Master Modeler Competition

(Laird Stewart/Joey Hodson)

Importing modules

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

import shap
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model
from sklearn import metrics
import nltk # python natural language toolkit
nltk.download('vader_lexicon') # Vader lexicon for sentiment analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer # vader sentiment analyzer
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/josephhodson/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [2]:
```
# Settings
pd.set_option('display.max_columns', None)
# %matplotlib notebook
```

Importing Data

In [3]:
```
df_original = pd.read_csv('master_model.csv')
```

Data Exploration

In [4]:
```
df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2400 entries, 0 to 2399
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   company           2400 non-null   object
 1   date              2400 non-null   object
 2   type              2400 non-null   object
 3   URL               2392 non-null   object
 4   text              2136 non-null   object
 5   total engagement  2400 non-null   int64
 6   engagement rate   2400 non-null   float64
 7   reactions         2400 non-null   int64
 8   shares            2400 non-null   int64
 9   comments          2400 non-null   int64
dtypes: float64(1), int64(4), object(5)
memory usage: 187.6+ KB
```

In [5]:
```
df_original.describe()
```

Out[5]:

	total engagement	engagement rate	reactions	shares	comments
count	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000
mean	28.837083	106.261380	13.742500	14.171667	0.922917
std	100.173733	265.710197	19.916801	88.641005	2.940761
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	6.000000	27.234558	4.000000	1.000000	0.000000
50%	12.000000	54.953937	8.000000	4.000000	0.000000
75%	27.000000	107.979334	15.000000	10.000000	1.000000
max	4057.000000	8595.338983	292.000000	3745.000000	67.000000

In [6]:
```
# Media Types

# Pie chart plot
types = list(df_original['type'])
labels = ['photo', 'link', 'video', 'status', 'event']
l = len(types)
photo = types.count('photo')/l
link = types.count('link')/l
video = types.count('video')/l
status = types.count('status')/l
event = types.count('event')/l
sizes = [photo, link, video, status, event]
print(labels)
print(sizes) # data used to create slideshow figures
# fig1, ax1 = plt.subplots()
# ax1.pie(sizes, labels=labels)
# ax1.axis('equal')
# plt.show()
```

```
['photo', 'link', 'video', 'status', 'event']
[0.2608333333333333, 0.6041666666666666, 0.11791666666666667, 0.012916666666666667, 0.0041666
66666666667]
```

Data Cleaning

In [7]:
```
df0 = df_original.copy()

# create datetime objects (for easier analysis)
df0['date'] = pd.to_datetime(df0['date'])

# sort by date
df0 = df0.sort_values(by='date')

# drop 'company' tag
df1 = df0.drop(['company', 'URL'],axis=1)

# create binary dummy variables for "type"
dummy = pd.get_dummies(df1['type'])
df2 = df1.merge(dummy, left_index=True, right_index=True)
df3 = df2.drop(['type'],axis=1)

# replace NaN values with "" in text (need to do this before counting hashtags)
df3['text'] = df3['text'].fillna("")
```

Extracting Text Features 1 (hashtags)

In [8]:
```
# number of hashtags (have to do this before removing hashtags in next cell)
df3['hashtags'] = df3['text'].map(lambda x: len(re.findall('\#',x)))
```

Text Cleaning

In [9]:
```
# replace \n with " " in 'text'
df3['text'] = df3['text'].map(lambda x: re.sub('\n',' ',x))

# remove non standard text charachters in 'text'
df3['text'] = df3['text'].map(lambda x: re.findall(r"[A-Za-z!.,']\s*",x)) # turns text into
 a list of hits
df3['text'] = df3['text'].map(lambda x: "".join(x)) # turn list of hits back into string

# turn multiple spaces into single spaces
df3['text'] = df3['text'].map(lambda x: re.sub('\s+',' ', x))

# replace any aA with a A
df3['text'] = df3['text'].map(lambda x: re.sub(r"([a-z])([A-Z])", r"\1 \2", x))
```

Extracting Text Features 2 (lenght, sentiment)

In [10]:
```
# length of post
df3['length'] = df3['text'].map(lambda x: len(x))

# sentiment score
sid = SentimentIntensityAnalyzer()
df3['sentiment'] = df3['text'].map(lambda x: sid.polarity_scores(x).get('compound'))
```

