INTRODUCTION TO STATISTICAL LEARNING

Fall 2019 Semester

Project Report

# Facebook Metrics Dataset

Submitted By

Tejaswini Rayapati
Geetanjali Makineni

Department of Computer Science and Electrical Engineering
University of Missouri Kansas City

# Objective: The main objective is to perform analysis on Facebook metrics which comprises the data on the Facebook's page of a renowned cosmetics brand. To find whether the data set is rich enough to predicate likes.

# Steps involved are:

1. Loading and viewing dataset
2. Pre-processing
3. Exploratory Data Analysis
4. Modeling

# Description of Dataset:

Facebook Metrics data set contains 500 of the 790 rows and part of the features analyzed by Moro et al. The dataset contains 19 features. The aim is to use 19 features to predict responses.

# Loading and viewing dataset:

We need to import all the required packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


df = pd.read_csv("dataset_Facebook.csv",delimiter=';')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 19 columns):
Page total likes                                                    500 non-null int64
Type                                                                500 non-null object
Category                                                            500 non-null int64
Post Month                                                          500 non-null int64
Post Weekday                                                        500 non-null int64
Post Hour                                                           500 non-null int64
Paid                                                                499 non-null float64
Lifetime Post Total Reach                                           500 non-null int64
Lifetime Post Total Impressions                                     500 non-null int64
Lifetime Engaged Users                                              500 non-null int64
Lifetime Post Consumers                                             500 non-null int64
Lifetime Post Consumptions                                          500 non-null int64
Lifetime Post Impressions by people who have liked your Page        500 non-null int64
Lifetime Post reach by people who like your Page                    500 non-null int64
Lifetime People who have liked your Page and engaged with your post 500 non-null int64
comment                                                             500 non-null int64
like                                                                499 non-null float64
share                                                               496 non-null float64
Total Interactions                                                  500 non-null int64
dtypes: float64(3), int64(15), object(1)
memory usage: 74.3+ KB
```

df.head(3)

| | Page total likes | Type | Category | Post Month | Post Weekday | Post Hour | Paid | Lifetime Post Total Reach | Lifetime Post Total Impressions | Lifetime Engaged Users | Lifetime Post Consumers | Li Consum |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 139441 | Photo | 2 | 12 | 4 | 3 | 0.0 | 2752 | 5091 | 178 | 109 | |
| 1 | 139441 | Status | 2 | 12 | 3 | 10 | 0.0 | 10460 | 19057 | 1457 | 1361 | |
| 2 | 139441 | Photo | 3 | 12 | 3 | 3 | 0.0 | 2413 | 4373 | 177 | 113 | |

Because columns 7 to 15 (like "Lifetime Engaged users", etc.) are recorded after posting, they will not be used for modeling. However, they can give useful information about post reach, and we will be looking at them in EDA.

Goal – Predict interactions based on features

We need to fill the values that are with blank with 0

```
df['like'].describe()
```

```
count     499.000000
mean      177.945892
std       323.398742
min         0.000000
25%        56.500000
50%       101.000000
75%       187.500000
max      5172.000000
Name: like, dtype: float64
```

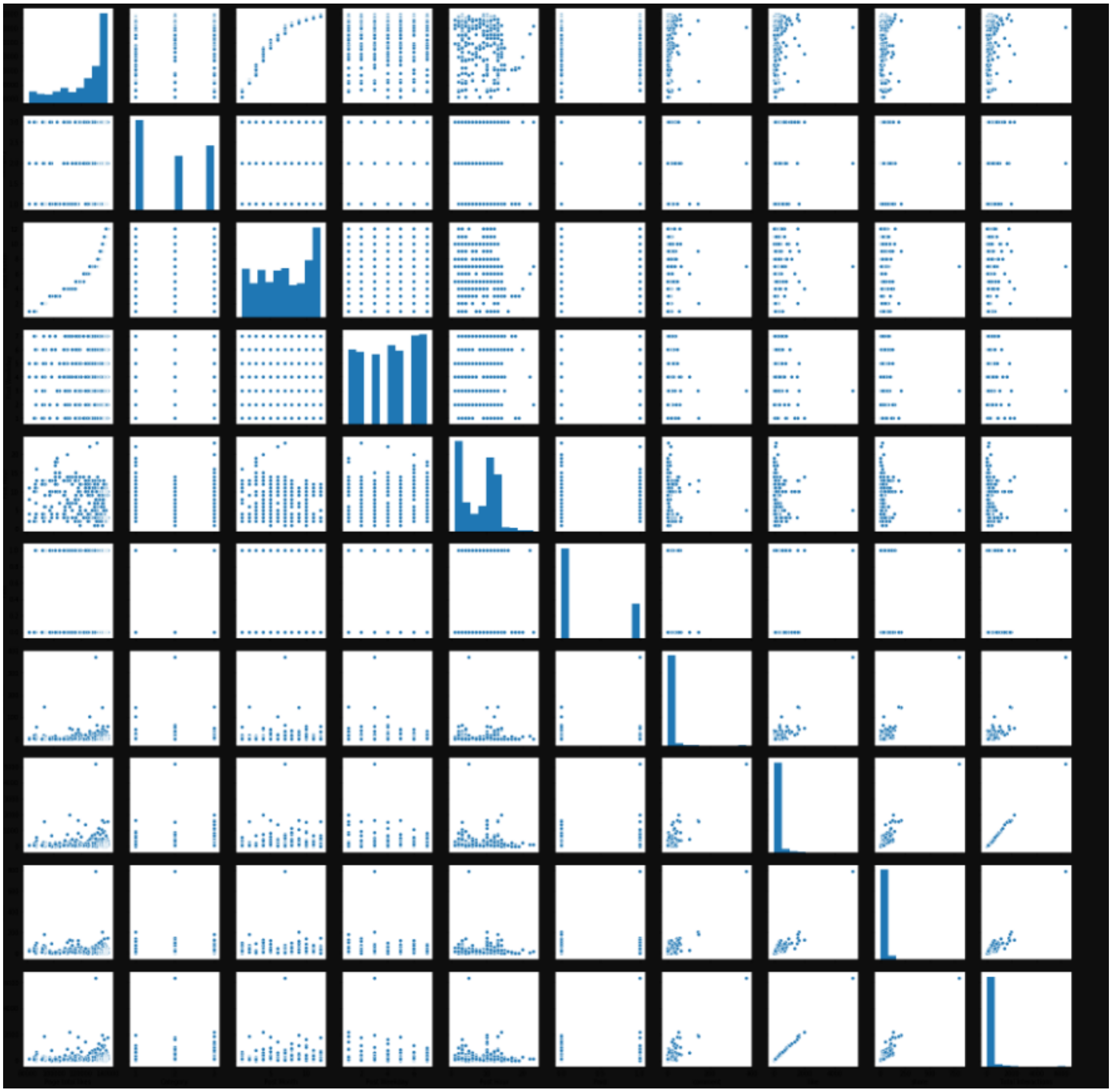
```
df['Type'].value_counts()
```

