TrustLab

December 23, 2022

Assignment - Josh Silverbeck

Goal: to predict adherence to lockdown rules (as measured by a combination of stringency and use of public transport), based off social media posts

[53]:
```
# Import packages

import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.inspection import permutation_importance
import warnings
```

[54]:
```
warnings.filterwarnings('ignore')
```

[55]:
```
# Load data

# The social media data comes from 10 Kaggle files, saved as jsons

df = pd.read_json('dutch_tweets_chunk0.json.zip')

for i in range(1, 10):
    print('Now processing chunk ' + str(i))

    chunk = pd.read_json('dutch_tweets_chunk' + str(i) + '.json.zip')
    frames = [df, chunk]

    df = pd.concat(frames)
```

```
print('Dataframe dimensions: ' + str(df.shape))
df.head()
```

```
Now processing chunk 1
Now processing chunk 2
Now processing chunk 3
Now processing chunk 4
Now processing chunk 5
Now processing chunk 6
Now processing chunk 7
Now processing chunk 8
Now processing chunk 9
Dataframe dimensions: (271342, 23)
```

[55]:

```
                                        full_text  … subjective_pattern
0  @pflegearzt @Friedelkorn @LAguja44 Pardon, wol…  …                0.0
1  RT @grantshapps: Aviation demand is reduced du…  …                0.0
2  RT @DDStandaard: De droom van D66 wordt werkel…  …                0.0
3  RT @DDStandaard: De droom van D66 wordt werkel…  …                0.0
4  De droom van D66 wordt werkelijkheid: COVID-19…  …                0.0

[5 rows x 23 columns]
```

[56]:

```
# Create column with the day of the tweet in format YYYYMMDD

df['date'] = pd.to_datetime(df[['year', 'month', 'day']])
earliest_date = min(df['date'])
latest_date = max(df['date'])
```

[57]:

```
# The other data is downloaded from Our World In Data

# Mobility data
mobility = pd.read_csv('visitors-transit-covid.csv')
mobility = mobility[mobility['Entity'] == 'Netherlands'] # filter to Netherlands
mobility['Day'] = pd.to_datetime(mobility['Day'])
mobility = mobility[(mobility['Day'] >= earliest_date) & (mobility['Day'] <=␣
↪latest_date)] # filter to relevant days
print('Mobility dataframe dimensions: ' + str(mobility.shape))
mobility.head()
```

```
Mobility dataframe dimensions: (222, 4)
```

[57]:

```
            Entity Code        Day  transit_stations
78214  Netherlands  NLD 2020-02-17            -1.667
78215  Netherlands  NLD 2020-02-18            -2.000
78216  Netherlands  NLD 2020-02-19            -2.200
78217  Netherlands  NLD 2020-02-20            -2.333
```

```
78218  Netherlands  NLD 2020-02-21          -1.857
```

[58]:
```python
# The other data is downloaded from Our World In Data

# Mobility data
stringency = pd.read_csv('owid-covid-data.csv')
stringency = stringency[stringency['location'] == 'Netherlands'] # filter to␣
↪Netherlands
stringency['date'] = pd.to_datetime(stringency['date'])
stringency = stringency[(stringency['date'] >= earliest_date) &␣
↪(stringency['date'] <= latest_date)] # filter to relevant days
print('Stringency dataframe dimensions: ' + str(stringency.shape))
stringency.head()
```

```
Stringency dataframe dimensions: (212, 67)
```

[58]:
```
       iso_code  … excess_mortality_cumulative_per_million
151431      NLD  …                                     NaN
151432      NLD  …                                     NaN
151433      NLD  …                                     NaN
151434      NLD  …                              -132.77712
151435      NLD  …                                     NaN

[5 rows x 67 columns]
```

The stringency dataframe is missing the first few days of data, so we'll drop these days when merging.

[59]:
```python
target = mobility[['Day', 'transit_stations']].merge(stringency[['date',␣
↪'stringency_index']], left_on = 'Day', right_on = 'date', how = 'inner').
↪drop('Day', axis = 1)
target['stringency_index'] = target['stringency_index']/100 # put on scale of 0␣
↪to 1, not 0 to 100
target['transit_stations_scaled'] = (target['transit_stations'] - np.
↪min(target['transit_stations'])) / (np.max(target['transit_stations']) - np.
↪min(target['transit_stations']))/2 + 0.5 # scale mobility between 0 and 1
target['defiance'] = target['stringency_index'] *␣
↪target['transit_stations_scaled'] # combine the two metrics by multiplying,␣
↪on the same scale of 0 to 1 (see README)
target.head()
```

