Python for data analysis

MAQUET François-Xavier

MOSER Paul

NESSI Victor

I. Data pre-processing

- ## Check for NA

```
df[df.isna().any(axis=1)]
```

url	timedelta	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words

- **Normalisation of column name**

```
column_headers = list(df.columns.values)
print("The Column Header :", column_headers)
```

```
The Column Header : ['url', ' timedelta', ' n_tokens_title', ' n_tokens_content', ' n_unique_tokens', ' n_non_stop_words', ' n_
non_stop_unique_tokens', ' num_hrefs', ' num_self_hrefs', ' num_imgs', ' num_videos', ' average_token_length', ' num_keywords',
' data_channel_is_lifestyle', ' data_channel_is_entertainment', ' data_channel_is_bus', ' data_channel_is_socmed', ' data_chann
el_is_tech', ' data_channel_is_world', ' kw_min_min', ' kw_max_min', ' kw_avg_min', ' kw_min_max', ' kw_max_max', ' kw_avg_ma
x', ' kw_min_avg', ' kw_max_avg', ' kw_avg_avg', ' self_reference_min_shares', ' self_reference_max_shares', ' self_reference_a
vg_sharess', ' weekday_is_monday', ' weekday_is_tuesday', ' weekday_is_wednesday', ' weekday_is_thursday', ' weekday_is_frida
y', ' weekday_is_saturday', ' weekday_is_sunday', ' is_weekend', ' LDA_00', ' LDA_01', ' LDA_02', ' LDA_03', ' LDA_04', ' globa
l_subjectivity', ' global_sentiment_polarity', ' global_rate_positive_words', ' global_rate_negative_words', ' rate_positive_wo
rds', ' rate_negative_words', ' avg_positive_polarity', ' min_positive_polarity', ' max_positive_polarity', ' avg_negative_pola
rity', ' min_negative_polarity', ' max_negative_polarity', ' title_subjectivity', ' title_sentiment_polarity', ' abs_title_subj
ectivity', ' abs_title_sentiment_polarity', ' shares']
```

```
df.columns = df.columns.str.replace(' ','')
```

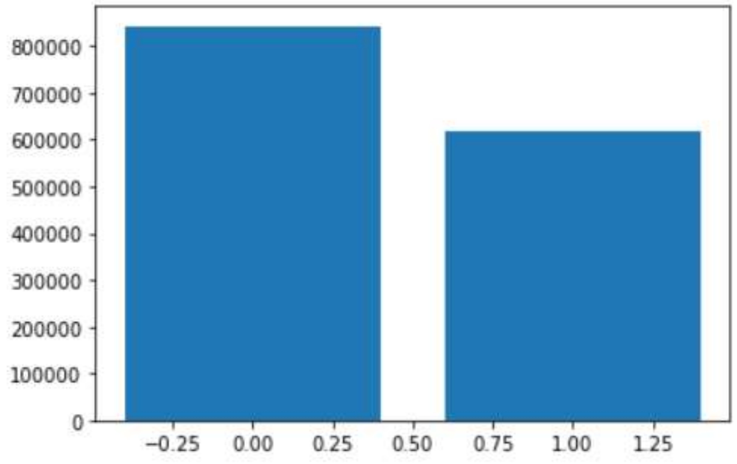
I. Data pre-processing

- **Imputation**

is_weekend

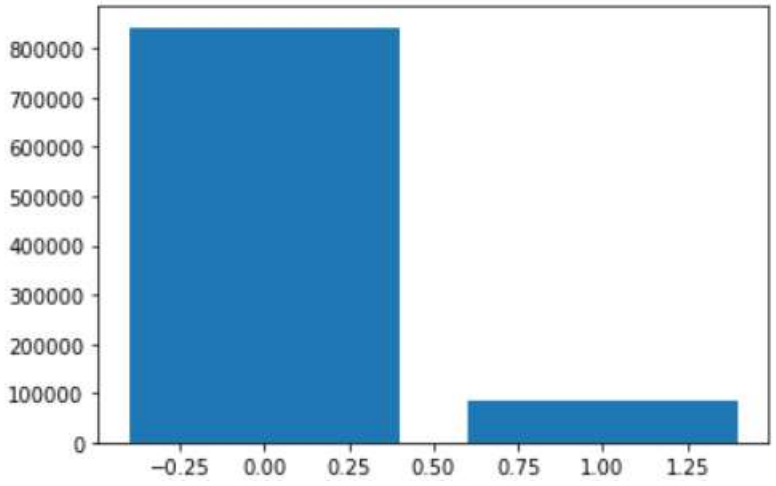
```
plt.bar(df['weekday_is_saturday'],df['shares'])
```

<BarContainer object of 39644 artists>




```
plt.bar(df['weekday_is_sunday'],df['shares'])
```

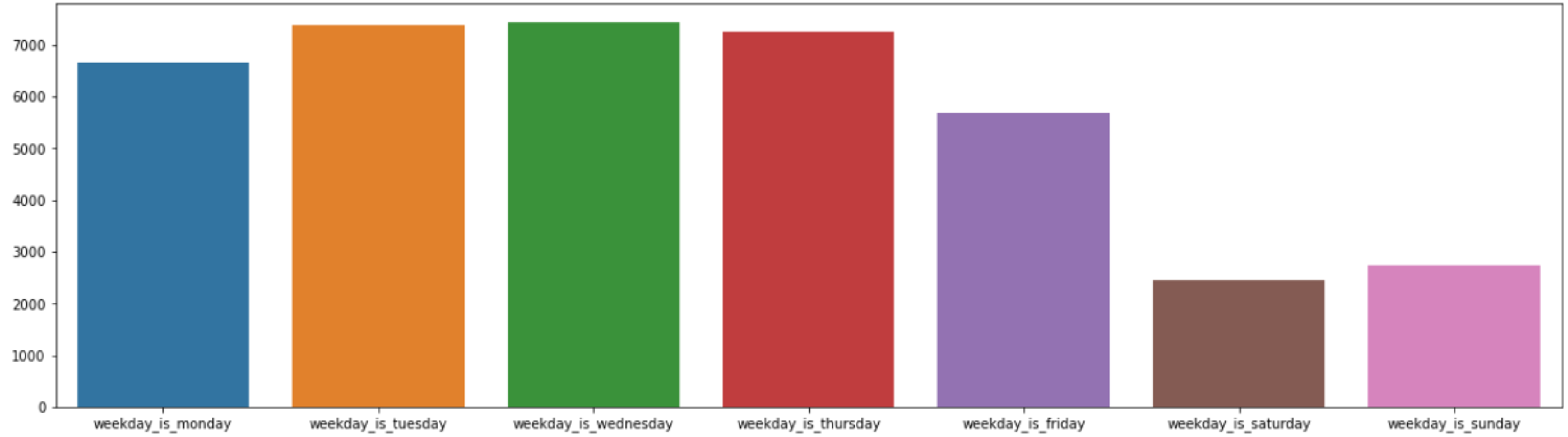
<BarContainer object of 39644 artists>



I. Data pre-processing

- **Optimization of days'columns**

```
dh = df[['weekday_is_monday','weekday_is_tuesday', 'weekday_is_wednesday', 'weekday_is_thursday', 'weekday_is_friday', 'week
x = dh.index
y = dh[['weekday_is_monday','weekday_is_tuesday', 'weekday_is_wednesday', 'weekday_is_thursday', 'weekday_is_friday', 'weekd
plt.rcParams["figure.figsize"] = (20,5.5)
sns.barplot(x,y)
plt.show()
```