```
Photo     426
Status     45
Link       22
Video       7
Name: Type, dtype: int64
```

```
[ ]  df['Paid'].value_counts()
     df['like'].fillna(0,inplace=True)
     df['share'].fillna(0,inplace=True)
     df['Paid'].fillna(0,inplace=True)
```

EDA- Exploratory Data Analysis

```
[ ]  df['Type'] = df['Type'].apply(lambda x: str(x))
     plotdf = df.drop(df.columns[7:15],axis =1)
     sns.pairplot(data=plotdf)
```



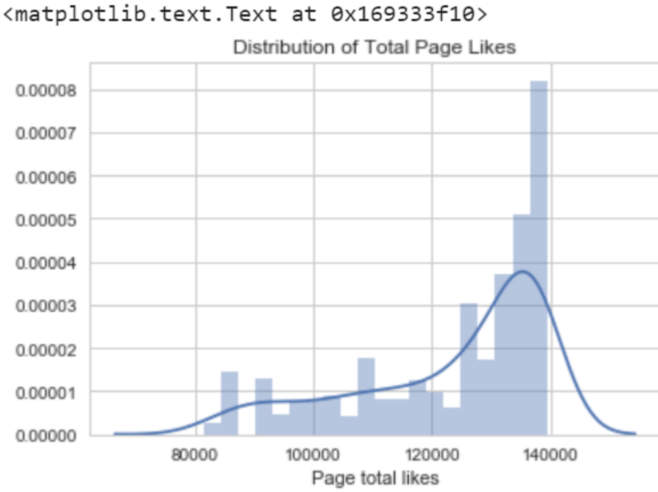
All the rows and columns will be in metric form, to find correlation coefficient the value lies between -1 to 1. If the value is between 0.5 to 1 then it has good correlation.

```
[ ] plt.figure(figsize=(12,10))
    sns.heatmap(df.corr(),cmap='viridis',annot=True,cbar=False)
```


Comparison of Page Likes and Post Likes

```
[ ] sns.distplot(df['Page total likes'],bins=20)
    plt.title("Distribution of Total Page Likes")
```

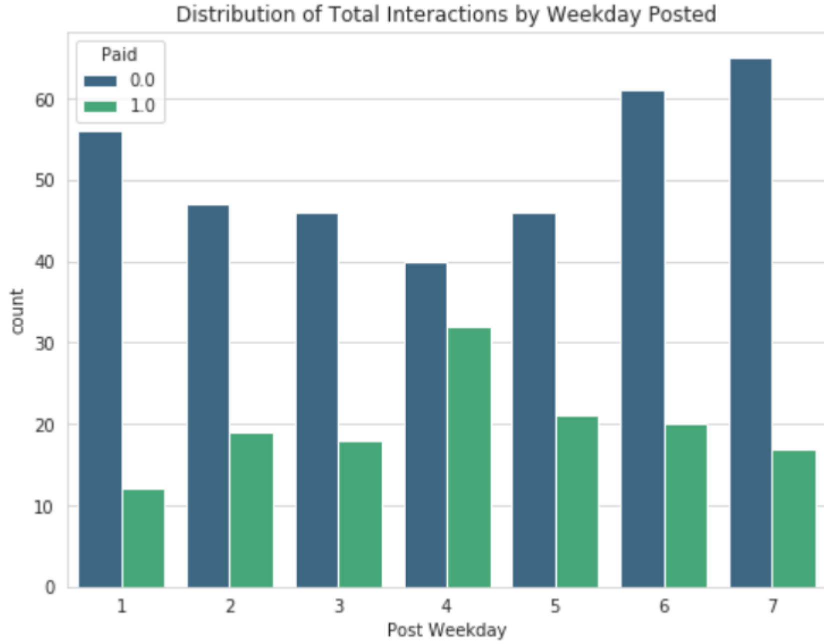
```
<matplotlib.text.Text at 0x169333f10>
```




The Data displayed for total week.

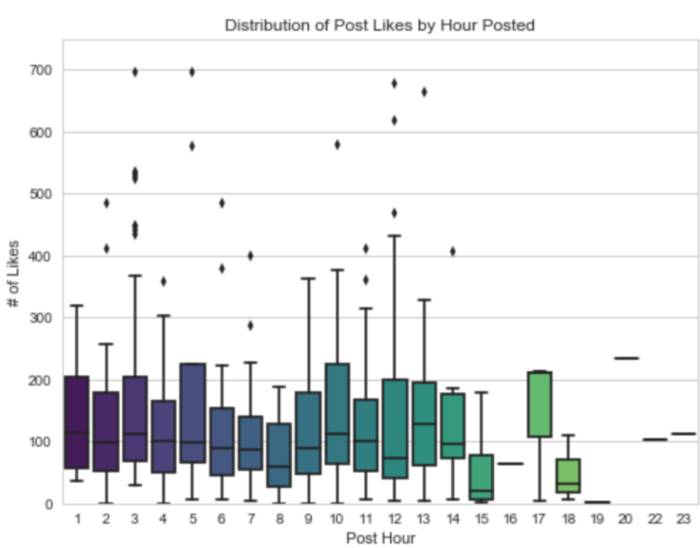
```
[ ] plt.figure(figsize=(8,6))
    sns.countplot(x='Post Weekday',hue='Paid',data=df,palette='viridis')
    plt.title("Distribution of Total Interactions by Weekday Posted")
```

