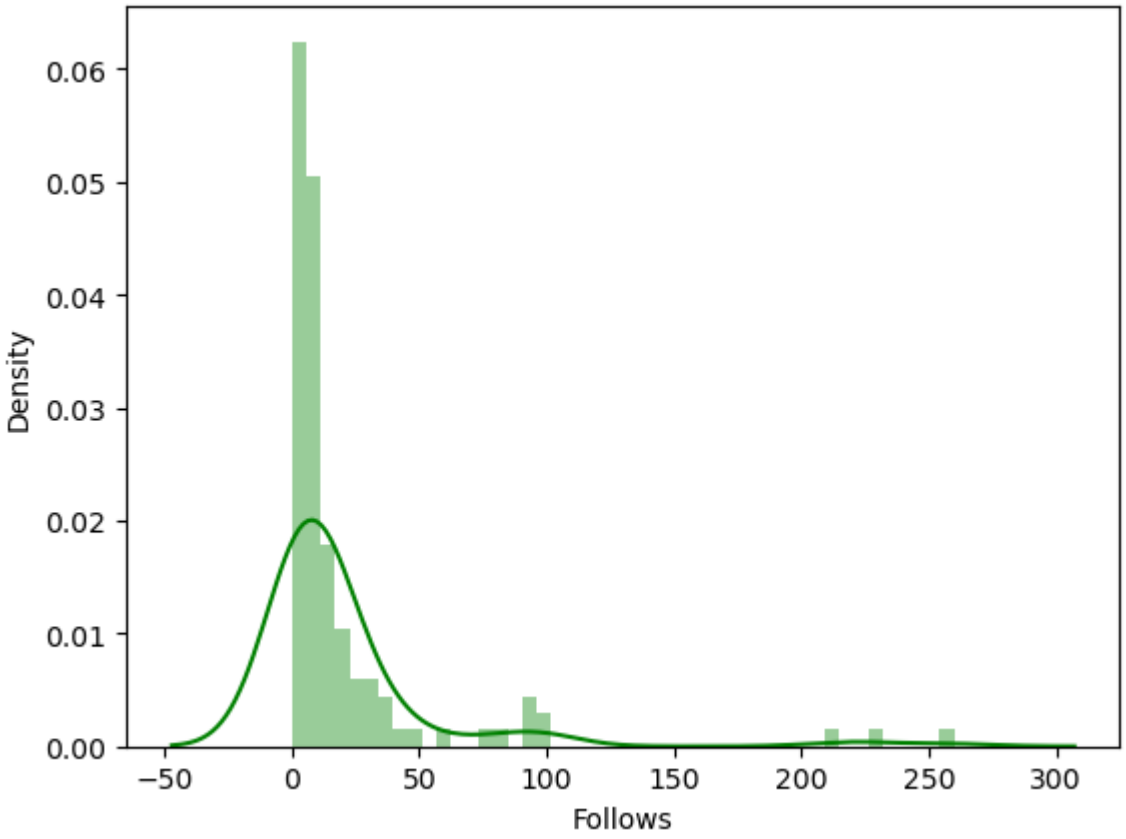

0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
−200
0
200
400
600
800
1000
1200
Saves


0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
−5
0
5
10
15
20
Comments


0.08
0.06
0.04
0.02
0.00
Density
0
20
40
60
80
Shares

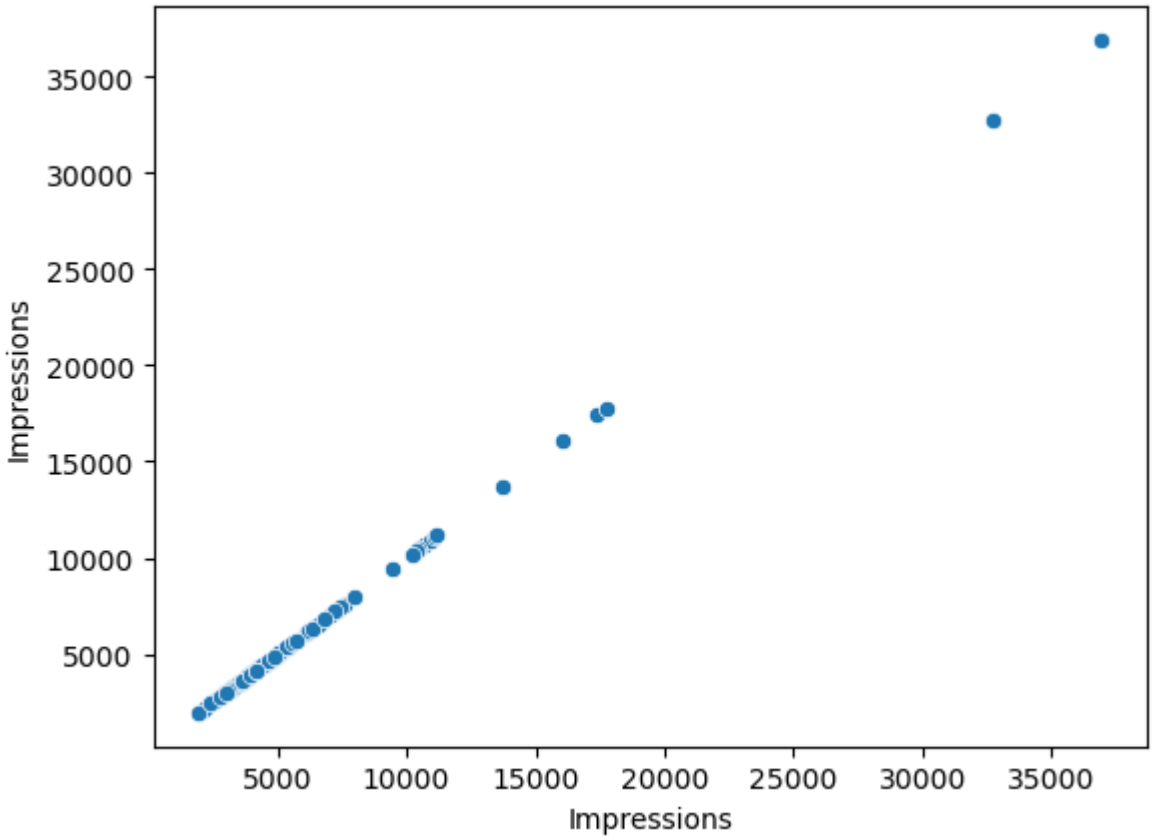

0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
0
100
200
300
400
500
600
Likes

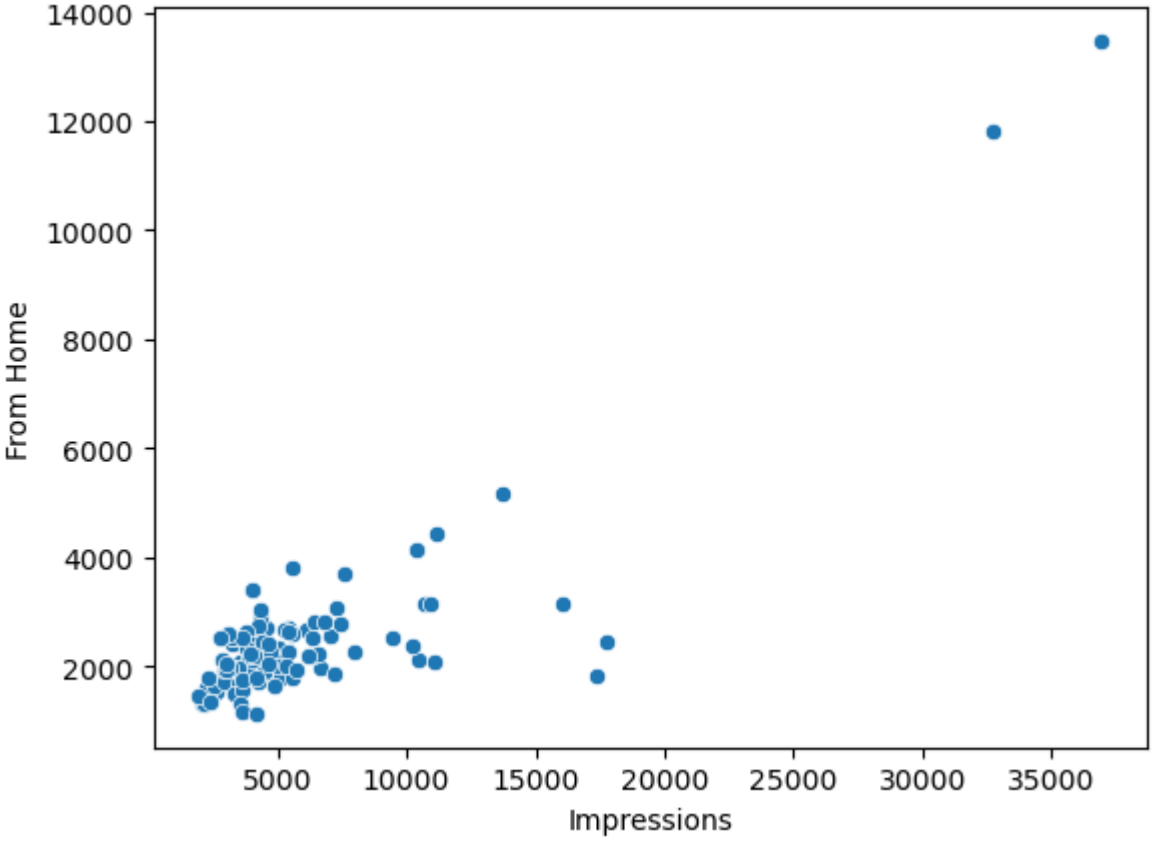

0.025
0.020
0.015
0.010
0.005
0.000
Density
−100
0
100
200
300
400
500
600
700
Profile Visits