Extracting Date/Time Features

In [11]:
```
# Season
df3['spring'] = df3['date'].map(lambda x: 1 if (x.month in (3,4,5) ) else 0)
df3['fall'] = df3['date'].map(lambda x: 1 if (x.month in (9,10,11) ) else 0)
df3['summer'] = df3['date'].map(lambda x: 1 if (x.month in (6,7,8) ) else 0)
df3['winter'] = df3['date'].map(lambda x: 1 if (x.month in (12,1,2) ) else 0)

# Time of day (morning, afternoon, night)
df3['morning'] = df3['date'].map(lambda x: 1 if (x.hour in range (0, 13) ) else 0)
df3['afternoon'] = df3['date'].map(lambda x: 1 if (x.hour in range (13, 18) ) else 0)
df3['night'] = df3['date'].map(lambda x: 1 if (x.hour in range (18, 24) ) else 0)

# Day of week
df3['monday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 0 ) else 0)
df3['tuesday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 1 ) else 0)
df3['wednesday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 2 ) else 0)
df3['thursday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 3 ) else 0)
df3['friday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 4 ) else 0)
df3['saturday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 5 ) else 0)
df3['sunday'] = df3['date'].map(lambda x: 1 if (x.weekday() == 6 ) else 0)
```

Calculating group size (page follows/likes)

In [12]:
```
# engagement rate  = ((total engagement) / (group size) )* 100
# --> group size = total engagement * 10,000 / engagement rate
df4 = df3.copy()
df4['engagement rate'] = df3['engagement rate'] / 100 # proper percentage out of 100 (mentio
ned this in video)
df4['group size'] = (df4['total engagement'] / df4['engagement rate']) * 100

# note, some values of engagement rate are 0, this gives NaN for group size. So replace with
next valid entry
df4['group size'] = df4['group size'].fillna(method='ffill')
```

Scoring based on metrics

In [13]:
```
# calculate a score where share is 50x as valuable as a reaction and comment is 10x
df4['score'] = (df4['reactions'] + 10*df4['comments'] + 50*df4['shares']) / df4['group size'
]
```

Feature Scaling

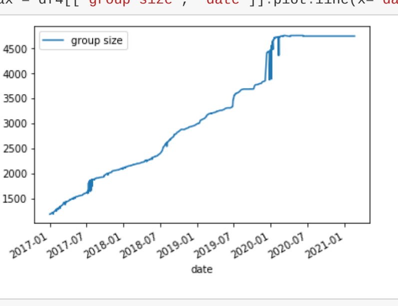
In [14]:
```
# log scale and normalize score
df4['log score'] = np.log(df4['score'] + 1) # +1 to avoid issues with log(0)
df4['norm log score'] = (df4['log score'] - min(df4['log score']))/(max(df4['log score']) -
min(df4['log score']))

# normalize length [0,1]
df4['norm length'] = df4['length'] / max(df4['length'])

# normalize sentiment [0,1]
df4['norm sentiment'] = (df4['sentiment'] - min(df4['sentiment']))/(max(df4['sentiment']) -
min(df4['sentiment']))

# normalize hastags [0,1]
df4['norm hashtags'] = df4['hashtags'] / max(df4['hashtags'])
```

Data Exploration (of cleaned data)