```
Text(0.5,1,'Distribution of Total Interactions by Weekday Posted')
```




The Data displayed for every hour

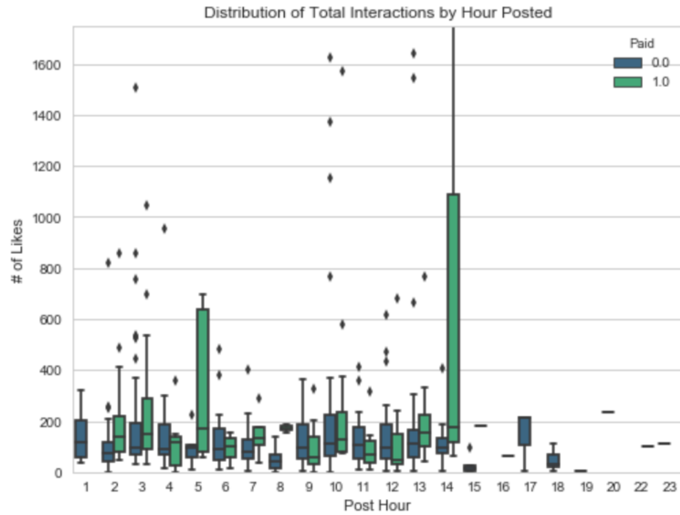
```
[ ] plt.figure(figsize=(8,6))
    sns.boxplot(x='Post Hour',y='like',data=df,palette='viridis')
    plt.ylim(0,750)
    plt.title("Distribution of Post Likes by Hour Posted")
    plt.ylabel("# of Likes")
    plt.savefig('hourBox.png', bbox_inches='tight')
```



Data in form of mean median, graph is displayed for every hour and paid, and unpaid posts are divided

```
[ ]  plt.figure(figsize=(8,6))
     sns.boxplot(x='Post Hour',y='like',hue='Paid',data=df,palette='viridis')
     plt.ylim(0,1750)
     plt.title("Distribution of Total Interactions by Hour Posted")
     plt.ylabel("# of Likes")
```

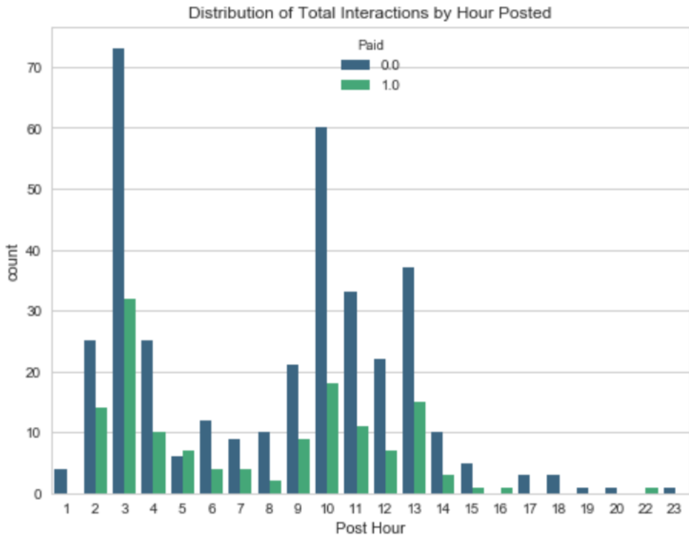
```
<matplotlib.text.Text at 0x130e8e990>
```




Data in form of values, graph is displayed for every hour and paid, and unpaid posts are divided