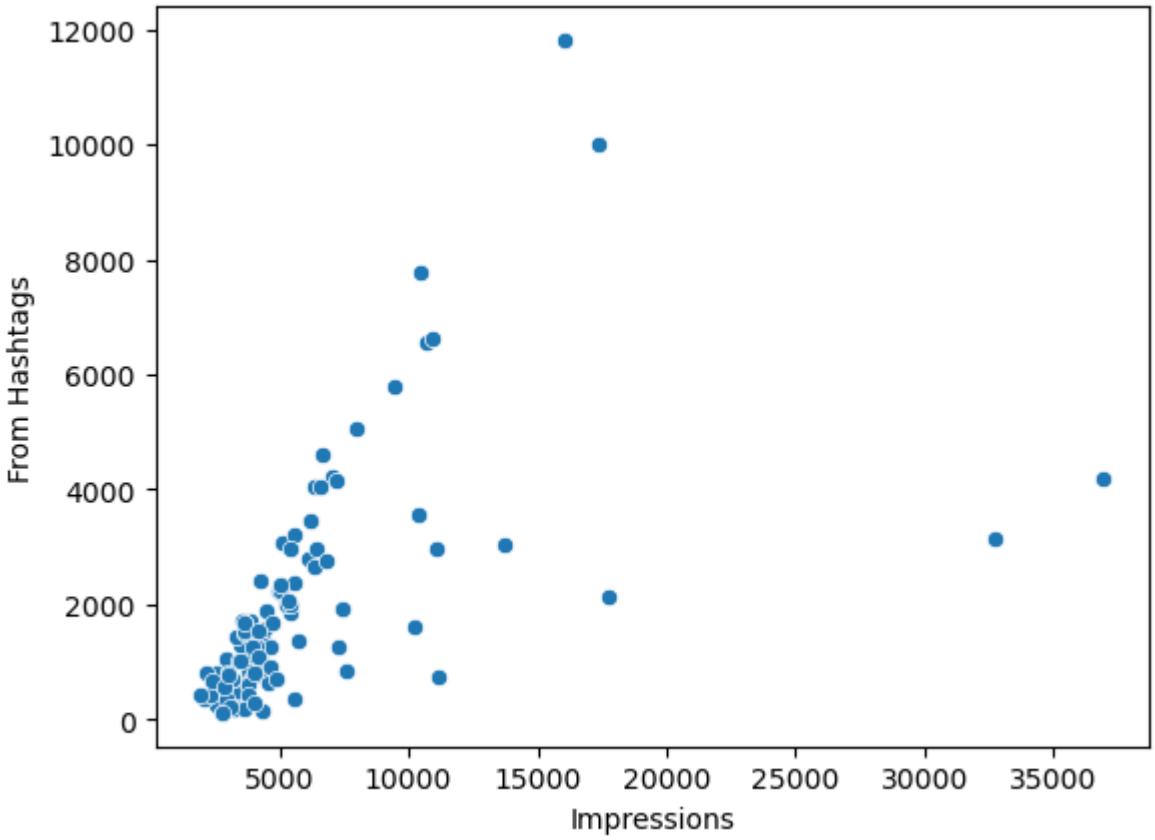


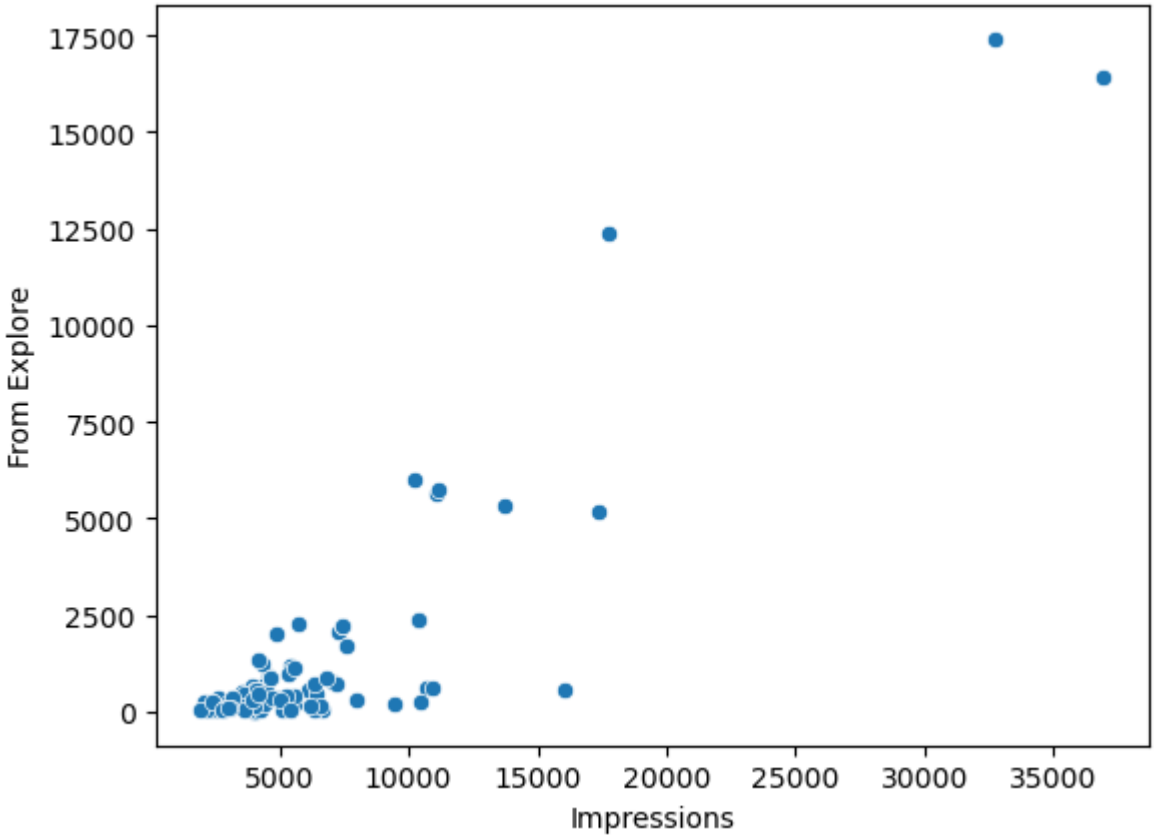
In [14]:
```
#we can see that in the dataset numerical distrubition in normal.
```

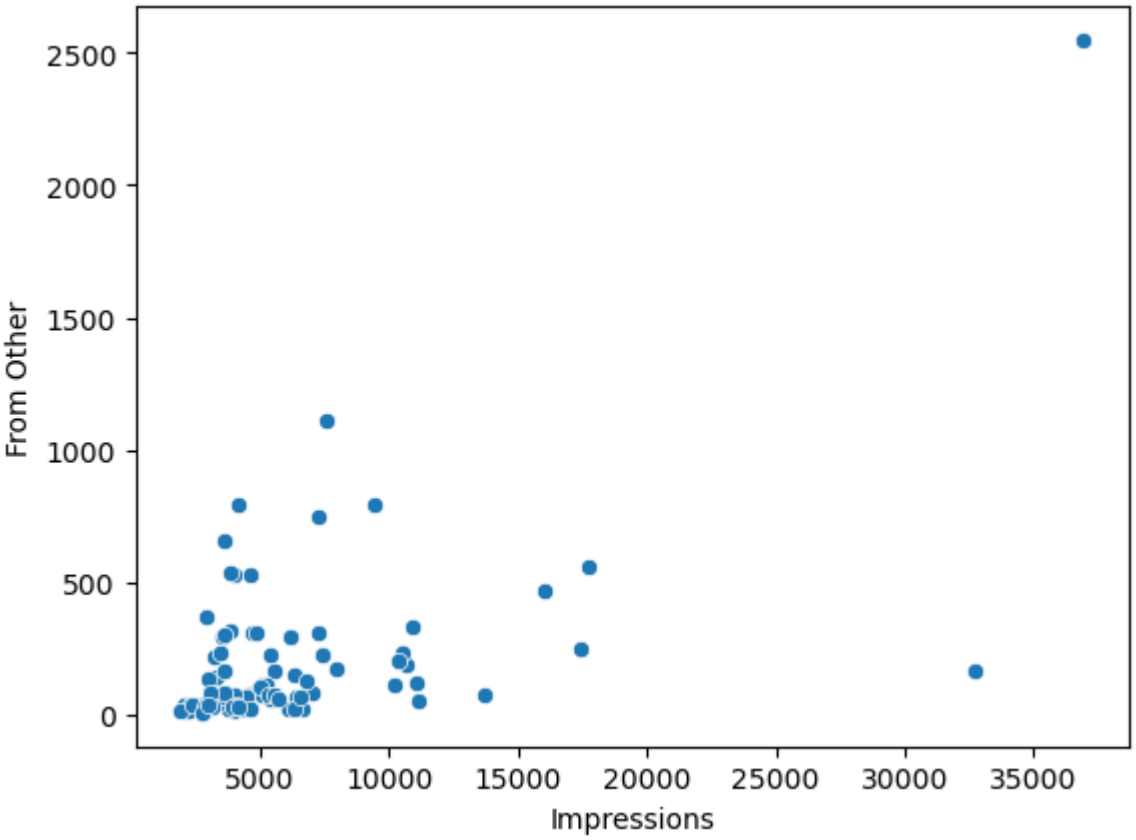
In [15]:
```
#now we can check the numerical feature and target variable correlation.
#This also helps in uncovering useful and actionable insights from the data.
#One can also get the outliers from the scatterplots.
for i in df_num:
    sns.scatterplot(df, y=df[i], x=df["Impressions"])
    plt.show()
```