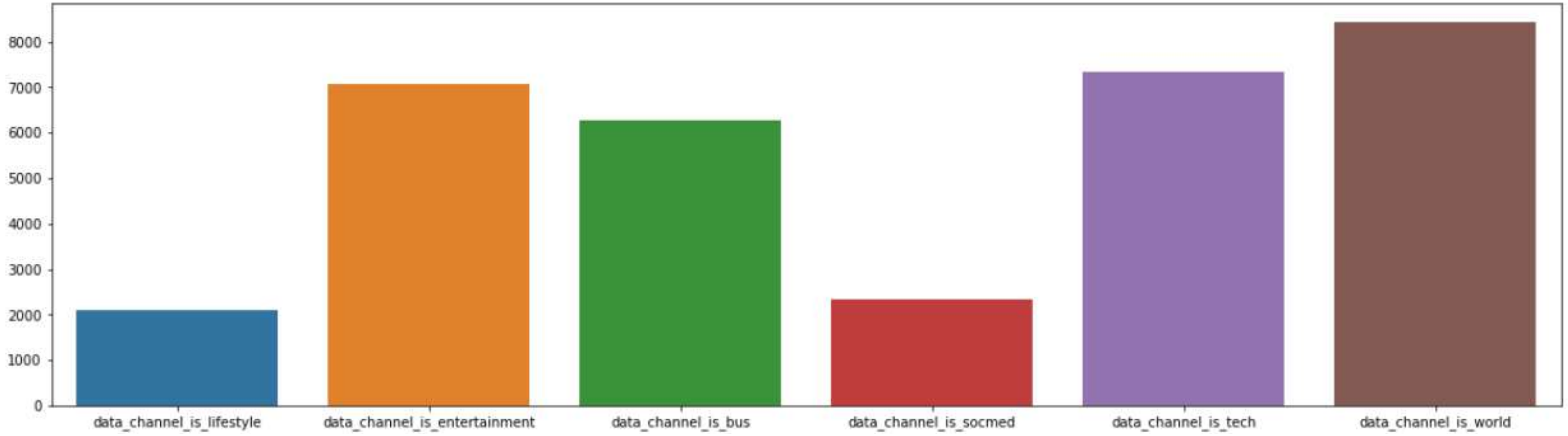
Let's optimize these days' column by creating a new column with the same amount of data

```
df['day'] = df['weekday_is_monday']+2*df['weekday_is_tuesday']+3*df['weekday_is_wednesday']+4*df['weekday_is_thursday']+5*df
df.drop(columns=["weekday_is_monday","weekday_is_tuesday","weekday_is_wednesday","weekday_is_thursday","weekday_is_friday","
```

I. Data pre-processing

- **Optimization of domain columns**

```
dh = df[['data_channel_is_lifestyle','data_channel_is_entertainment', 'data_channel_is_bus', 'data_channel_is_socmed', 'data
x = dh.index
y = dh[['data_channel_is_lifestyle','data_channel_is_entertainment', 'data_channel_is_bus', 'data_channel_is_socmed', 'data_
plt.rcParams["figure.figsize"] = (20,5.5)
sns.barplot(x,y)
plt.show()
```



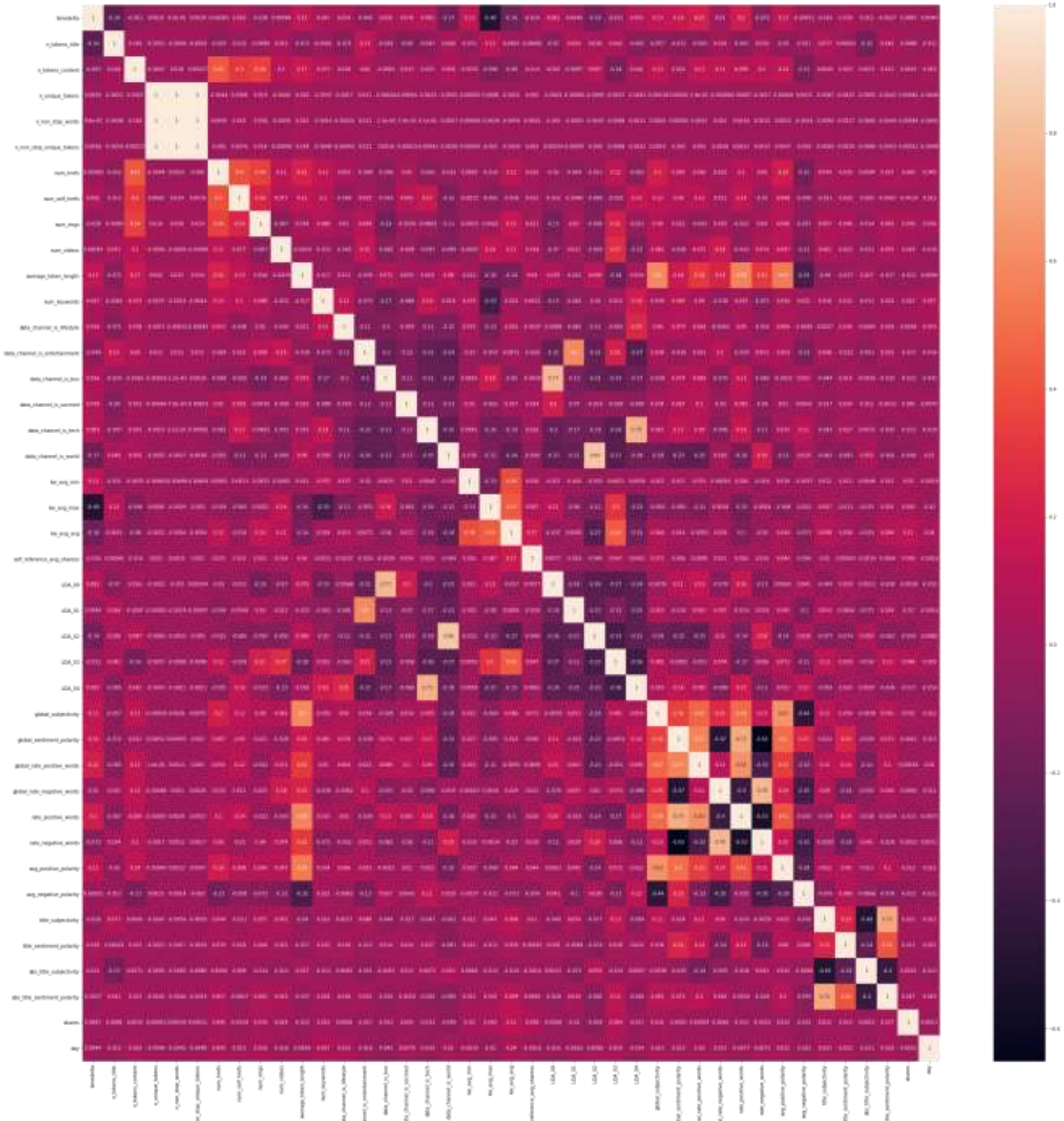
Let's optimize these days' column by creating a new column with the same amount of data

```
df['domain'] = df['data_channel_is_lifestyle']+2*df['data_channel_is_entertainment']+3*df['data_channel_is_bus']+4*df['data_
df
```