```
[ ]  plt.figure(figsize=(8,6))
     sns.countplot(x='Post Hour',hue='Paid',data=df,palette='viridis')
     #plt.ylim(0,2200)
     plt.title("Distribution of Total Interactions by Hour Posted")
```

<matplotlib.text.Text at 0x12af34850>

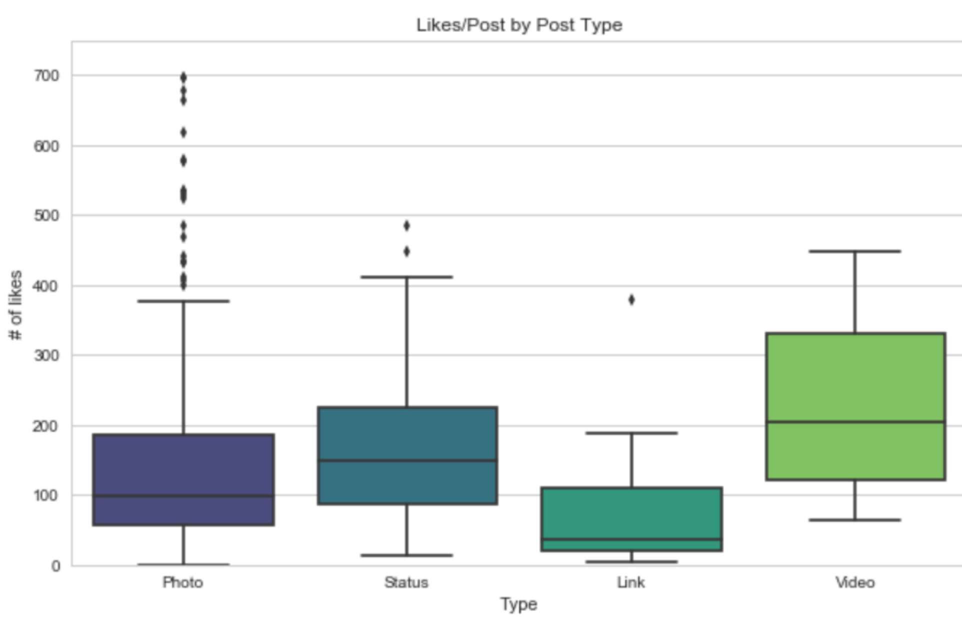



Post Type Vs Likes:

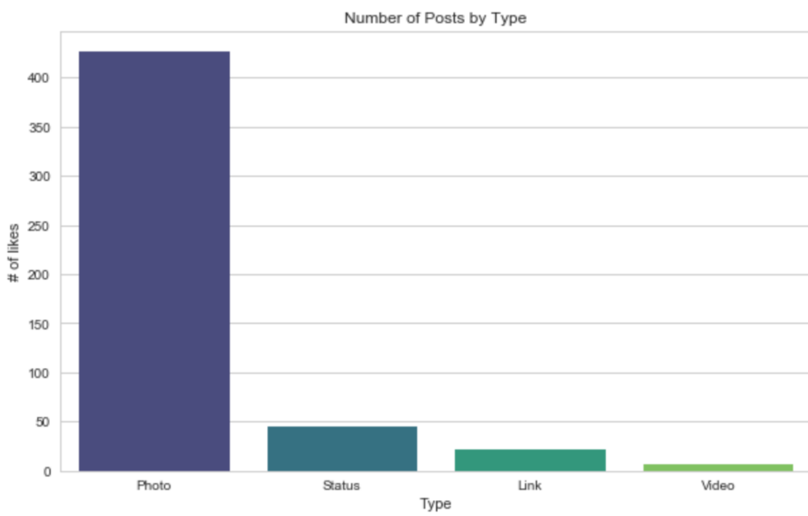
As like said there are four types of posts i.e., Photo, Status, Link, Video

Here we find the dependency of posts on likes

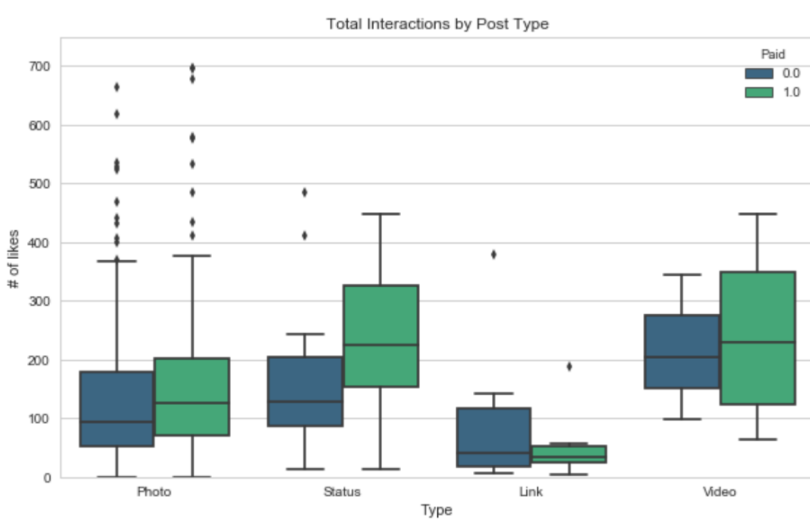
```
[ ]  plt.figure(figsize=(10,6))
     sns.boxplot(x='Type',y='like',data=df,palette='viridis')
     plt.ylim(0,750)
     #sns.despine(offset=4,bottom=True)
     plt.title("Likes/Post by Post Type")
     plt.ylabel("# of likes")
     #plt.legend(loc='upper left')
     plt.savefig('typeBox.png', bbox_inches='tight')
```



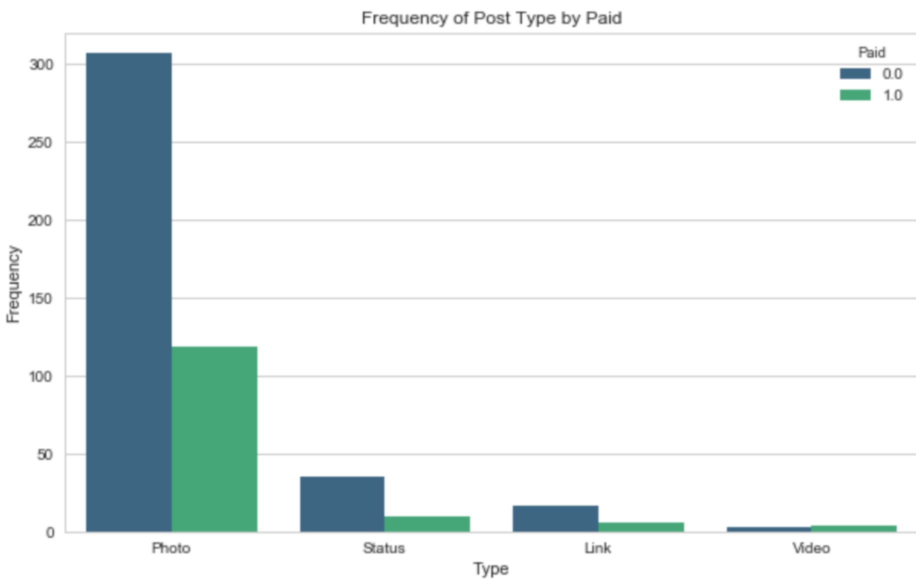
```
[ ]  plt.figure(figsize=(10,6))
     sns.countplot(x='Type',data=df,palette='viridis')
     #plt.ylim(0,750)
     #sns.despine(offset=4,bottom=True)
     plt.title("Number of Posts by Type")
     plt.ylabel("# of likes")
     #plt.legend(loc='upper left')
     plt.savefig('typeCount.png', bbox_inches='tight')
```




```
[ ]  plt.figure(figsize=(10,6))
     sns.boxplot(x='Type',y='like',hue='Paid',data=df,palette='viridis')
     plt.ylim(0,750)
     #sns.despine(offset=4,bottom=True)
     plt.title("Total Interactions by Post Type")
     plt.ylabel("# of likes")
     plt.savefig('typePaidBox.png', bbox_inches='tight')
```