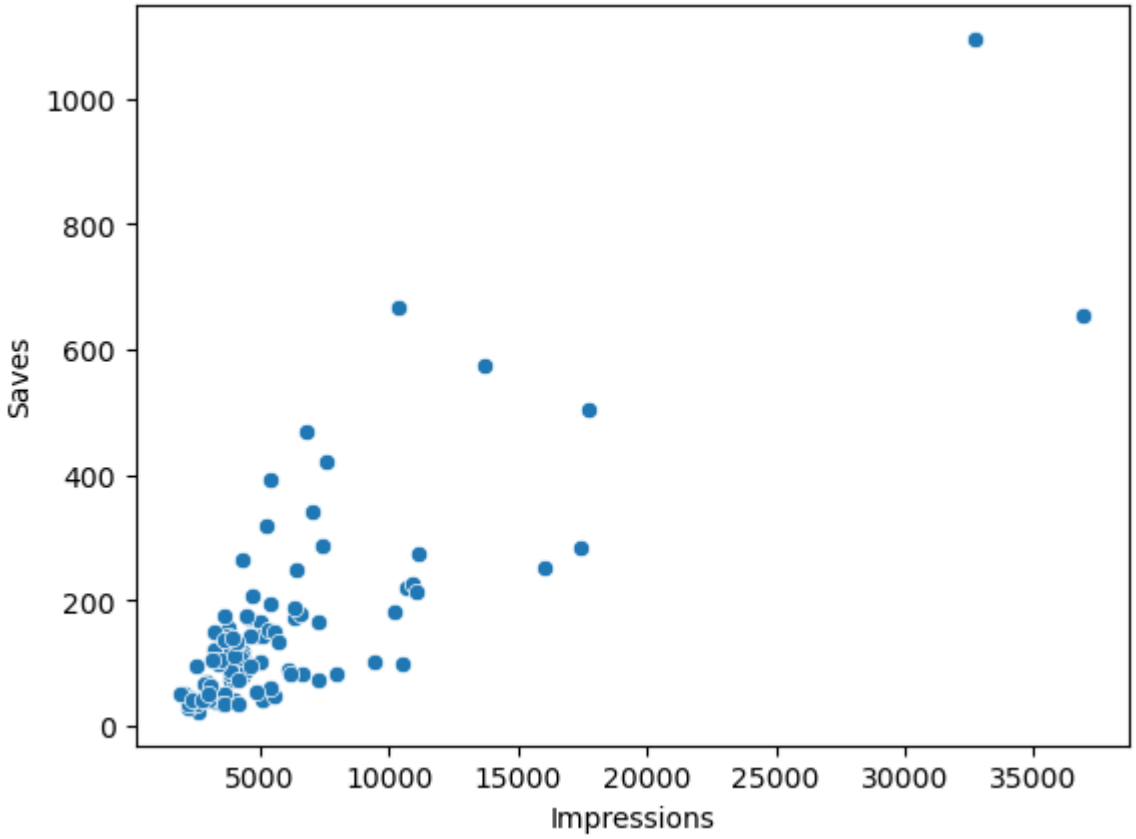

35000
30000
25000
20000
15000
10000
5000
Impressions
5000
10000
15000
20000
25000
30000
35000
Impressions

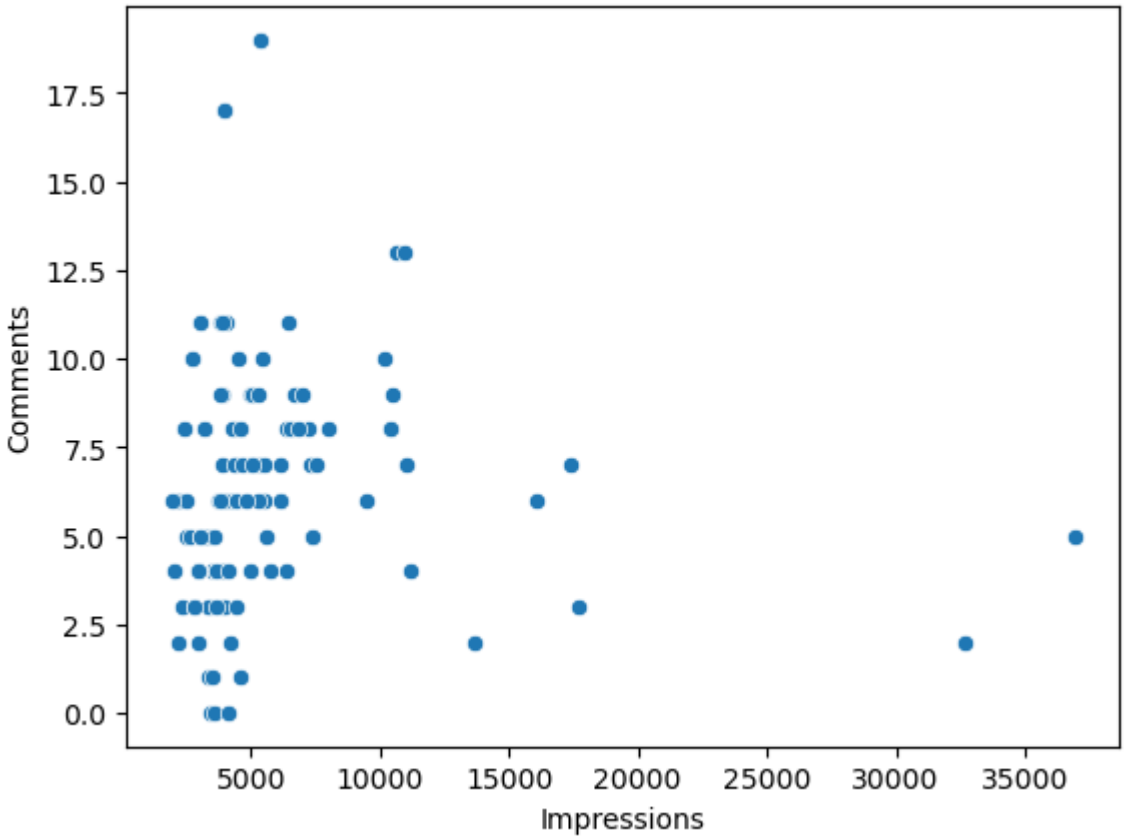

14000
12000
10000
8000
6000
4000
2000
From Home
5000
10000
15000
20000
25000
30000
35000
Impressions

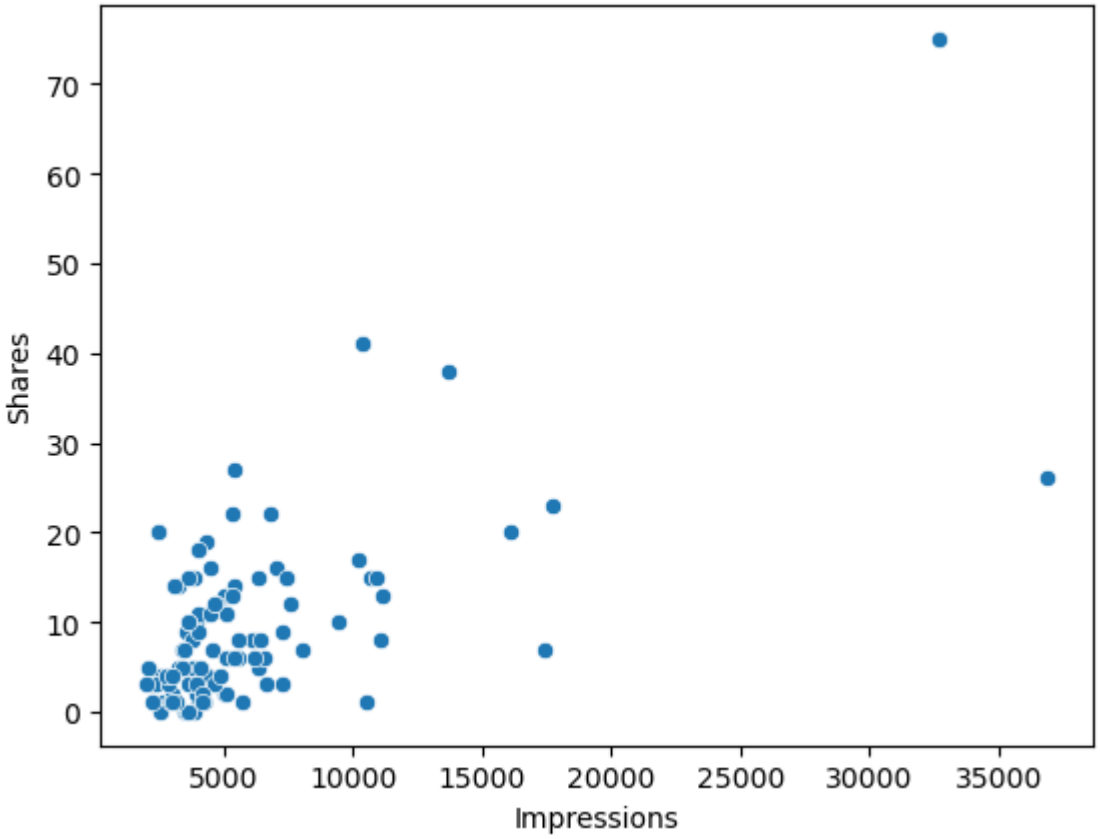

From Hashtags
Impressions
0
2000
4000
6000
8000
10000
12000
5000
10000
15000
20000
25000
30000
35000