II. Visualization

Plot the correlations

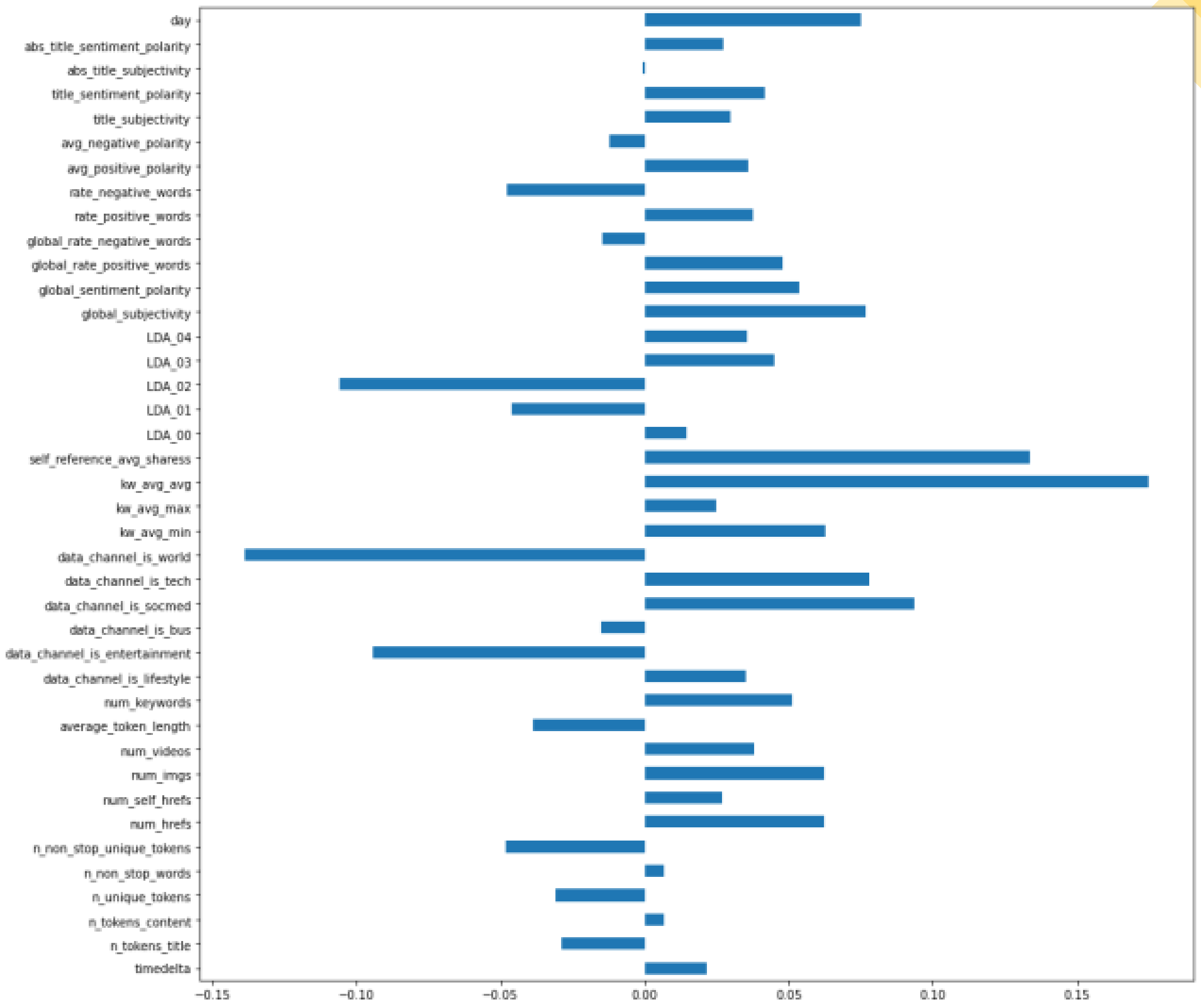
```
plt.figure(figsize=(40,40))
sns.heatmap(data=df.corr(),annot=True)
```