```
[ ] plt.figure(figsize=(10,6))
    sns.countplot(x='Type',hue='Paid',data=df,palette='viridis')
    plt.ylim(0,320)
    #sns.despine(offset=4,bottom=True)
    plt.title("Frequency of Post Type by Paid")
    plt.ylabel("Frequency")
    plt.savefig('typePaidCount.png', bbox_inches='tight')
```




Video posts had the highest mean, median, and percentiles. Photo posts had the largest range

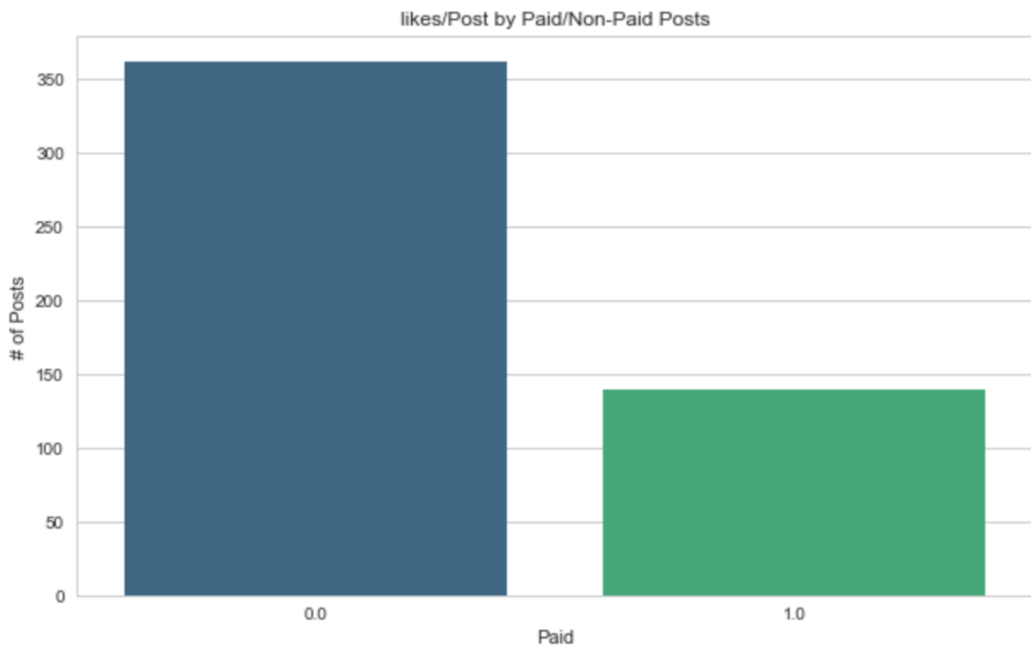
Observations:

- On average video posts had higher engagement
- Photo posts had the largest range
- This suggests that total interactions can depend on the photo posted
- Links performed the worst, with the lowest mean, range, and median
- No difference in paid link posts

```
[ ] from numpy import median
    print(median(df[df['Paid']==0]['like']))
    print(median(df[df['Paid']==1]['like']))
```

```
96.0
128.0
```

```
[ ] plt.figure(figsize=(10,6))
    sns.countplot(x='Paid',data=df,palette='viridis')
    #sns.despine(offset=4,bottom=True)
    plt.title("likes/Post by Paid/Non-Paid Posts")
    plt.ylabel("# of Posts")
    plt.savefig('paidCount.png', bbox_inches='tight')
```




Observations:

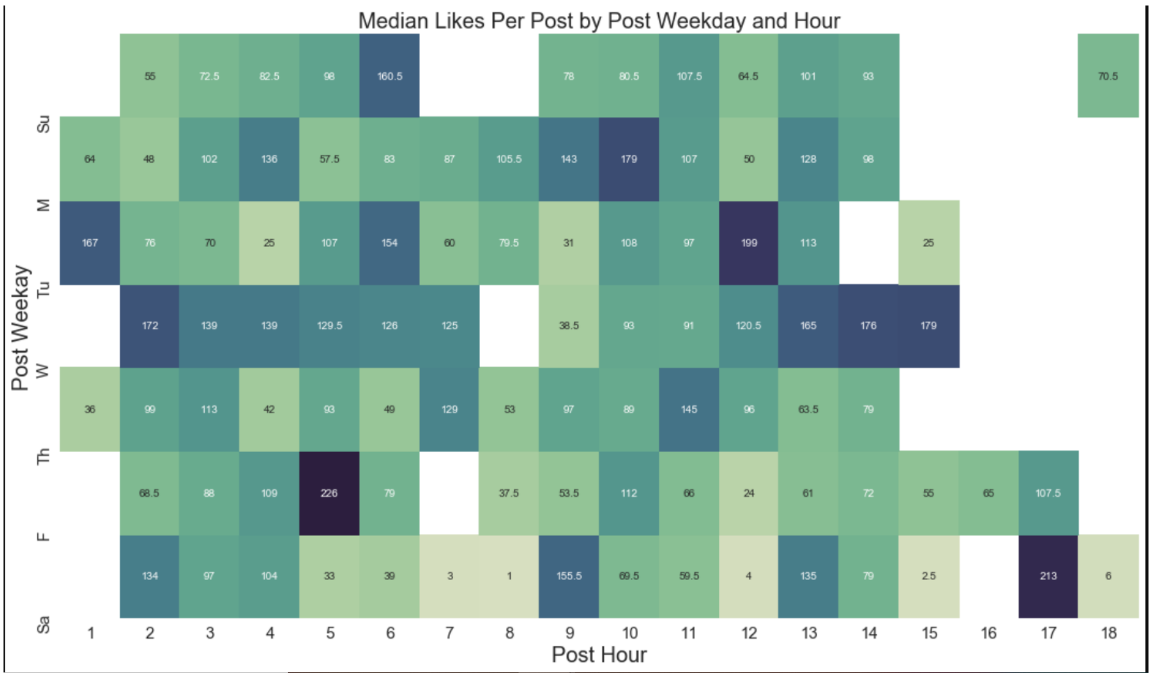
- There were more non-paid posts than paid posts
- Paid posts performed 30 likes by median better than non-paid posts

```
[ ] df.columns
```

```
Index([u'Page total likes', u'Type', u'Category', u'Post Month',
       u'Post Weekday', u'Post Hour', u'Paid', u'comment', u'like', u'share',
       u'Total Interactions', u'Video', u'Status', u'Photo', u'Cat_1',
       u'Cat_2'],
      dtype='object')
```

```
[ ] timePivot = pd.pivot_table(df,aggfunc='median',
                              columns='Post Hour',
                              index='Post Weekday',
                              values='like')
    timePivot = timePivot[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]]
```

```
[ ] plt.figure(figsize=(18,10))
    cmap= sns.cubehelix_palette(8, start=.5, rot=-.75,as_cmap=True)
    sns.heatmap(timePivot,cbar=False,cmap=cmap,annot=True, fmt='g')
    #plt.pcolor(lnch_pivot,cmap=plt.cm.Blues, alpha=0.8)
    plt.yticks(np.arange(7),['Sa','F','Th','W','Tu','M','Su'],fontsize=15)
    plt.xticks(fontsize=15)
    plt.ylabel('Post Weekay',fontsize=20)
    plt.xlabel('Post Hour',fontsize=20)
    plt.title('Median Likes Per Post by Post Weekday and Hour',fontsize=20)
    plt.savefig('medianLikeHeatmap.png', bbox_inches='tight')
```




Modeling:

```
[ ] from sklearn import linear_model
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import r2_score
    from sklearn.preprocessing import StandardScaler
```

To remove outliners, I will remove nay variables that are above 90$^{th}$ percentile

To avoid multicollinearity, I will be taking n-1 columns for each feature.