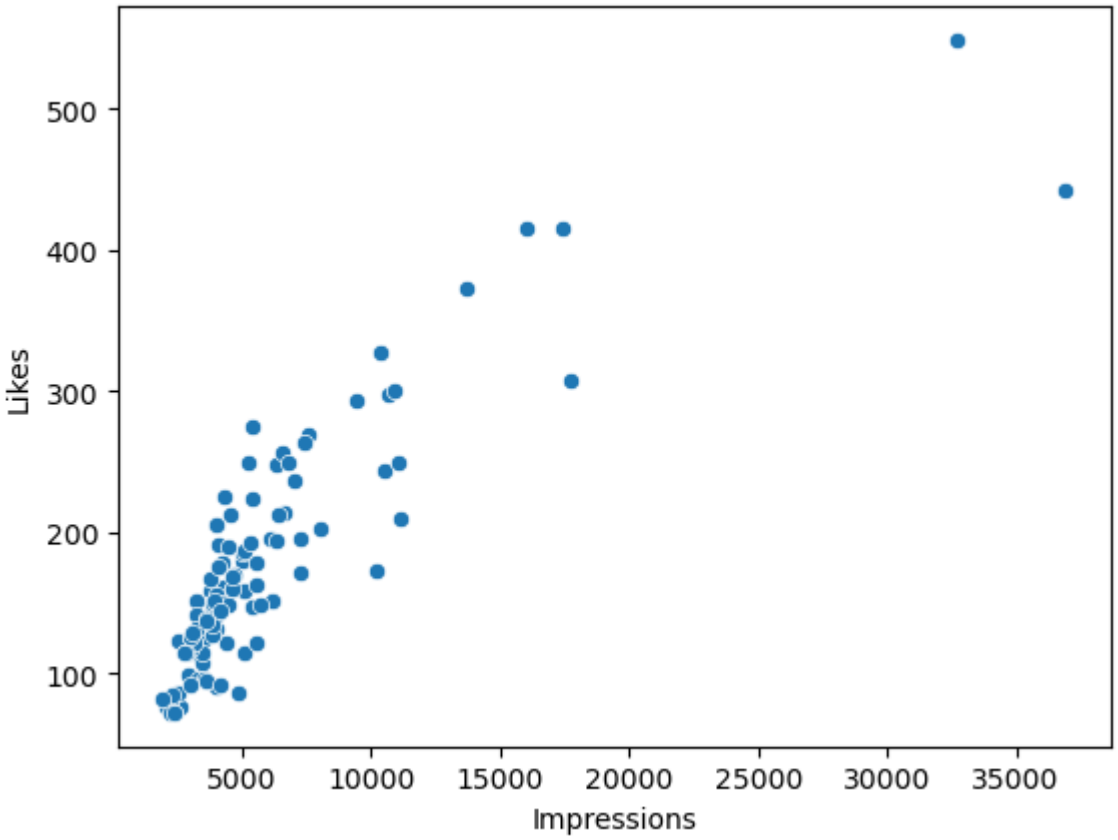

From Explore
Impressions
0
2500
5000
7500
10000
12500
15000
17500
5000
10000
15000
20000
25000
30000
35000

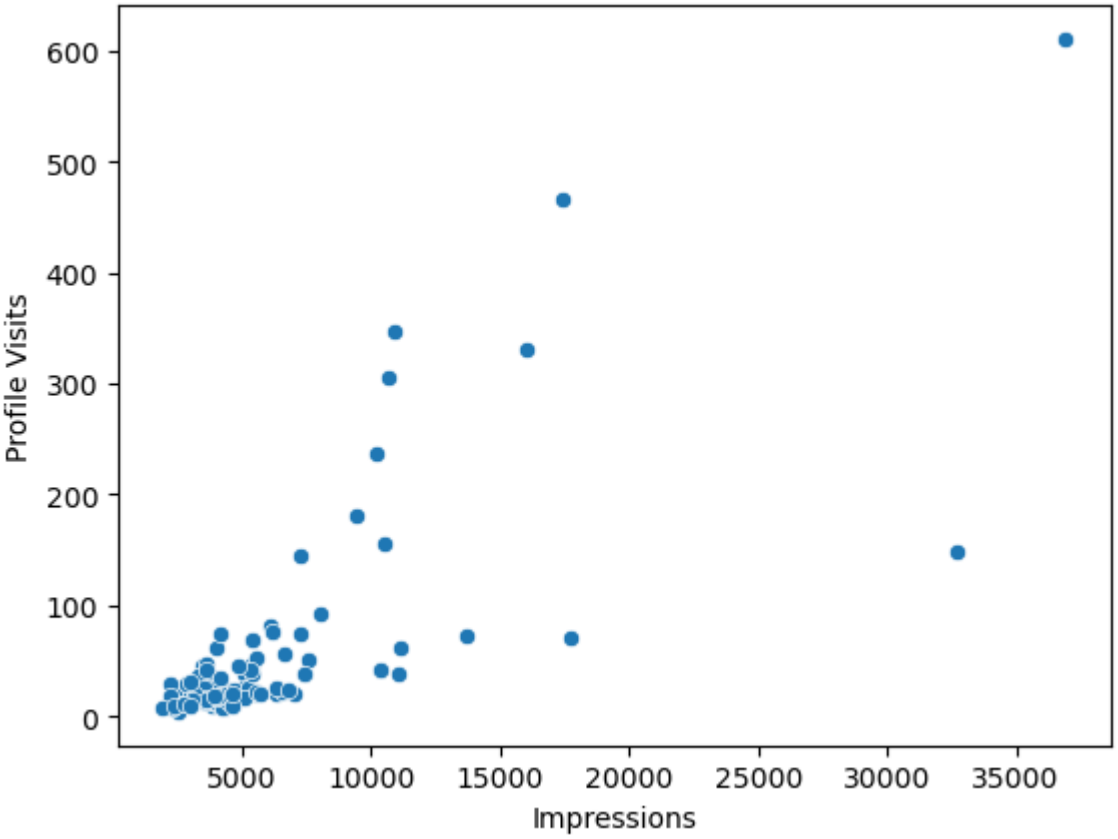

From Other
Impressions
0
500
1000
1500
2000
2500
5000
10000
15000
20000
25000
30000
35000

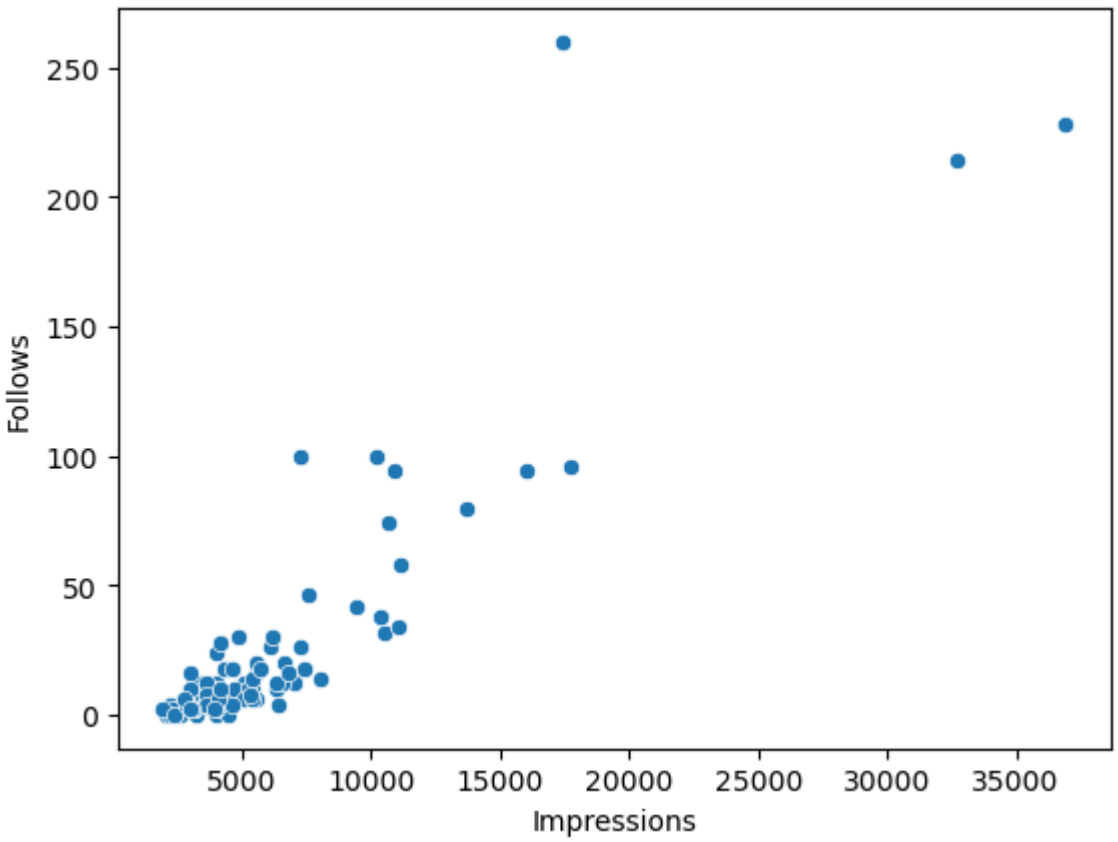

Saves
Impressions
0
200
400
600
800
1000
5000
10000
15000
20000
25000
30000
35000


Comments
Impressions
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
5000
10000
15000
20000
25000
30000
35000


Shares
Impressions
0
10
20
30
40
50
60
70
5000
10000
15000
20000
25000
30000
35000


Likes
Impressions
100
200
300
400
500
5000
10000
15000
20000
25000
30000
35000


Profile Visits
Impressions
0
100
200
300
400
500
600
5000
10000
15000
20000
25000
30000
35000



Here, all the features are important and also correlated with target variable.