In [15]:
```
# Final Data Table
df4.info()
```

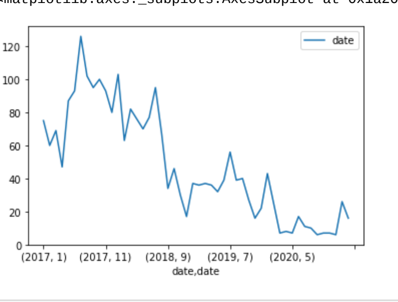
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2400 entries, 2205 to 1644
Data columns (total 36 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   date              2400 non-null   datetime64[ns]
 1   text              2400 non-null   object
 2   total engagement  2400 non-null   int64
 3   engagement rate   2400 non-null   float64
 4   reactions         2400 non-null   int64
 5   shares            2400 non-null   int64
 6   comments          2400 non-null   int64
 7   event             2400 non-null   uint8
 8   link              2400 non-null   uint8
 9   photo             2400 non-null   uint8
 10  status            2400 non-null   uint8
 11  video             2400 non-null   uint8
 12  hashtags          2400 non-null   int64
 13  length            2400 non-null   int64
 14  sentiment         2400 non-null   float64
 15  spring            2400 non-null   int64
 16  fall              2400 non-null   int64
 17  summer            2400 non-null   int64
 18  winter            2400 non-null   int64
 19  morning           2400 non-null   int64
 20  afternoon         2400 non-null   int64
 21  night             2400 non-null   int64
 22  monday            2400 non-null   int64
 23  tuesday           2400 non-null   int64
 24  wednesday         2400 non-null   int64
 25  thursday          2400 non-null   int64
 26  friday            2400 non-null   int64
 27  saturday          2400 non-null   int64
 28  sunday            2400 non-null   int64
 29  group size        2400 non-null   float64
 30  score             2400 non-null   float64
 31  log score         2400 non-null   float64
 32  norm log score    2400 non-null   float64
 33  norm length       2400 non-null   float64
 34  norm sentiment    2400 non-null   float64
 35  norm hashtags     2400 non-null   float64
dtypes: datetime64[ns](1), float64(9), int64(20), object(1), uint8(5)
memory usage: 611.7+ KB
```

In [16]:
```
# Group Size over time
ax = df4[['group size', 'date']].plot.line(x='date')
```



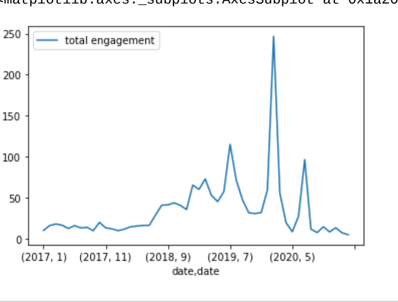
In [17]:
```
# Monthly Post Frequency
month = df4[['date']].groupby([df4["date"].dt.year, df4["date"].dt.month]).count()
month.plot(kind="line")
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20d6e450>



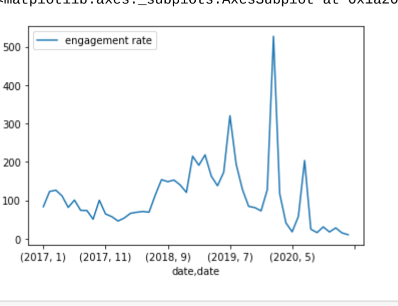
In [18]:
```
# Average total engagement (by month)
g = df4.groupby([df4["date"].dt.year, df4["date"].dt.month])
monthly_averages = g.aggregate({"total engagement":np.mean})

monthly_averages.plot(kind="line")
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20e21350>



In [19]:
```
# Average engagement RATE (by month)
g2 = df3.groupby([df3["date"].dt.year, df3["date"].dt.month])
monthly_averages2 = g2.aggregate({"engagement rate":np.mean})

monthly_averages2.plot(kind="line")
```

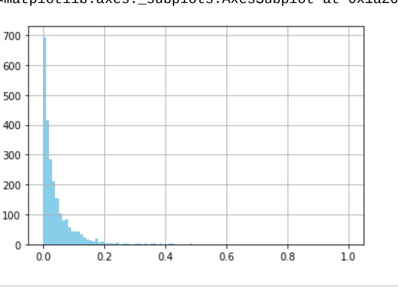
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20de4910>



In [20]:
```
# Normal Log scaled score
print(df4['norm log score'].describe(percentiles=[0.33, 0.5, 0.66]))
print("median of norm log score:\n\t" + str(np.median(df4['norm log score'])))
df4['norm log score'].hist(bins=100, color='skyblue')
```

```
count    2400.000000
mean        0.044679
std         0.070369
min         0.000000
33%         0.012823
50%         0.023578
66%         0.038691
max         1.000000
Name: norm log score, dtype: float64
median of norm log score:
        0.023578034608563336
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20b94050>