```
[ ] outlierCut = np.percentile(df['like'],90)
    outlierCut
```

```
330.10000000000002
```

```
[ ]  df = df[df['like']<outlierCut]
```

The function below will translate the weekdays to their labels, rather than 1-7.

```
[ ]  def Weekday(x):
         if x == 1:
             return 'Su'
         elif x== 2:
             return 'Mo'
         elif x == 3:
             return 'Tu'
         elif x == 4:
             return 'We'
         elif x == 5:
             return 'Th'
         elif x ==6:
             return 'Fr'
         elif x == 7:
             return "Sa"

     df['Weekday'] = df['Post Weekday'].apply(lambda x: Weekday(x))
```

```
[ ]  dayDf = pd.get_dummies(df['Weekday'])
```

```
[ ]  df = pd.concat([df,dayDf],axis=1)
```

```
[ ]  hours = list(range(0,18))
     #hours
     for i in hours:
         hours[i] = str(hours[i])
         hours[i]='hr_'+ hours[i]
         #print str(hours[i])
```

```
[ ]  hourDf = pd.get_dummies(df['Post Hour'],prefix='hr_')
     df = pd.concat([df,hourDf],axis=1)
     monthDf = pd.get_dummies(df['Post Month'],prefix='Mo')
     df = pd.concat([df,monthDf],axis=1)
     df['Video'] = pd.get_dummies(df['Type'])['Video']
     df['Status'] = pd.get_dummies(df['Type'])['Status']
     df['Photo'] = pd.get_dummies(df['Type'])['Photo']
     df['Cat_1'] = pd.get_dummies(df['Category'])[1]
     df['Cat_2'] = pd.get_dummies(df['Category'])[2]
     #To avoid multicollinearity with the post types I am not including Links.
```

```
df.head()
```

| | Page total likes | Type | Category | Post Month | Post Weekday | Post Hour | Paid | Lifetime Post Total Reach | Lifetime Post Total Impressions | Lifetime Engaged Users | ... | Mo_8 | Mo_9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 139441 | Photo | 2 | 12 | 4 | 3 | 0.0 | 2752 | 5091 | 178 | ... | 0 | 0 |
| **1** | 139441 | Status | 2 | 12 | 3 | 10 | 0.0 | 10460 | 19057 | 1457 | ... | 0 | 0 |
| **2** | 139441 | Photo | 3 | 12 | 3 | 3 | 0.0 | 2413 | 4373 | 177 | ... | 0 | 0 |
| **3** | 139441 | Photo | 2 | 12 | 2 | 10 | 1.0 | 50128 | 87991 | 2211 | ... | 0 | 0 |
| **4** | 139441 | Photo | 2 | 12 | 2 | 3 | 0.0 | 7244 | 13594 | 671 | ... | 0 | 0 |

5 rows × 66 columns

Train Test Split:

```
[ ] #
    x = df[['Page total likes','Paid','Video','Status','Photo',
        'Cat_1','Cat_2','Mo','Tu','Sa',"We",'Th','Fr',
           'hr__17','hr__1','hr__2','hr__3','hr__4','hr__5', 'hr__6','hr__7','hr__8',
            'hr__9','hr__10','hr__11','hr__12','hr__13','hr__14','hr__15','hr__16','Mo_1',
           'Mo_2','Mo_12','Mo_4','Mo_5','Mo_6','Mo_7','Mo_8','Mo_9','Mo_11','Mo_10']]
    y = df['like']
```

```
[ ] x_train,x_test,y_train, y_test = train_test_split(x,
                                                  y, test_size=0.1,
                                                  random_state=42)
```

```
[ ] y_test.count()
```

```
50
```

```
[ ] x_test.columns
```

```
Index([u'Page total likes', u'Paid', u'Video', u'Status', u'Photo', u'Cat_1',
       u'Cat_2', u'Mo', u'Tu', u'Sa', u'We', u'Th', u'Fr', u'hr__17', u'hr__1',
       u'hr__2', u'hr__3', u'hr__4', u'hr__5', u'hr__6', u'hr__7', u'hr__8',
       u'hr__9', u'hr__10', u'hr__11', u'hr__12', u'hr__13', u'hr__14',
       u'hr__15', u'hr__16', u'Mo_1', u'Mo_2', u'Mo_12', u'Mo_4', u'Mo_5',
       u'Mo_6', u'Mo_7', u'Mo_8', u'Mo_9', u'Mo_11', u'Mo_10'],
      dtype='object')
```

Linear and Lasso Regression:

```
[ ] reg = linear_model.LinearRegression(normalize=True)
    lasso = linear_model.Lasso(normalize=True)
    reg.fit(x_train,y_train)
    lasso.fit(x_train,y_train)
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=True, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

```
[ ] reg.coef_
```

```
array([ 1.34914696e-02,  6.75973028e+01,  1.03332565e+02,
        1.52291213e+01,  4.16422561e+01, -8.37494900e+01,
       -2.45355283e+01, -5.49602040e+01,  2.28616762e+01,
       -9.40413182e+01, -3.61694842e+01, -4.21830055e+01,
       -5.82335606e+01,  1.78264974e+02, -1.71968886e+01,
       -1.91943455e+00,  1.98051523e+01, -3.85992229e+00,
        4.00480242e+02, -8.09779829e+00, -3.74563671e+01,
       -7.19084877e+01, -2.30154165e+01,  7.08546648e+01,
       -3.17929722e+01,  2.56306438e+01,  5.54397700e+01,
        1.63793523e+02,  1.78423586e+01,  8.87908390e-14,
        2.79184527e+02,  2.16715932e+02, -3.99393906e+02,
       -5.21359759e+01, -9.83820752e+01, -2.84032713e+02,
       -2.34626855e+02, -3.23320344e+02, -3.15195609e+02,
       -3.92223835e+02, -3.80306626e+02])
```

```
[ ] lasso.coef_
```

```
array([  0.        ,  28.39789747,   0.        ,  -0.        ,
         0.        , -48.92957062,   0.        ,  -0.        ,
        15.54410522,  -2.62262774,   0.        ,  -0.        ,
        -0.        ,   0.        ,  -0.        ,  -0.        ,
        -0.        ,  -0.        , 253.05635563,  -0.        ,
        -0.        ,  -0.        ,  -0.        ,   3.71844117,
        -0.        ,  -0.        ,   0.        ,   0.        ,
        -0.        ,   0.        ,  -0.        ,   0.        ,
         0.        ,   0.        ,   0.        ,  -0.        ,
        11.36245645,   0.        ,   0.        ,   0.        ,  -0.        ])
```