Display and remove the duplicate rows in the Dataframe. Duplicate rows increase the computational time of the Machine Learning model and also result in falsely positive results.

Seperate data in X and Y as well as Split data into train and Test

In [16]:
```
df.columns
```

Out[16]:
```
Index(['Impressions', 'From Home', 'From Hashtags', 'From Explore',
       'From Other', 'Saves', 'Comments', 'Shares', 'Likes', 'Profile Visit
s',
       'Follows', 'Caption', 'Hashtags'],
      dtype='object')
```

In [17]:
```
# I am using a df1 data which was copy of the original data set.
x = df1.drop(["Impressions"], axis=1)
y = df1["Impressions"]
```

In [18]:
```
#for train test split import neccasary library
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y, random_state=50, tes
t_size=0.2)
```

In [19]:
```
train_x
```

Out[19]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	C
8	2384	857	248	49	155	6	8	159	36	4	H some be a p
102	1807	1085	463	792	74	4	2	145	75	28	co has it of c an C
18	2177	3450	153	296	82	6	6	151	77	30	anxie depr threa
42	2746	869	589	24	111	8	4	150	19	18	Here diff betwe proce
3	2700	621	932	73	172	10	7	213	23	8	Here you ca a prog
...	...	...	...	...	...	...	...	...	...	...	
33	2278	1460	521	27	105	6	3	152	24	8	I lar proc or N s
109	2449	2141	12389	561	504	3	23	308	70	96	H some resou lea
11	2414	476	185	75	122	8	14	151	15	0	Here to liv pric
96	2541	116	51	9	40	10	4	114	11	6	Here progra lang th
48	2782	1938	2237	226	288	5	15	263	39	18	H some im machi

95 rows × 12 columns

In [20]:
```
#we can reset index
train_x.reset_index(inplace=True, drop=True)
test_x.reset_index(inplace=True, drop=True)

train_y.reset_index(inplace=True, drop=True)
test_y.reset_index(inplace=True, drop=True)
```

Encoding using LeaveOneOut Encoder

In [21]:
```
#for create encoding for input variables we can seperate dataof numerical and categorical
train_cat = train_x.select_dtypes(include="object")
train_num = train_x.select_dtypes(include="number")

test_cat = test_x.select_dtypes(include="object")
test_num = test_x.select_dtypes(include="number")
```

In [22]:
```
train_cat
```

Out[22]:

	Caption	Hashtags
0	Here are some of the best data analysis projec...	#dataanalytics #datascience #data #machinelear...
1	Each company has its ways of creating an OTP f...	#python #pythonprogramming #pythonprojects #py...
2	Stress, anxiety, and depression are threatenin...	#data #datascience #dataanalysis #dataanalytic...
3	Here is the difference between the process of ...	#data #datascience #dataanalysis #dataanalytic...
4	Here□s how you can write a Python program to d...	#python #pythonprogramming #pythonprojects #py...
...	...	...
90	Natural language processing or NLP is a subfie...	#data #datascience #dataanalysis #dataanalytic...
91	Here are some of the best resources to learn S...	#sql #mysql #datascience #datasciencejobs #dat...
92	Here□s how to get the live stock price data of...	#python #pythonprogramming #pythonprojects #py...
93	Here are all the programming languages that Fa...	#programming #coding #programmer #python #deve...
94	Here are some of the most important machine le...	#data #datascience #dataanalysis #dataanalytic...

95 rows × 2 columns

In [23]:
```
#First we check null value of dataset. If have then first impute null value.
train_cat.isnull().sum()
```

Out[23]:
```
Caption     0
Hashtags    0
dtype: int64
```

In [24]:
```
import category_encoders as ce
encoder = ce.LeaveOneOutEncoder()
encoder.fit(train_cat, train_y)
```

Out[24]:
```
LeaveOneOutEncoder(cols=['Caption', 'Hashtags'])
```

In [25]:
```
train_cat = encoder.transform(train_cat)
test_cat = encoder.transform(test_cat)
```

In [26]:
```
train_cat
```

Out[26]:

	Caption	Hashtags
0	5764.694737	3287.500000
1	4015.000000	4015.000000
2	6168.000000	6168.000000
3	5764.694737	5764.694737
4	5764.694737	6436.571429
...	...	...
90	5764.694737	5077.250000
91	5764.694737	5764.694737
92	5764.694737	6436.571429
93	5764.694737	3306.250000
94	5764.694737	6969.250000

95 rows × 2 columns

In [27]:
```
# Now, we concat the both categorical and numerical data
train_x1 = pd.concat([train_num, train_cat], axis=1)
test_x1 = pd.concat([test_num, test_cat], axis=1)
```

In [28]:
```
train_x1
```

Out[28]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Ca
0	2384	857	248	49	155	6	8	159	36	4	5764.69
1	1807	1085	463	792	74	4	2	145	75	28	4015.00
2	2177	3450	153	296	82	6	6	151	77	30	6168.00
3	2746	869	589	24	111	8	4	150	19	18	5764.69
4	2700	621	932	73	172	10	7	213	23	8	5764.69
...	...	...	...	...	...	...	...	...	...	...	
90	2278	1460	521	27	105	6	3	152	24	8	5764.69
91	2449	2141	12389	561	504	3	23	308	70	96	5764.69
92	2414	476	185	75	122	8	14	151	15	0	5764.69
93	2541	116	51	9	40	10	4	114	11	6	5764.69
94	2782	1938	2237	226	288	5	15	263	39	18	5764.69

95 rows × 12 columns

Scaling Using MinmaxScaler

In [29]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
scaler = MinMaxScaler()
scaler.fit(train_x1)
```

Out[29]:

▾ MinMaxScaler

MinMaxScaler()

In [30]:
```
train_x1 = pd.DataFrame(scaler.transform(train_x1), columns=train_x1.columns)
test_x1 = pd.DataFrame(scaler.transform(test_x1), columns=test_x1.columns)
```

In [31]:
```
train_x1
```

Out[31]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits
0	0.117113	0.063328	0.014241	0.036166	0.123952	0.315789	0.106667	0.182390	0.069114
1	0.063097	0.082813	0.026588	0.707957	0.048462	0.210526	0.026667	0.153040	0.153348
2	0.097735	0.284933	0.008786	0.259494	0.055918	0.315789	0.080000	0.165618	0.157667
3	0.151002	0.064353	0.033823	0.013562	0.082945	0.421053	0.053333	0.163522	0.032397
4	0.146695	0.043159	0.053520	0.057866	0.139795	0.526316	0.093333	0.295597	0.041037
...	...	...	...	...	...	...	...	...	...
90	0.107190	0.114862	0.029918	0.016275	0.077353	0.315789	0.040000	0.167715	0.043197
91	0.123198	0.173062	0.711439	0.499096	0.449208	0.157895	0.306667	0.494759	0.142549
92	0.119921	0.030767	0.010624	0.059675	0.093197	0.421053	0.186667	0.165618	0.023758
93	0.131811	0.000000	0.002929	0.000000	0.016775	0.526316	0.053333	0.088050	0.015119
94	0.154372	0.155713	0.128460	0.196203	0.247903	0.263158	0.200000	0.400419	0.075594

95 rows × 12 columns

Model Building And Evaluation

In [32]:
```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.svm import SVR
import xgboost as Xgb
```

In [33]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [34]:
```
#LinerRegression
log_model = LinearRegression()
log_model.fit(train_x1, train_y)
pred_log = log_model.predict(test_x1)
mea_log = mean_absolute_error(test_y, pred_log)
mea_log
```

Out[34]: 81.7535792170957

In [35]:
```
log_model.score(train_x1, train_y)
```

Out[35]: 0.9991329465087581

In [36]:
```
log_model.score(test_x1, test_y)
```

Out[36]: 0.9995112477143003

In [37]:
```
#KNEARASTNEIGHBORS REGRESSOR
knn_model = KNeighborsRegressor(n_neighbors=10)
knn_model.fit(train_x1, train_y)
pred_knn = knn_model.predict(test_x1)
mea_knn = mean_absolute_error(test_y, pred_knn)
mea_knn
```

Out[37]: 1512.7749999999999

In [38]:
```
knn_model.score(train_x1, train_y)
```

Out[38]: 0.6349149903298423

In [39]:
```
knn_model.score(test_x1, test_y)
```

Out[39]: 0.47226856116825744

In [40]:
```
# SUPPORT VECTOR REGRESSOR
svm_model = SVR(kernel="rbf")
svm_model.fit(train_x1, train_y)
pred_svm = svm_model.predict(test_x1)
mea_svm = mean_absolute_error(test_y, pred_svm)
mea_svm
```

Out[40]: 2372.1453299610607

In [41]:
```
#DECISION TREE REGRESSOR
dt_model = DecisionTreeRegressor(random_state=50)
dt_model.fit(train_x1, train_y)
pred_dt = dt_model.predict(test_x1)
mea_dt = mean_absolute_error(test_y, pred_dt)
mea_dt
```

Out[41]: 516.0

In [42]:
```
dt_model.score(train_x1, train_y)
```

Out[42]: 1.0

In [43]:
```
dt_model.score(test_x1,test_y)
```

Out[43]: 0.9748934332264583

In [44]:
```
#RANDOM FOREST REGRESSOR
rfc_model = RandomForestRegressor(random_state=50)
rfc_model.fit(train_x1, train_y)
pred_rfc = rfc_model.predict(test_x1)
mea_rfc = mean_absolute_error(test_y, pred_rfc)
mea_rfc
```

Out[44]: 960.7120833333332

In [45]:
```
rfc_model.score(train_x1, train_y)
```

Out[45]: 0.9531413802906008

In [46]:
```
rfc_model.score(test_x1, test_y)
```

Out[46]: 0.7233024078534206

In [47]:
```
#XGBOOST REGRESSOR
xgb_model = Xgb.XGBRegressor(n_estimators=100)
xgb_model.fit(train_x1, train_y)
pred_xgb = xgb_model.predict(test_x1)
mea_xgb = mean_absolute_error(test_y, pred_xgb)
mea_xgb
```

Out[47]: 350.55299886067706

In [48]:
```
xgb_model.score(train_x1, train_y)
```

Out[48]: 0.9999999999993842

In [49]:
```
xgb_model.score(test_x1, test_y)
```

Out[49]: 0.980260753353657

In [50]:
```
#ADABOOST REGRESSOR
from sklearn.ensemble import AdaBoostRegressor
adb_model = AdaBoostRegressor(random_state=50)
adb_model.fit(train_x1, train_y)
pred_adb = adb_model.predict(test_x1)
mea_adb = mean_absolute_error(test_y, pred_adb)
mea_adb
```

Out[50]: 1409.7594502411914

In [51]:
```
adb_model.score(train_x1, train_y)
```

Out[51]: 0.9790460504777033

In [52]:
```
adb_model.score(test_x1, test_y)
```

Out[52]: 0.5618448373633542

HYPERPARAMTER TUNING

In [53]:
```
from sklearn.model_selection import GridSearchCV
```

In [54]:
```
#HYPERPERAMETER TUING OF KNN
knn =KNeighborsRegressor()
params_knn= {'algorithm' :['auto', 'ball_tree', 'kd_tree', 'brute'], 'weight
s': ['uniform', 'distance'],
         "n_neighbors" : [1,25,14,13,26,85,45]}
clf2 = GridSearchCV(knn, params_knn, cv=5, scoring="neg_mean_absolute_error")
clf2.fit(train_x1, train_y)
print(clf2.best_params_)
print(-(clf2.best_score_))
```