II. Visualization

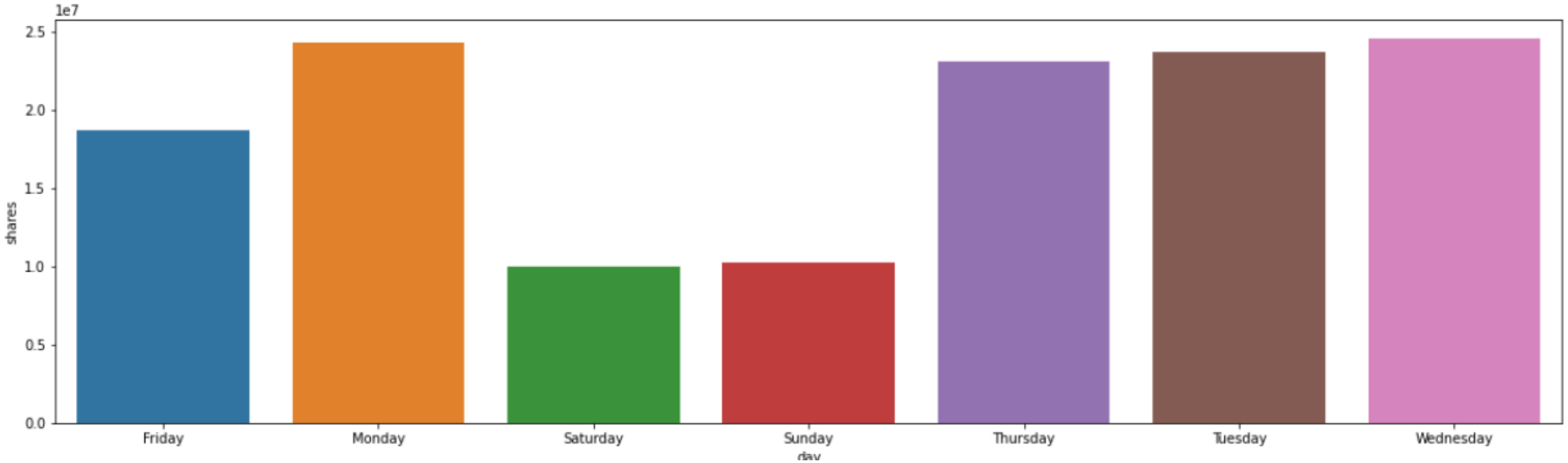
```
dcorr = df.corr(method='kendall')
```

```
dcorr.drop(columns='shares', inplace = True)
line = dcorr.loc['shares']
line.plot.barh(figsize=(15,15))
```