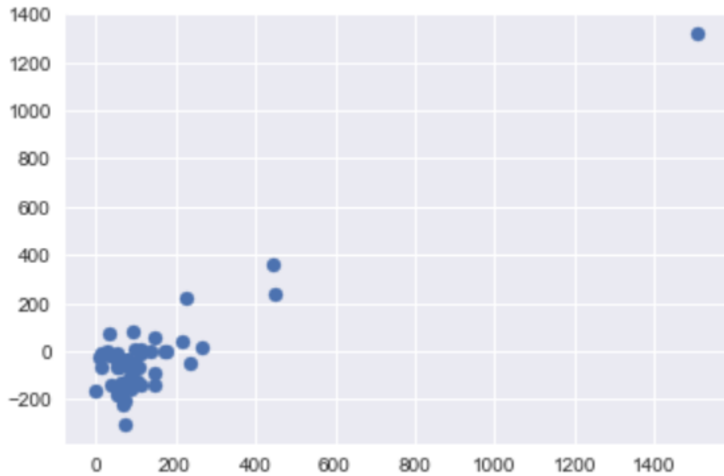
Model Validation:

```
[ ] pred = reg.predict(x_test)
    pred_train = reg.predict(x_train)

    lpred = lasso.predict(x_test)
    lpred_train = lasso.predict(x_train)
```

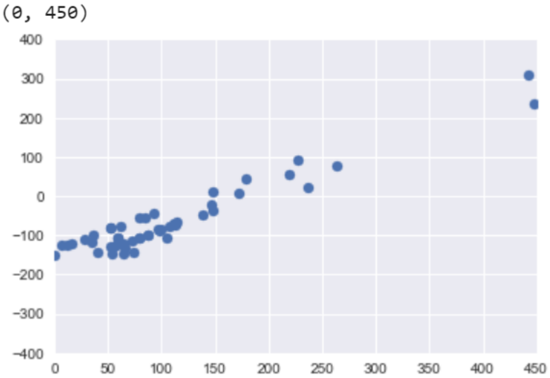
```
[ ] LRerror = y_test - pred
    plt.scatter(y_test, LRerror)
    #plt.ylim(-400,400)
    #plt.xlim(0,450)
```

```
<matplotlib.collections.PathCollection at 0x118745510>
```



```
[ ] LSerror = y_test - lpred
    plt.scatter(y_test, LSerror)
    plt.ylim(-400,400)
    plt.xlim(0,450)
```

```
(0, 450)
```



Find R2 value - R-squared (statistical measure of how close the data are to the fitted regression line.)

```
[ ] testScore = r2_score(y_pred=pred,y_true=y_test)
    trainScore = r2_score(y_pred=pred_train,y_true=y_train)

    ltestScore = r2_score(y_pred=lpred,y_true=y_test)
    ltrainScore = r2_score(y_pred=lpred_train,y_true=y_train)
```

```
[ ] lrResults = pd.DataFrame()
    lrResults['Score'] = [trainScore,testScore]
    lrResults['Step'] = ['train','test']

    lrResults
```

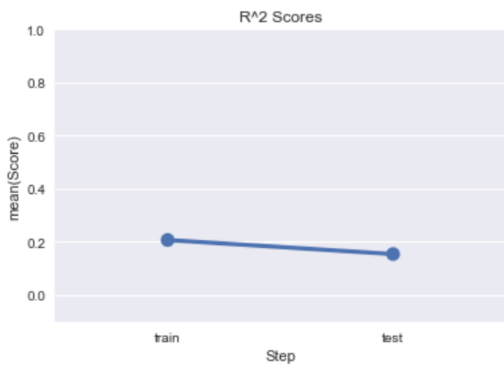
| | Score | Step |
|---|---|---|
| **0** | 0.207322 | train |
| **1** | 0.154473 | test |

```
[ ] lassoResults = pd.DataFrame()
    lassoResults['Score'] = [ltrainScore,ltestScore]
    lassoResults['Step'] = ['train','test']

    lassoResults
```

| | Score | Step |
|---|---|---|
| **0** | 0.122903 | train |
| **1** | 0.085031 | test |

```
[ ] sns.pointplot(y=lrResults['Score'],x=lrResults['Step'])
    plt.ylim([-.1,1])
    plt.title('R^2 Scores')
    plt.savefig('LRScores.png',bbox_inches='tight')
```




The linear regression model performed poorly overall.