```
{'algorithm': 'auto', 'n_neighbors': 1, 'weights': 'uniform'}
1165.8736842105263
```

In [55]:
```
#HYPERPERAMETER TUNING OF SUPPORT VECTOR
svm = SVR()
params_svm = {"gamma" :["scale", "auto"]}
clf4 = GridSearchCV(svm, params_svm, cv=5, scoring="neg_mean_absolute_error")
clf4.fit(train_x1, train_y)
print(clf4.best_params_)
print(-(clf4.best_score_))
```

```
{'gamma': 'scale'}
2395.2378812261604
```

In [56]:
```
#HYPERPERAMETER TUNING OF DECISION TREE
dt = DecisionTreeRegressor()
params_dt = { 'max_depth' :[1,25,14,13,45,75,26],'splitter':['best', 'rando
m']}
clf5 = GridSearchCV(dt, params_dt, cv=5, scoring="neg_mean_absolute_error")
clf5.fit(train_x1, train_y)
print(clf5.best_params_)
print(-(clf5.best_score_))
```

```
{'max_depth': 13, 'splitter': 'random'}
1036.378947368421
```

In [57]:
```
#HYPERPERAMETER TUNING OF RANDOMFOREST
rfc = RandomForestRegressor()
params_rfc = {"n_estimators" : [10,15,125,10,8,85],"max_depth" : [10,25,48,85,
42,3]}
clf6 = GridSearchCV(rfc, params_rfc, cv=5, scoring="neg_mean_absolute_error")
clf6.fit(train_x1, train_y)
print(clf6.best_params_)
print(-(clf6.best_score_))
```

```
{'max_depth': 48, 'n_estimators': 8}
759.6552631578948
```

In [58]:
```
#HYPERPERAMETER TUNING OF XGBOOST
xgb = Xgb.XGBRegressor()
params_xgb = {'eta': [0.1, 0.2, 0.3,0.4,0.5], 'n_estimators' : [10, 50, 100,1
2,15], 'max_depth': [3, 6, 9,14]}
clf7 = GridSearchCV(xgb, params_xgb, cv=5, scoring="neg_mean_absolute_error")
clf7.fit(train_x1, train_y)
print(clf7.best_params_)
print(-(clf7.best_score_))
```

```
{'eta': 0.4, 'max_depth': 3, 'n_estimators': 100}
863.0469572368422
```

In [59]:
```
#HYPERPERAMETER TUNING OF ADABOOST
adb = AdaBoostRegressor()
params_adb = {'n_estimators' : [10, 50, 100,12,15]}
clf8 = GridSearchCV(adb, params_adb, cv=5, scoring="neg_mean_absolute_error")
clf8.fit(train_x1, train_y)
print(clf8.best_params_)
print(-(clf8.best_score_))
```

```
{'n_estimators': 15}
1056.9163613317749
```

In [60]:
```
#best perameter for model
print("KNeighborsRegressor score is :", clf2.best_params_)
print("Support vector machine score is :", clf4.best_params_)
print("DecisionTreeRegressor score is :", clf5.best_params_)
print("RandomForestRegressor score is :", clf6.best_params_)
print("XGBOOST score is :", clf7.best_params_)
print("AdaBoostRegressor score is :", clf8.best_params_)
```

```
KNeighborsRegressor score is : {'algorithm': 'auto', 'n_neighbors': 1, 'weigh
ts': 'uniform'}
Support vector machine score is : {'gamma': 'scale'}
DecisionTreeRegressor score is : {'max_depth': 13, 'splitter': 'random'}
RandomForestRegressor score is : {'max_depth': 48, 'n_estimators': 8}
XGBOOST score is : {'eta': 0.4, 'max_depth': 3, 'n_estimators': 100}
AdaBoostRegressor score is : {'n_estimators': 15}
```

In [61]:
```
#Score for all model
print("KNeighborsRegressor score is :", -clf2.best_score_)
print("Support vector machine score is :", -clf4.best_score_)
print("DecisionTreeRegressor score is :", -clf5.best_score_)
print("RandomForestRegressor score is :", -clf6.best_score_)
print("XGBOOST score is :", -clf7.best_score_)
print("AdaBoostRegressor score is :", -clf8.best_score_)
```

```
KNeighborsRegressor score is : 1165.8736842105263
Support vector machine score is : 2395.2378812261604
DecisionTreeRegressor score is : 1036.378947368421
RandomForestRegressor score is : 759.6552631578948
XGBOOST score is : 863.0469572368422
AdaBoostRegressor score is : 1056.9163613317749
```

Feature Selection

In [62]:
```
corr = train_x1.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[62]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes
From Home	1.000000	0.138370	0.717187	0.071340	0.833363	-0.042784	0.802148	0.708273
From Hashtags	0.138370	1.000000	0.134536	0.189124	0.268558	0.238623	0.189295	0.674721
From Explore	0.717187	0.134536	1.000000	0.133221	0.738737	-0.189402	0.667193	0.622187
From Other	0.071340	0.189124	0.133221	1.000000	0.144780	-0.118884	0.026854	0.242007
Saves	0.833363	0.268558	0.738737	0.144780	1.000000	-0.066305	0.874237	0.828665
Comments	-0.042784	0.238623	-0.189402	-0.118884	-0.066305	1.000000	-0.085883	0.066612
Shares	0.802148	0.189295	0.667193	0.026854	0.874237	-0.085883	1.000000	0.685262
Likes	0.708273	0.674721	0.622187	0.242007	0.828665	0.066612	0.685262	1.000000
Profile Visits	0.213396	0.780445	0.301381	0.327679	0.235005	0.208269	0.189151	0.598232
Follows	0.557238	0.563546	0.729475	0.315655	0.589145	-0.036981	0.497840	0.744226
Caption	0.692465	0.420616	0.594892	0.028475	0.679746	-0.074794	0.588303	0.705111
Hashtags	0.529953	0.242084	0.531133	0.015002	0.611595	-0.108138	0.539785	0.531928

In [63]:
```
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range (len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

In [64]:
```
corr_features = correlation(train_x1, 0.7)
len(set(corr_features))
```

Out[64]: 7

In [65]:
```
print(corr_features)
```