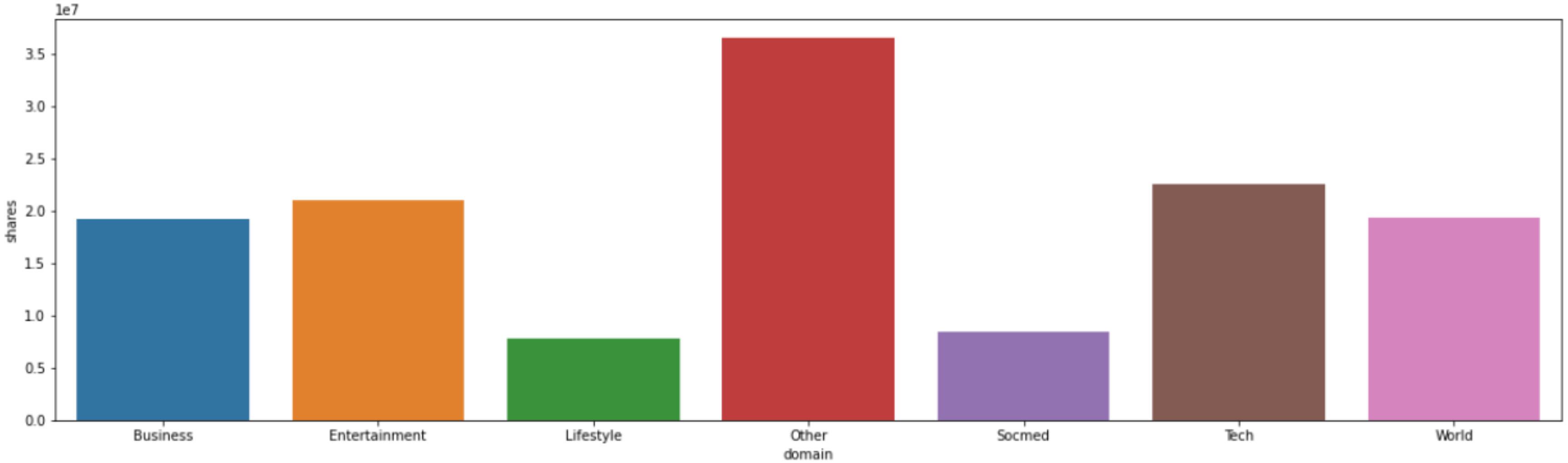


II. Visualization

- **Amount of shares by day**

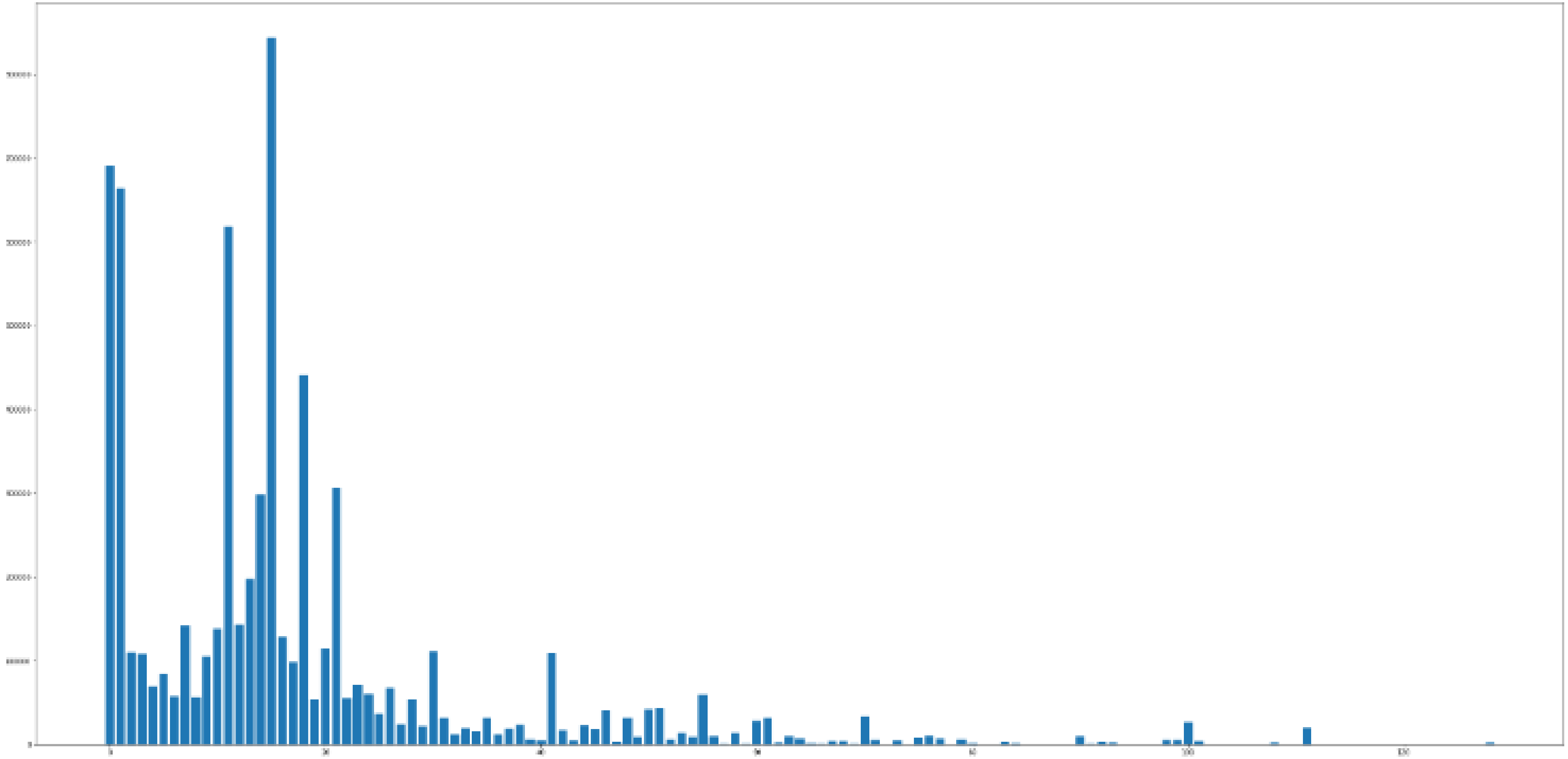


- **Amount of shares by domain**



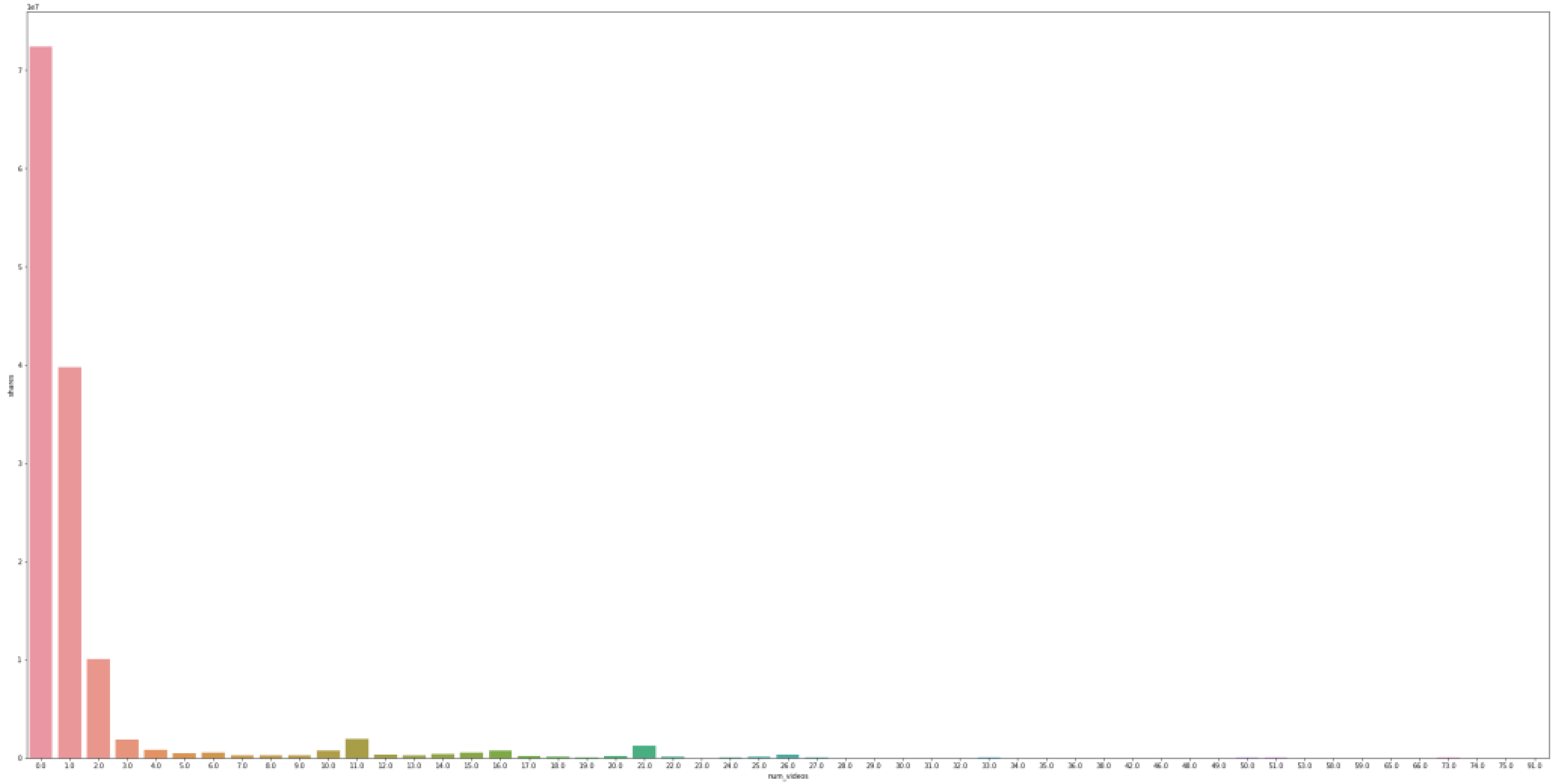
II. Visualization

- **Number of shares for the number of images**



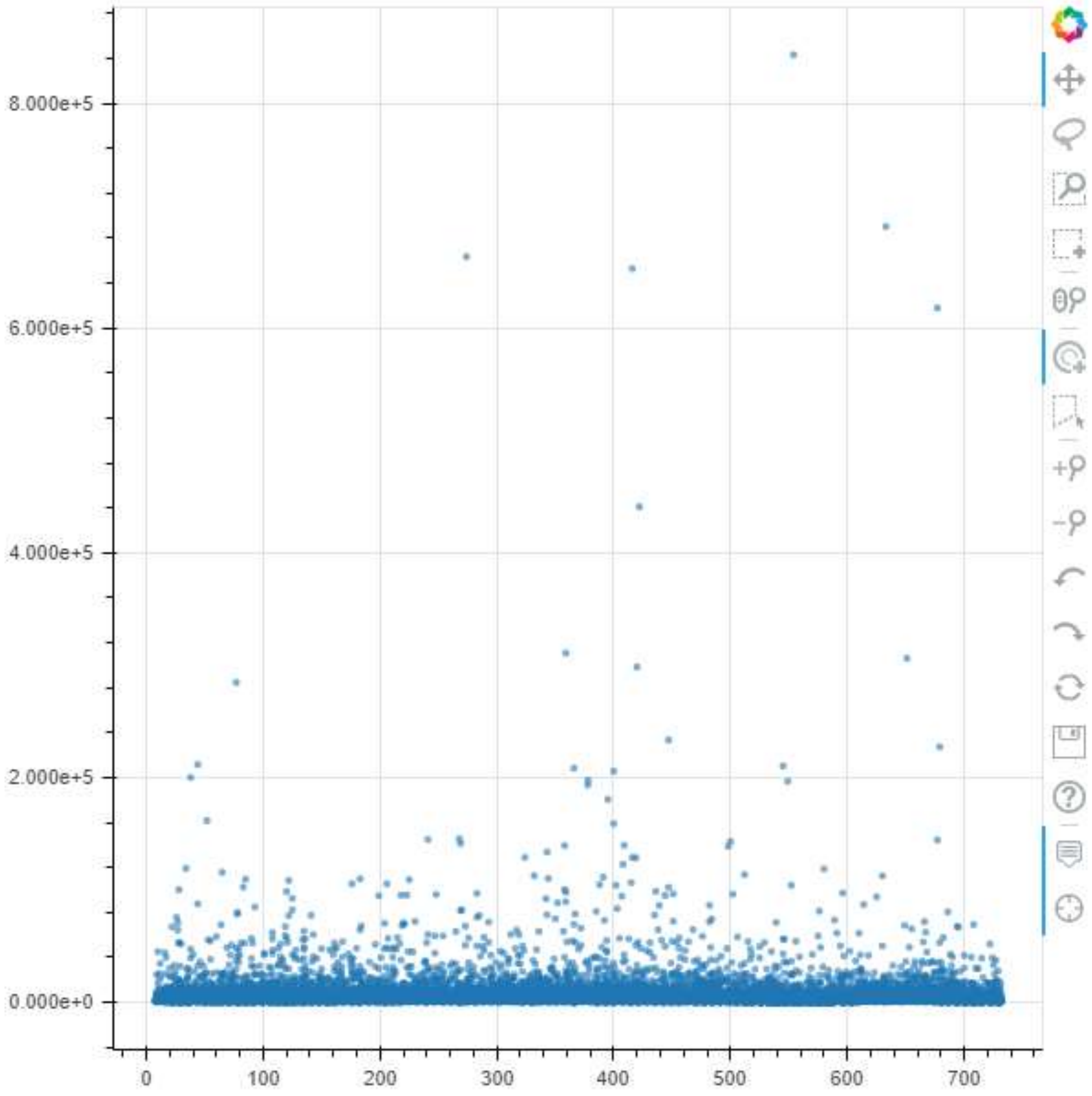
II. Visualization

- **Number of shares for the number of videos**



II. Visualization

- shares for the time delta using bokeh

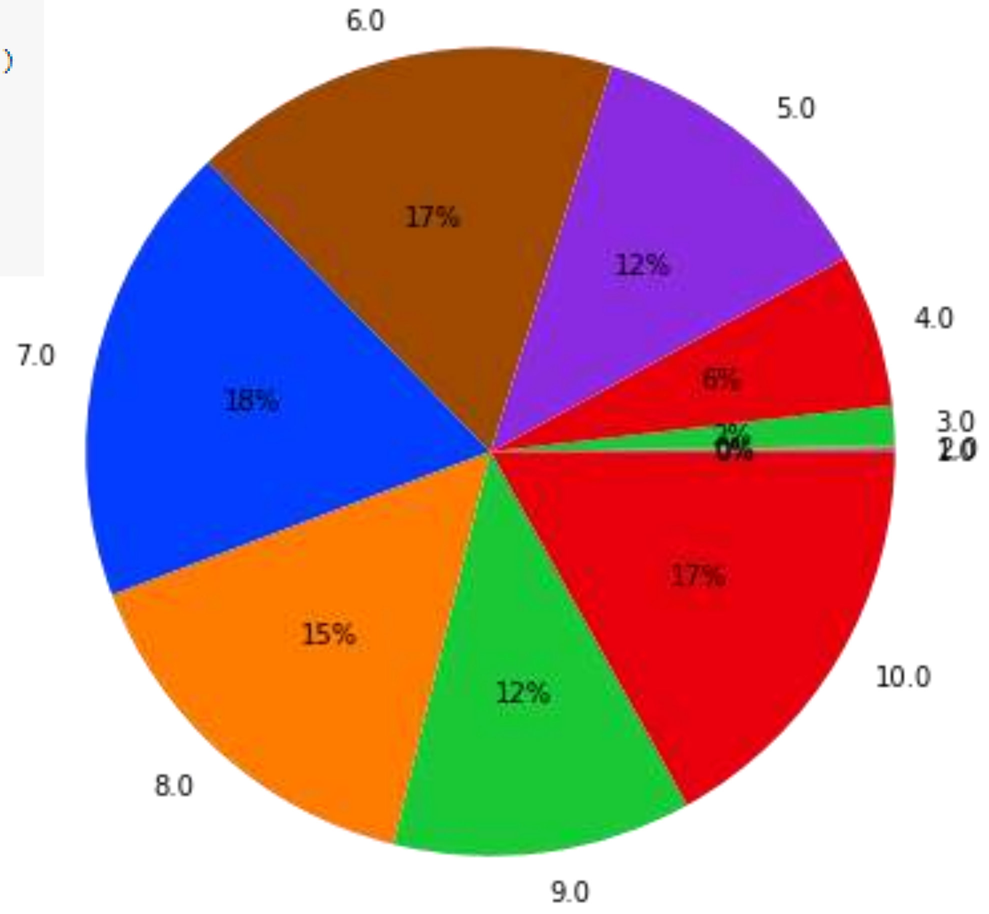


II. Visualization

- **Distribution of articles according to their number of keywords**

```
import matplotlib.pyplot as plt

tabgene = df[["num_keywords","url"]].groupby("num_keywords").count()
data = tabgene["url"]
labels = tabgene.index
colors = sns.color_palette('bright')[0:6]
plt.pie(data, labels = labels, colors = colors, autopct='%.0f%%')
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```



III.Modeling

Regression

- **Splitting and fitting the dataset**

```
X = df.drop(['shares','url'],axis=1)
y = df['shares']
sc=StandardScaler()
X=sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