In [21]:
```
# Total posts by day of week
print("monday:\t" + str(sum(df4['monday'])))
print("tuesday:\t" + str(sum(df4['tuesday'])))
print("wednesday:\t" + str(sum(df4['wednesday'])))
print("thursday:\t" + str(sum(df4['thursday'])))
print("friday:\t" + str(sum(df4['friday'])))
print("saturday:\t" + str(sum(df4['saturday'])))
print("sunday:\t" + str(sum(df4['sunday'])))

# Total posts by time of day
print("\nmorning:\t" + str(sum(df4['morning'])))
print("afternoon:\t" + str(sum(df4['afternoon'])))
print("night:\t" + str(sum(df4['night'])))
```

```
monday: 321
tuesday:        384
wednesday:      424
thursday:       352
friday: 326
saturday:       304
sunday: 289

morning:        1114
afternoon:      744
night:  542
```

In [22]:
```
# Summary statistics of important variables
df4[['total engagement', 'engagement rate','reactions', 'shares', 'comments', 'length', 'has
htags', 'sentiment', 'score']].describe()
```

Out[22]:

	total engagement	engagement rate	reactions	shares	comments	length	hashtags	sentiment	score
count	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000	2400.000000
mean	28.837083	1.062614	13.742500	14.171667	0.922917	192.175417	0.848750	0.097684	0.258786
std	100.173733	2.657102	19.916801	88.641005	2.940761	182.776642	1.801544	0.541331	1.169741
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.993600	0.000000
25%	6.000000	0.272346	4.000000	1.000000	0.000000	78.750000	0.000000	-0.296000	0.027919
50%	12.000000	0.549539	8.000000	4.000000	0.000000	169.000000	0.000000	0.000000	0.091365
75%	27.000000	1.079793	15.000000	10.000000	1.000000	256.000000	1.000000	0.598525	0.221454
max	4057.000000	85.953390	292.000000	3745.000000	67.000000	2148.000000	32.000000	0.992800	39.775847

In [23]:
```
df4[['norm length', 'norm hashtags', 'norm sentiment', 'norm log score']].describe()
```

Out[23]:

	norm length	norm hashtags	norm sentiment	norm log score
count	2400.000000	2400.000000	2400.000000	2400.000000
mean	0.089467	0.026523	0.549378	0.044679
std	0.085092	0.056298	0.272519	0.070369
min	0.000000	0.000000	0.000000	0.000000
25%	0.036662	0.000000	0.351188	0.007426
50%	0.078678	0.000000	0.500201	0.023578
75%	0.119181	0.031250	0.801513	0.053947
max	1.000000	1.000000	1.000000	1.000000

Model Building

In [24]:
```
# Feature (X) and Output (Y) data
# Note X does not include winter, night, sunday or status because the information is already
contained
X = df4[['spring', 'fall', 'summer', 'morning', 'afternoon', 'monday', 'tuesday', 'wednesda
y', 'thursday', 'friday', 'saturday', 'norm length', 'norm sentiment', 'norm hashtags', "eve
nt", "link", "photo", "video"]]
Y = df4[['norm log score']]
Y0 = Y.values.ravel()
```

Multiple OLS Linear Regression

In [25]:
```
# Linear regressoin fit
ols_reg = linear_model.LinearRegression()
ols_reg.fit(X, Y)

# coeficients and intercept
print('Coefficients: \n', ols_reg.coef_)
```

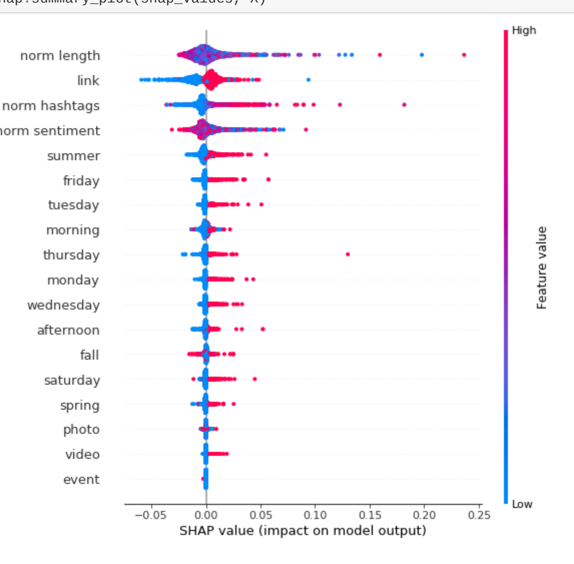
```
Coefficients:
 [[-0.00108957 -0.00689529  0.00839194  0.00463118 -0.00046971  0.00955364
   0.00774542  0.00687699  0.00494183  0.01112297  0.00211908  0.01446628
  -0.00543627  0.10347633 -0.02088688  0.02459389  0.01007298  0.00851037]]
```