- Slight overfitting: the R2 value fell .05 points from train to test
- Weak predictive power: .207 R2 train, .154 in test

Random Forest Approach

```
[ ] from sklearn.ensemble import RandomForestRegressor
```

```
[ ] x_train,x_test,y_train, y_test = train_test_split(x,
                                                      y, test_size=0.4,
                                                      random_state=42)
```

For the Random Forest we will use a min samples split of 10, as to avoid overfitting.

```
[ ] rf = RandomForestRegressor(n_estimators=500,min_samples_split=10)
    rf.fit(x_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=10, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

Model Validation:

```
from sklearn.metrics import r2_score
from scipy.stats import spearmanr, pearsonr
predicted_train = rf.predict(x_train)
predicted_test = rf.predict(x_test)
test_score = r2_score(y_test, predicted_test)
spearman = spearmanr(y_test, predicted_test)
pearson = pearsonr(y_test, predicted_test)

print('Test data R-2 score: {}').format(test_score)
print('Test data Spearman correlation: {}').format(spearman[0])
print('Test data Pearson correlation: {}').format(pearson[0])

train_score = r2_score(y_train, predicted_train)
spearmanTrain = spearmanr(y_train, predicted_train)
pearsonTrain = pearsonr(y_train, predicted_train)

print(' ')

print('Train data R-2 score: {}').format(train_score)
print('Train data Spearman correlation: {}').format(spearmanTrain[0])
print('Train data Pearson correlation: {}').format(pearsonTrain[0])
```

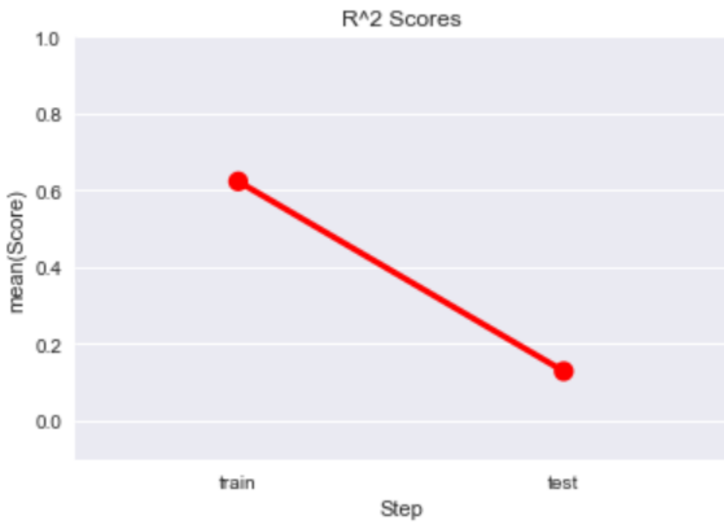
```
Test data R-2 score: 0.131753109439
Test data Spearman correlation: 0.398813691839
Test data Pearson correlation: 0.376914037366

Train data R-2 score: 0.61892366114
Train data Spearman correlation: 0.859328980587
Train data Pearson correlation: 0.833012895279
```

```
RFperf = pd.DataFrame()
RFperf['Score'] = [round(train_score,3),round(test_score,3)]
RFperf['Step'] = ['train','test']
RFperf
```

| | Score | Step |
|---|---|---|
| 0 | 0.619 | train |
| 1 | 0.132 | test |

```
[ ] sns.pointplot(y=RFperf['Score'],x=RFperf['Step'],color='Red')
    plt.ylim([-.1,1])
    plt.title('R^2 Scores')
    plt.savefig('RFScores.png',bbox_inches='tight')
```




Best results came from Random Forest Approach

- 500 estimators
- 10 min sample split

We had solid performance in the test set, with: - .623 $R^2$ value

Feature Importance:

```
[ ] predicted_test = rf.predict(x_test)

    fI = pd.DataFrame()
    fI['Variable'] = list(x_train.columns)
    fI['Importance'] = rf.feature_importances_
    fI.sort_values(by='Importance',ascending=False)[0:15]
```

[ ]

| | Variable | Importance |
|---|---|---|
| 0 | Page total likes | 0.188888 |
| 5 | Cat_1 | 0.123647 |
| 10 | We | 0.062571 |
| 40 | Mo_10 | 0.057278 |
| 1 | Paid | 0.048075 |
| 9 | Sa | 0.037287 |
| 2 | Video | 0.031045 |
| 23 | hr__10 | 0.029460 |
| 11 | Th | 0.027905 |
| 17 | hr__4 | 0.026808 |
| 34 | Mo_5 | 0.025562 |
| 31 | Mo_2 | 0.023368 |
| 36 | Mo_7 | 0.019990 |
| 15 | hr__2 | 0.018551 |
| 37 | Mo_8 | 0.017013 |

[ ]

```
topVars= list(fI.sort_values(by='Importance',ascending=False)[0:15]['Variable'])
topVars
```

```
['Page total likes',
 'Cat_1',
 'Paid',
 'Mo_10',
 'hr__13',
 'Sa',
 'Mo_7',
 'Th',
 'hr__10',
 'hr__4',
 'Tu',
 'Fr',
 'Video',
 'Cat_2',
 'hr__1']
```

```
[ ] x = df[topVars]
```

```
[ ] x_train,x_test,y_train, y_test = train_test_split(x,
                                                    y, test_size=0.3,
                                                    random_state=50)
```

```
[ ] rf = RandomForestRegressor(n_estimators=500,min_samples_split=15)
    rf.fit(x_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=15, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

```
[ ] from sklearn.metrics import r2_score
    from scipy.stats import spearmanr, pearsonr
    predicted_train = rf.predict(x_train)
    predicted_test = rf.predict(x_test)
    test_score = r2_score(y_test, predicted_test)
    spearman = spearmanr(y_test, predicted_test)
    pearson = pearsonr(y_test, predicted_test)

    #print(f'Out-of-bag R-2 score estimate: {rf.oob_score_:>5.3}')
    print('Test data R-2 score: {}').format(test_score)
    print('Test data Spearman correlation: {}').format(spearman[0])
    print('Test data Pearson correlation: {}').format(pearson[0])

    train_score = r2_score(y_train, predicted_train)
    spearmanTrain = spearmanr(y_train, predicted_train)
    pearsonTrain = pearsonr(y_train, predicted_train)

    print(' ')

    #print(f'Out-of-bag R-2 score estimate: {rf.oob_score_:>5.3}')
    print('Train data R-2 score: {}').format(train_score)
    print('Train data Spearman correlation: {}').format(spearmanTrain[0])
    print('Train data Pearson correlation: {}').format(pearsonTrain[0])
```

```
Test data R-2 score: 0.0735779002202
Test data Spearman correlation: 0.366331840555
Test data Pearson correlation: 0.30490941908

Train data R-2 score: 0.463470449251
Train data Spearman correlation: 0.709508194683
Train data Pearson correlation: 0.71481692673
```

The model performed substantially worse when taking the top 15 features by importance from the old model

**Modelling Conclusion:** After iterating through a random forest using the most important variables and seeing no improvement, this suggests that the data here is not rich enough to sufficiently predict likes based only on the information here.