```
{'Saves', 'Likes', 'Profile Visits', 'Caption', 'From Explore', 'Shares', 'Fo
llows'}
```

In [66]:
```
#Apply SelectKbest class to extract top Features
from sklearn.feature_selection import SelectKBest, chi2
bestfeatures = SelectKBest(score_func=chi2, k=7)
fit = bestfeatures.fit(train_x1,train_y)
```

In [67]:
```
dfscores = pd.DataFrame(fit.scores_)
```

In [68]:
```
dfcolumns = pd.DataFrame(x.columns)
```

In [69]:
```
features = pd.concat([dfcolumns, dfscores], axis=1)
features.columns = ["specs", "score"]
```

In [70]:
```
features
```

Out[70]:

	specs	score
0	From Home	9.696609
1	From Hashtags	17.276084
2	From Explore	30.195933
3	From Other	22.190105
4	Saves	16.899517
5	Comments	7.213436
6	Shares	14.404950
7	Likes	13.075443
8	Profile Visits	25.834542
9	Follows	26.682716
10	Caption	11.341093
11	Hashtags	11.535113

In [71]:
```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(train_x1, train_y)
```

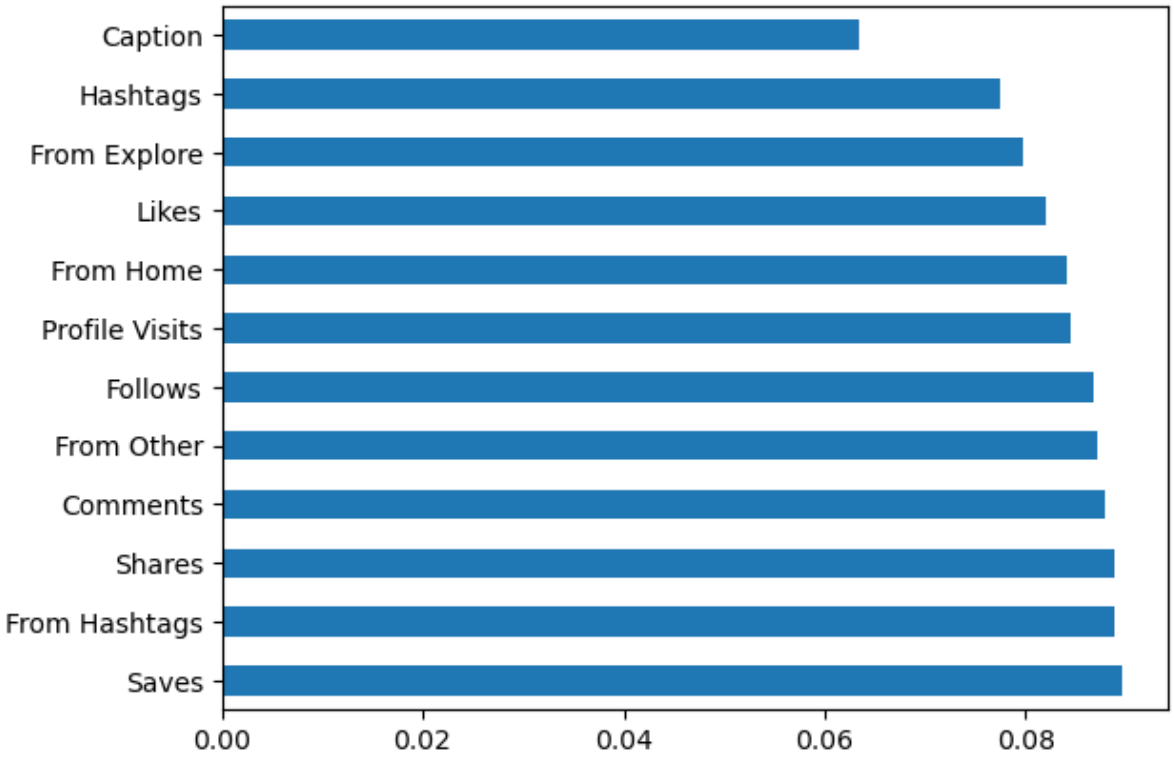
Out[71]:
```
▾ ExtraTreesClassifier

ExtraTreesClassifier()
```

In [72]:
```
print(model.feature_importances_)
```

```
[0.08404783 0.08879931 0.07965104 0.08708181 0.08957946 0.08786947
 0.08879183 0.08191126 0.08452973 0.08682356 0.06346818 0.07744653]
```

In [73]:
```
feat_importance = pd.Series(model.feature_importances_, index=x.columns)
feat_importance.nlargest(12).plot(kind="barh")
plt.show()
```



In [74]:
```
from sklearn.ensemble import RandomForestClassifier
fe_model = RandomForestClassifier(random_state=50)

fe_model.fit(train_x1, train_y)
```

Out[74]:
```
RandomForestClassifier
RandomForestClassifier(random_state=50)
```

In [75]:
```
feature_scores = pd.Series(fe_model.feature_importances_, index=train_x.columns).sort_values(ascending=False)
```

In [76]:
```
feature_scores
```

Out[76]:
```
From Other         0.099350
From Explore       0.098265
From Hashtags      0.095978
Likes              0.091009
Profile Visits     0.090553
Saves              0.086201
From Home          0.084156
Shares             0.078471
Comments           0.074952
Follows            0.071203
Hashtags           0.066235
Caption            0.063629
dtype: float64
```

After the all Feature Selection method use then we decide to we can't drop any features because all are impotrtant fetures.

Here, we can see that in traing dataset model performing well but in testing dataset model performing not well and it model performing overfiting. So, we need to do over sampling for model performing well in test dataset.

Start Whole Process of Model training in second time for Over Sampling

In [77]:
```
X = df1.drop(['Impressions'], axis=1)
Y = df1["Impressions"]
```

In [78]:
```
X
```

Out[78]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Ca
0	2586	1028	619	56	98	9	5	162	35	2	Her so the impo vis
1	2727	1838	1174	78	194	7	14	224	48	10	Her so the sc proj
2	2085	1188	0	533	41	11	1	131	62	12	Learr to tr ma lea model
3	2700	621	932	73	172	10	7	213	23	8	He hov can w Py progra
4	1704	255	279	37	96	5	4	123	8	0	Pl annota visua your
...	...	...	...	...	...	...	...	...	...	...	
114	5185	3041	5352	77	573	2	38	373	73	80	Her so the sc ce
115	1923	1368	2266	65	135	4	1	148	20	18	Clust ma lea techı
116	1133	1538	1367	33	36	0	1	92	34	10	Clust r genre ta groupi
117	11815	3147	17414	170	1095	2	75	549	148	214	Her so the sc ce

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Ca
118	13473	4176	16444	2547	653	5	26	443	611	228	175 P Pro S solved

119 rows × 12 columns

In [79]:
```
cat = X.select_dtypes(include="object")
num = X.select_dtypes(include="number")
```

In [80]:
```
encoder.fit(cat,Y)
cat = encoder.transform(cat)
```

In [81]:
```
cat
```

Out[81]:

	Caption	Hashtags
0	5703.991597	5703.991597
1	6097.333333	6097.333333
2	5703.991597	5703.991597
3	5703.991597	10246.875000
4	5703.991597	5703.991597
...	...	...
114	23197.500000	15853.750000
115	5703.991597	4935.000000
116	5703.991597	4935.000000
117	23197.500000	15853.750000
118	5703.991597	10246.875000

119 rows × 2 columns

In [82]:
```
X = pd.concat([num, cat], axis=1)
```

we converted categorical feature to numerical because over sampling technique performing on numerical value.

Over Sampling

In [83]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [84]:
```
ros = RandomOverSampler(random_state=50)
xos, yos = ros.fit_resample(X, Y)
```

In [85]:
```
xos
```

Out[85]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	
0	2586	1028	619	56	98	9	5	162	35	2	5703
1	2727	1838	1174	78	194	7	14	224	48	10	6097
2	2085	1188	0	533	41	11	1	131	62	12	5703
3	2700	621	932	73	172	10	7	213	23	8	5703
4	1704	255	279	37	96	5	4	123	8	0	5703
...	...	...	...	...	...	...	...	...	...	...	
298	2449	2141	12389	561	504	3	23	308	70	96	5703
299	11815	3147	17414	170	1095	2	75	549	148	214	23197
300	11815	3147	17414	170	1095	2	75	549	148	214	23197
301	13473	4176	16444	2547	653	5	26	443	611	228	5703
302	13473	4176	16444	2547	653	5	26	443	611	228	5703

303 rows × 12 columns

In [86]:
```
train_x11, test_x11, train_y11, test_y11 = train_test_split(xos,yos, random_state=50, test_size=0.2)
```

In [87]:
```
scaler.fit(train_x11)
train_x11 = pd.DataFrame(scaler.transform(train_x11), columns=train_x11.columns)
test_x11 = pd.DataFrame(scaler.transform(test_x11), columns=test_x11.columns)
```

In [88]:
```
test_x11
```

Out[88]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits
0	0.267909	0.053927	0.330883	0.017336	0.233924	0.210526	0.173333	0.289308	0.093904
1	0.042950	0.099821	0.009533	0.088652	0.014911	0.000000	0.000000	0.073375	0.067545
2	0.156564	0.097257	0.119502	0.291174	0.135135	0.368421	0.120000	0.257862	0.230643
3	0.026985	0.025212	0.002125	0.003152	0.025163	0.315789	0.040000	0.020964	0.006590
4	0.112804	0.485685	0.011944	0.309299	0.072693	0.315789	0.133333	0.465409	0.291598
...	...	...	...	...	...	...	...	...	...
56	0.163614	0.551064	0.035431	0.070134	0.183597	0.684211	0.200000	0.471698	0.497529
57	0.071637	0.191009	0.017113	0.039007	0.073625	0.368421	0.146667	0.182390	0.021417
58	0.099190	0.027861	0.014356	0.207644	0.087605	0.315789	0.000000	0.129979	0.034596
59	0.014182	0.059226	0.002584	0.003546	0.012116	0.105263	0.013333	0.000000	0.023064
60	0.124797	0.161268	0.024865	0.041371	0.275862	0.473684	0.293333	0.373166	0.034596

61 rows × 12 columns

In [89]:
```
test_y11
```

Out[89]:
```
112    11149
63      3454
270     7281
39      1941
277     9453
       ...
283    10667
15      5055
175     3818
124     2191
81      5273
Name: Impressions, Length: 61, dtype: int64
```

In [90]:
```
#KNeighborsRegressor
knn1 = KNeighborsRegressor(algorithm="auto", n_neighbors=1, weights="uniform")
knn1.fit(train_x11, train_y11)
pred22 = knn1.predict(test_x11)
mea_knn1 = mean_absolute_error(test_y11,pred22)
mea_knn1
```

Out[90]:
```
35.01639344262295
```

In [91]:
```
knn1.score(train_x11, train_y11)
```

Out[91]:
```
1.0
```

In [92]:
```
knn1.score(test_x11, test_y11)
```

Out[92]: 0.9973244076465144

In [93]:
```
#SVR
svm1 = SVR(gamma="scale")
svm1.fit(train_x11, train_y11)
pred44 = svm1.predict(test_x11)
mea_svm = mean_absolute_error(test_y11, pred44)
mea_svm
```

Out[93]: 1952.059779462308

In [94]:
```
svm1.score(train_x11, train_y11)
```

Out[94]: -0.09349293658436397

In [95]:
```
svm1.score(test_x11, test_y11)
```

Out[95]: -0.09492643913092946

In [96]:
```
#DecisionTreeRegressor
dt1 = DecisionTreeRegressor( max_depth=45, splitter="best")
dt1.fit(train_x11, train_y11)
pred55 = dt1.predict(test_x11)
mea_dt = mean_absolute_error(test_y11, pred55)
mea_dt
```

Out[96]: 0.14754098360655737

In [97]:
```
dt1.score(train_x11, train_y11)
```

Out[97]: 1.0

In [98]:
```
dt1.score(test_x11, test_y11)
```

Out[98]: 0.9999999524990311

In [99]:
```
#RandomForestRegressor
rfc1 = RandomForestRegressor(max_depth=48 ,n_estimators= 8)
rfc1.fit(train_x11, train_y11)
pred66 = rfc1.predict(test_x11)
mea_rfc = mean_absolute_error(test_y11, pred66)
mea_rfc
```

Out[99]: 146.58401639344262

In [100]:
```
rfc1.score(train_x11, train_y11)
```

Out[100]: 0.9858166809317759

In [101]:
```
rfc1.score(test_x11, test_y11)
```

Out[101]: 0.9935459051125513

In [102]:
```
#XGBRegressor
xgb = Xgb.XGBRegressor(eta=0.3 ,max_depth=6 ,n_estimators= 10)
xgb.fit(train_x11, train_y11)
pred7 = xgb.predict(test_x11)
mea_xgb = mean_absolute_error(test_y11, pred7)
mea_xgb
```

Out[102]: 206.6979820376537

In [103]:
```
xgb.score(train_x11, train_y11)
```

Out[103]: 0.993151427732353

In [104]:
```
xgb.score(test_x11, test_y11)
```

Out[104]: 0.9919002000972295

In [105]:
```
#Score for all model
print("KNeighborsRegressor score is :", mea_knn1)
print("Support vector machine score is :", mea_svm)
print("DecisionTreeRegressor score is :", mea_dt)
print("RandomForestRegressor score is :", mea_rfc)
print("XGBOOST score is :", mea_xgb)
```