Random Forest Regression

In [26]:
```
# define the model
rfr = RandomForestRegressor()
# fit the model, Hyperparamater turning was tested with no significant changes in feature im
portance found
rfr.fit(X, Y0)
```

Out[26]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [27]:
```
#SHAP used to visualize feature importance
explainer = shap.TreeExplainer(rfr)
shap_values = explainer.shap_values(X)
shap.summary_plot(shap_values, X)
```



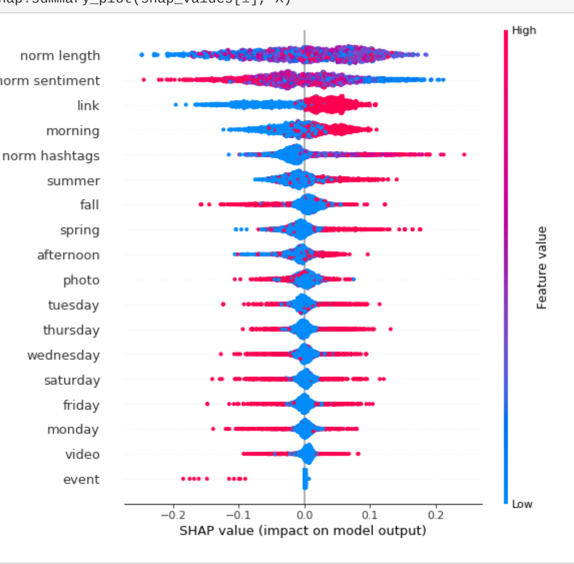
Random Forest Classifier

In [28]:
```
#defining classification criteria of "good" as a log normalized score greater than the 50th
 percentile and "bad" as the opposite
df4['class'] = df4['norm log score'].map(lambda x: 1 if (x >= .02358) else 0 )
df_labels = df4[['class']]
df_array = df_labels.loc[:,'class']
numbers = df_array.values
```

In [29]:
```
# define the model
rfc = RandomForestClassifier()
# fit the model
rfc.fit(X, numbers)
```

Out[29]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [30]:
```
#SHAP used to visualize feature importance
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X)
shap.summary_plot(shap_values[1], X)
```

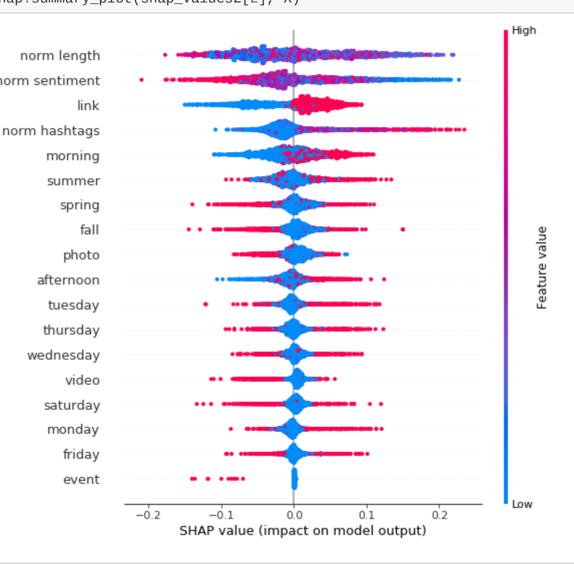


In [31]:
```
#defining classification criteria of "good" as a log normalized score greater than the 66th
 percentile
#bad as  a log normalized score less than the 33th percentile, and "medium" as anything inbe
tween
df4['class2'] = df4['norm log score'].map(lambda x: 0 if (x < .012823) else 1 if (x >= .0128
23 and x <.038691) else 2 )
df_labels2 = df4[['class2']]
df_array2 = df_labels2.loc[:,'class2']
numbers2 = df_array2.values
```