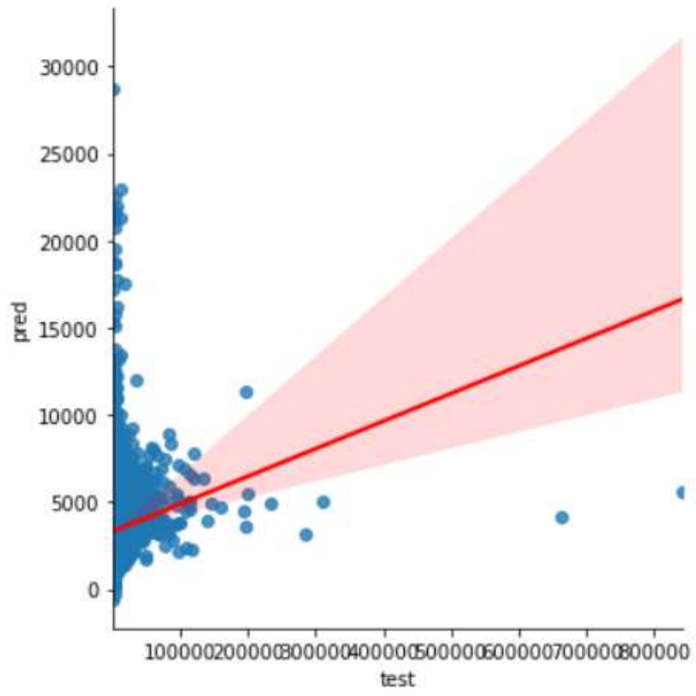
III.Modeling

- **Linear Regression**

```
lm = linear_model.LinearRegression()
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
print(mean_squared_error(y_test, y_pred))
eval_logic = pd.DataFrame()
eval_logic['test'] = y_test
eval_logic['pred'] = y_pred

sns.lmplot(x = 'test', y = 'pred', data = eval_logic, line_kws={'color':'red'})
plt.show()
```

```
168711874.01750997
```



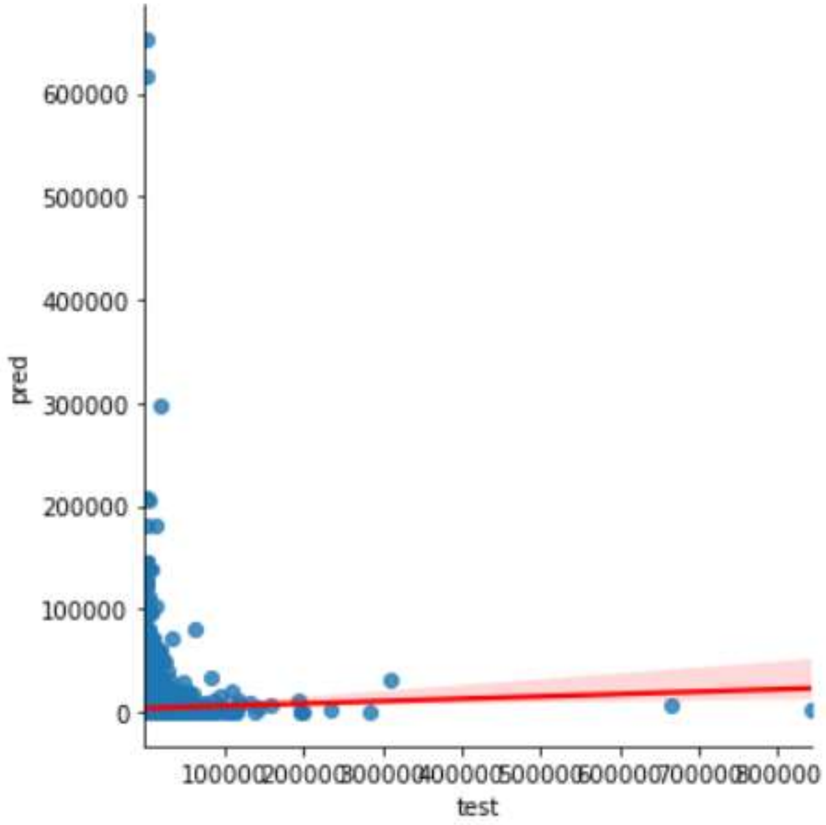
III.Modeling

- **Decision Tree**

```
dt = tree.DecisionTreeRegressor()
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
print(mean_squared_error(y_test, y_pred))
eval_logic = pd.DataFrame()
eval_logic['test'] = y_test
eval_logic['pred'] = y_pred

sns.lmplot(x = 'test', y = 'pred', data = eval_logic, line_kws={'color':'red'})
plt.show()
```

316264834.3105557



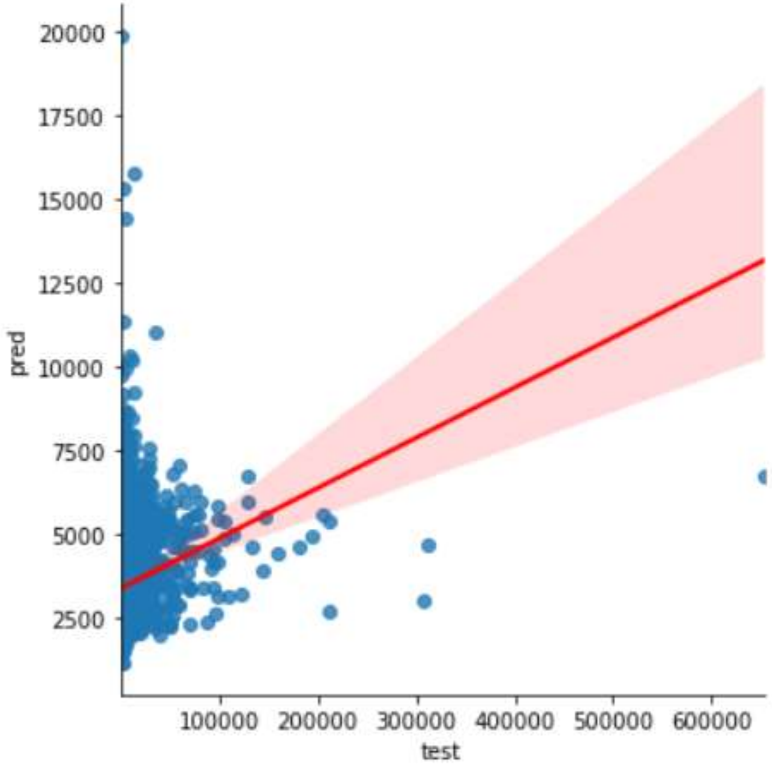
III.Modeling

- **Logic Regression**

```
from sklearn.linear_model import TweedieRegressor
lr = TweedieRegressor()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print(mean_squared_error(y_test, y_pred))
eval_logic = pd.DataFrame()
eval_logic['test'] = y_test
eval_logic['pred'] = y_pred

sns.lmplot(x = 'test', y = 'pred', data = eval_logic, line_kws={'color':'red'})
plt.show()
```

```
110492485.29553321
```