```
KNeighborsRegressor score is : 35.01639344262295
Support vector machine score is : 1952.059779462308
DecisionTreeRegressor score is : 0.14754098360655737
RandomForestRegressor score is : 146.58401639344262
XGBOOST score is : 206.6979820376537
```

Here, All Model give best score but Decision Tree Regressor give best score in train and test data as well as mean absolute error also 0.1475409860655737 compare to other model.

Testing the new data

In [106]:
```
new_df = {'From Home' : 3500, 'From Hashtags': 1500, 'From Explore': 690, 'Fro
m Other': 70, 'Saves' : 100,
          'Comments' : 6, 'Shares' : 3, 'Likes': 250, 'Profile Visits': 45, 'F
ollows' : 8,
          'Caption' : 'Plotting annotations while visualizing your data is con
sidered good practice to make the graphs self-explanatory. Here is an example
of how you can annotate a graph using Python.',
          'Hashtags' : '#healthcare\xa0#health\xa0#covid\xa0#data\xa0#datascie
nce\xa0#dataanalysis\xa0#dataanalytics\xa0#datascientist\xa0#machinelearning\x
a0#python\xa0#pythonprogramming\xa0#pythonprojects\xa0#pythoncode\xa0#artifici
alintelligence\xa0#ai\xa0#dataanalyst\xa0#amankharwal\xa0#thecleverprogramme
r'}
index = [0]
```

In [107]:
```
new_df = pd.DataFrame(new_df,index=index)
```

In [108]:
```
new_df
```

Out[108]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Capti
0	3500	1500	690	70	100	6	3	250	45	8	Plott annotatic wh visualiz your da

In [109]:
```
new_cat = new_df.select_dtypes(include="object")
new_num = new_df.select_dtypes(include="number")
```

In [110]:
```
new_cat = encoder.transform(new_cat)
```

In [111]:
```
new_cat
```

Out[111]:

	Caption	Hashtags
0	5703.991597	6097.333333

In [112]:
```
new_df = pd.concat([new_num, new_cat], axis=1)
```

In [113]:
```
new_df
```

Out[113]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Cap
0	3500	1500	690	70	100	6	3	250	45	8	5703.991

In [114]:
```
new_df =pd.DataFrame(scaler.transform(new_df), columns=new_df.columns)
```

In [115]:
```
new_df
```

Out[115]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Fo
0	0.191815	0.11828	0.039623	0.024035	0.072693	0.315789	0.04	0.373166	0.067545	0.0

In [116]:
```
Pred_new = dt1.predict(new_df)
```

In [117]:
```
Pred_new
```

Out[117]:
```
array([4528.])
```

In [118]:
```
new_df1 = {'From Home' : 2500, 'From Hashtags': 1000, 'From Explore': 450, 'Fr
om Other': 55, 'Saves' : 85,
          'Comments' : 6, 'Shares' : 3, 'Likes': 200, 'Profile Visits': 56, 'F
ollows' : 8,
          'Caption' : 'Plotting annotations while visualizing your data is con
sidered good practice to make the graphs self-explanatory. Here is an example
of how you can annotate a graph using Python.',
          'Hashtags' : '#healthcare\xa0#health\xa0#covid\xa0#data\xa0#datascie
nce\xa0#dataanalysis\xa0#dataanalytics\xa0#datascientist\xa0#machinelearning\x
a0#python\xa0#pythonprogramming\xa0#pythonprojects\xa0#pythoncode\xa0#artifici
alintelligence\xa0#ai\xa0#dataanalyst\xa0#amankharwal\xa0#thecleverprogramme
r'}
index = [0]
```

In [119]:
```
new_df1 = pd.DataFrame(new_df1, index=index)
```

In [120]:
```
new_df1
```

Out[120]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Capti
0	2500	1000	450	55	85	6	3	200	56	8	Plott annotatic wh visualiz your da

In [121]:
```
new_cat = new_df1.select_dtypes(include="object")
new_num = new_df1.select_dtypes(include="number")
```

In [122]:
```
new_cat = encoder.transform(new_cat)
```

In [123]:
```
new_df1 = pd.concat([new_num, new_cat], axis=1)
```

In [124]:
```
new_df1
```

Out[124]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Follows	Cap
0	2500	1000	450	55	85	6	3	200	56	8	5703.991

In [125]:
```
new_df1 =pd.DataFrame(scaler.transform(new_df1), columns=new_df1.columns)
```

In [126]:
```
new_df1
```

Out[126]:

	From Home	From Hashtags	From Explore	From Other	Saves	Comments	Shares	Likes	Profile Visits	Fo
0	0.110778	0.075549	0.025841	0.018125	0.058714	0.315789	0.04	0.268344	0.085667	0.0

In [127]:
```
pred_new1 = dt1.predict(new_df1)
```

In [128]:
```
pred_new1
```

Out[128]:
```
array([3749.])
```

In [129]:
```
print('DecisionTreeRegressor Score is ', mean_absolute_error(test_y11, pred5
5))
print('DecisionTreeRegressor Training Score is ', dt1.score(train_x11, train_y
11))
print('DecisionTreeRegressor Testing Score is ', dt1.score(test_x11, test_y1
1))
```

```
DecisionTreeRegressor Score is  0.14754098360655737
DecisionTreeRegressor Training Score is  1.0
DecisionTreeRegressor Testing Score is  0.9999999524990311
```

CONCLUSION : From the above all Different Model Decision Tree Regressor have generated the model with higher accuracy in both defulat model and Hyperperameter tuning. In this Model have no correlation with each other. But there is small dataset and model predict overfiting so we do over sampling for reduce overfiting. So, we get best score. All model give best score but Decision Tree Regressor score is : 0.14754098360655737 Training Score is 1.0 Testing Score is 0.9999999524990311