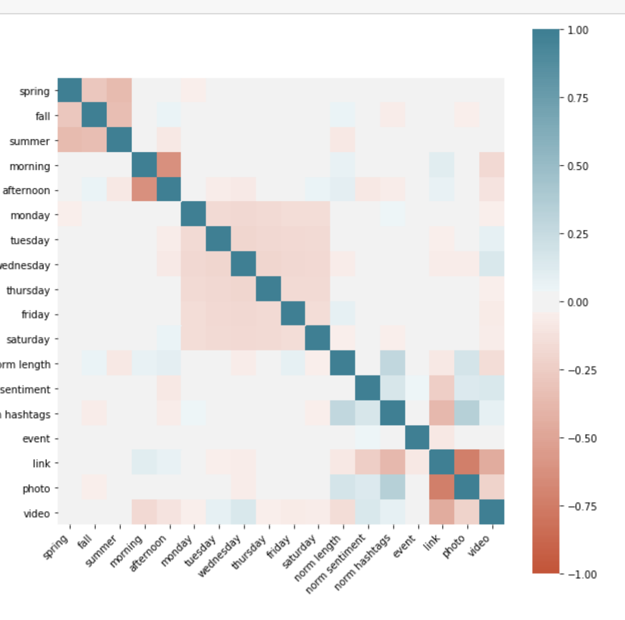
In [32]:
```
# define the model
rfc2 = RandomForestClassifier()
# fit the model
rfc2.fit(X, numbers2)
```

Out[32]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [33]:
```
#SHAP used to visualize feature importance
explainer2 = shap.TreeExplainer(rfc2)
shap_values2 = explainer2.shap_values(X)
shap.summary_plot(shap_values2[2], X)
```



In [35]:
```
#Creating a correlation matrix of features to test for multi-collinearity
fig1, ax1 = plt.subplots(figsize=(10,10))

ax1 = sns.heatmap(X.corr(),
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True

)
ax1.set_xticklabels(
    ax1.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



```{r}
data <- read.csv("OLSregression.csv") # read in data

lm <- lm(norm.log.score ~ spring + summer + fall + winter + sunday + morning + afternoon +
night + monday + tuesday + wednesday + thursday + friday + saturday + sunday + norm.length +
norm.sentiment + norm.hashtags + event + link + photo + status + video, data = data)
summary(lm)
```

```
Call:
lm(formula = norm.log.score ~ spring + summer + fall + winter +
    sunday + morning + afternoon + night + monday + tuesday +
    wednesday + thursday + friday + saturday + sunday + norm.length +
    norm.sentiment + norm.hashtags + event + link + photo + status +
    video, data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.09168 -0.03416 -0.01927  0.01032  0.96246

Coefficients: (4 not defined because of singularities)
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     0.0267729  0.0075510   3.546 0.000399 ***
spring         -0.0010880  0.0041033  -0.265 0.790919
summer          0.0083927  0.0038705   2.168 0.030228 *
fall           -0.0068928  0.0042195  -1.634 0.102482
winter                 NA         NA      NA       NA
sunday         -0.0021187  0.0057359  -0.369 0.711878
morning         0.0046292  0.0039665   1.167 0.243305
afternoon      -0.0004717  0.0042852  -0.110 0.912358
night                  NA         NA      NA       NA
monday          0.0074337  0.0055984   1.328 0.184358
tuesday         0.0056253  0.0053971   1.042 0.297381
wednesday       0.0047575  0.0053075   0.896 0.370151
thursday        0.0028226  0.0054666   0.516 0.605676
friday          0.0090065  0.0055825   1.613 0.106804
saturday               NA         NA      NA       NA
norm.length     0.0144721  0.0182279   0.794 0.427300
norm.sentiment -0.0054779  0.0054602  -1.003 0.315847
norm.hashtags   0.1034904  0.0286971   3.606 0.000317 ***
event          -0.0293940  0.0224639  -1.308 0.190830
link            0.0160778  0.0050464   3.186 0.001461 **
photo           0.0015604  0.0053675   0.291 0.771300
status         -0.0085057  0.0132957  -0.640 0.522408
video                  NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0697 on 2381 degrees of freedom
Multiple R-squared:  0.02641,	Adjusted R-squared:  0.01905
F-statistic: 3.589 on 18 and 2381 DF,  p-value: 4.633e-07
```