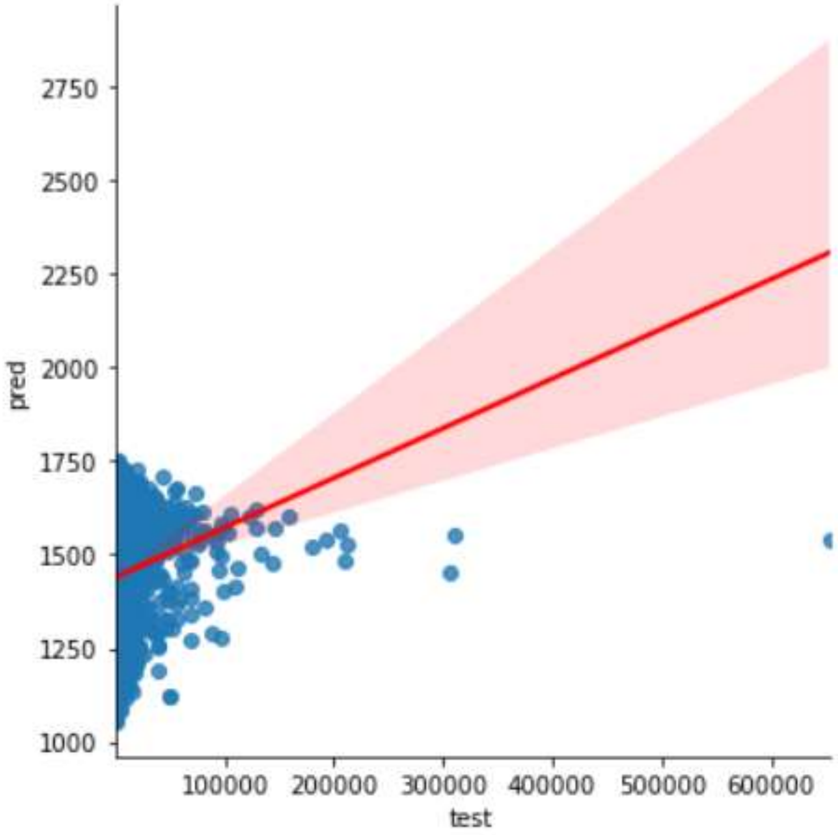
III.Modeling

- **SVM**

```
from sklearn import svm
regr = svm.SVR()
regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
print(mean_squared_error(y_test, y_pred))
eval_logic = pd.DataFrame()
eval_logic['test'] = y_test
eval_logic['pred'] = y_pred
eval_logic.sort_values(by = 'test',inplace = True)

sns.lmplot(x = 'test', y = 'pred', data = eval_logic, line_kws={'color':'red'})
plt.show()
```

116231013.14354613

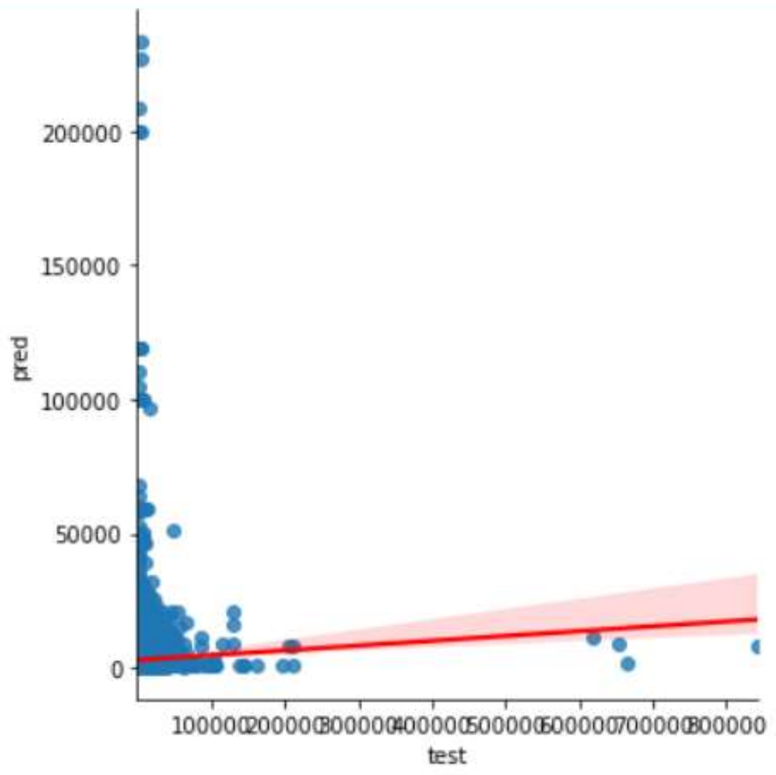


III.Modeling

- **Neural Network**

```python
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
np.mean(y_test== y_pred)
eval_logic = pd.DataFrame()
eval_logic['test'] = y_test
eval_logic['pred'] = y_pred
eval_logic.sort_values(by = 'test',inplace = True)

sns.lmplot(x = 'test', y = 'pred', data = eval_logic, line_kws={'color':'red'})
plt.show()
```



III.Modeling

Classification

- **Spliting and fiting the dataset**

```
x = df.drop(['shares','url'],axis=1)
y = df['shares'].apply(lambda x: 0 if x <df['shares'].mean()  else 1)
```

```
sc=StandardScaler()
x=sc.fit_transform(x)
x,y = SMOTE().fit_resample(x, y)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=0)
```

III.Modeling

- **Logistic Regression**

```
LR = LogisticRegression(multi_class='auto')
LR.fit(x_train,y_train)
pred = LR.predict(x_test)
print("Accu", accuracy_score(y_test,pred))
print("MSE", metrics.mean_squared_error(y_test, pred))
```

```
Accu 0.6396102337637403
MSE 0.3603897662362598
```

III.Modeling

- **Decision Tree**

```python
dtree = tree.DecisionTreeRegressor()
dtree.fit(x_train, y_train)
pred = dtree.predict(x_test)
print("Accu", accuracy_score(y_test,pred))
print("MSE", metrics.mean_squared_error(y_test, pred))
```

```
Accu 0.7727163634618154
MSE 0.2272836365381846
```

III.Modeling

- **Random Forest**

```
randf = RandomForestClassifier(n_estimators=100, max_features="sqrt", random_state=40)
randf.fit(x_train, y_train)
pred = randf.predict(x_test)
print("Accu", accuracy_score(y_test,pred))
print("MSE", metrics.mean_squared_error(y_test, pred))
```

```
Accu 0.8773100369605914
MSE 0.12268996303940863
```

III.Modeling

- **SVM**

```
clf = svm.SVC()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
mean_squared_error(y_test, y_pred)
```

```
0.2874285988575817
```

III.Modeling

Model name	Logic Regression	Decision Tree	Random Forest	
Accuracy	0.6396102337637403	0.772716363461815	0.8818221091537465	
MSE	0.3603897662362598	0.2272836365381846 2	0.1181778908462535 3	