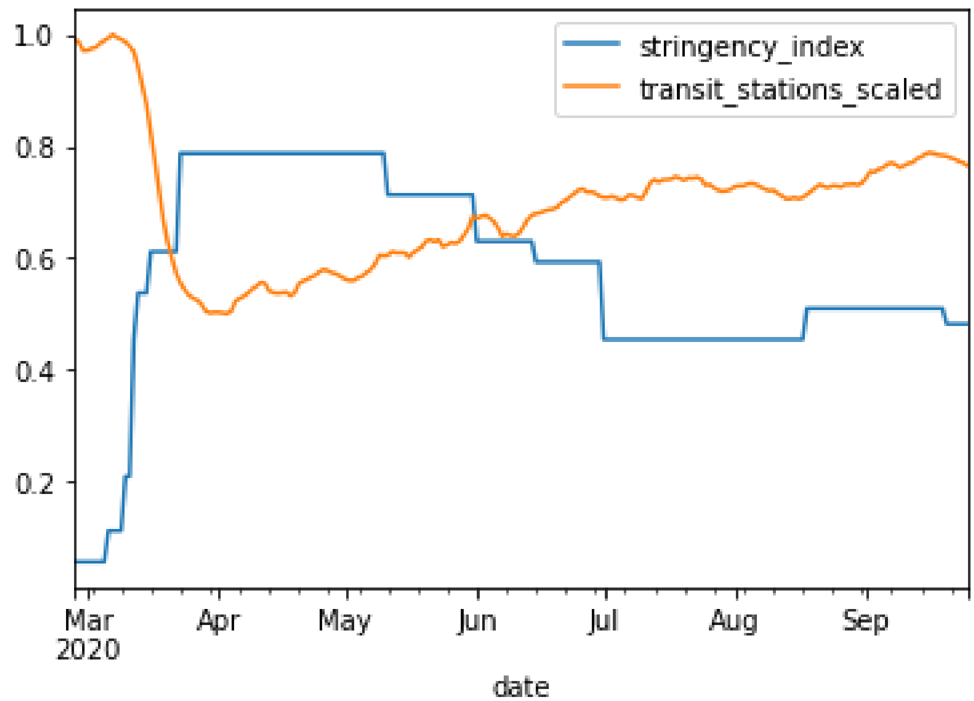
[59]:
```
   transit_stations       date  …  transit_stations_scaled  defiance
0            -1.429 2020-02-27  …                 0.992291  0.055171
1            -2.286 2020-02-28  …                 0.985684  0.054804
2            -4.143 2020-02-29  …                 0.971368  0.054008
3            -4.143 2020-03-01  …                 0.971368  0.054008
4            -3.714 2020-03-02  …                 0.974675  0.054192
```

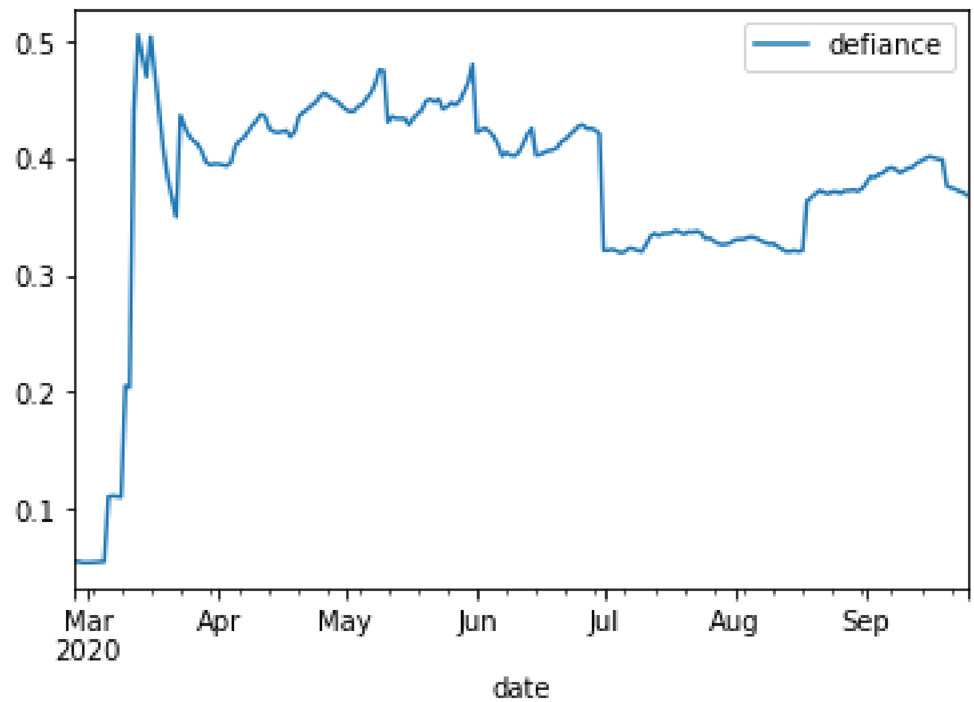
[5 rows x 5 columns]

Visualizing and defining the target

[60]:
```
target.plot('date', ['stringency_index', 'transit_stations_scaled'])
target.plot('date', 'defiance')
```

[60]: <AxesSubplot:xlabel='date'>

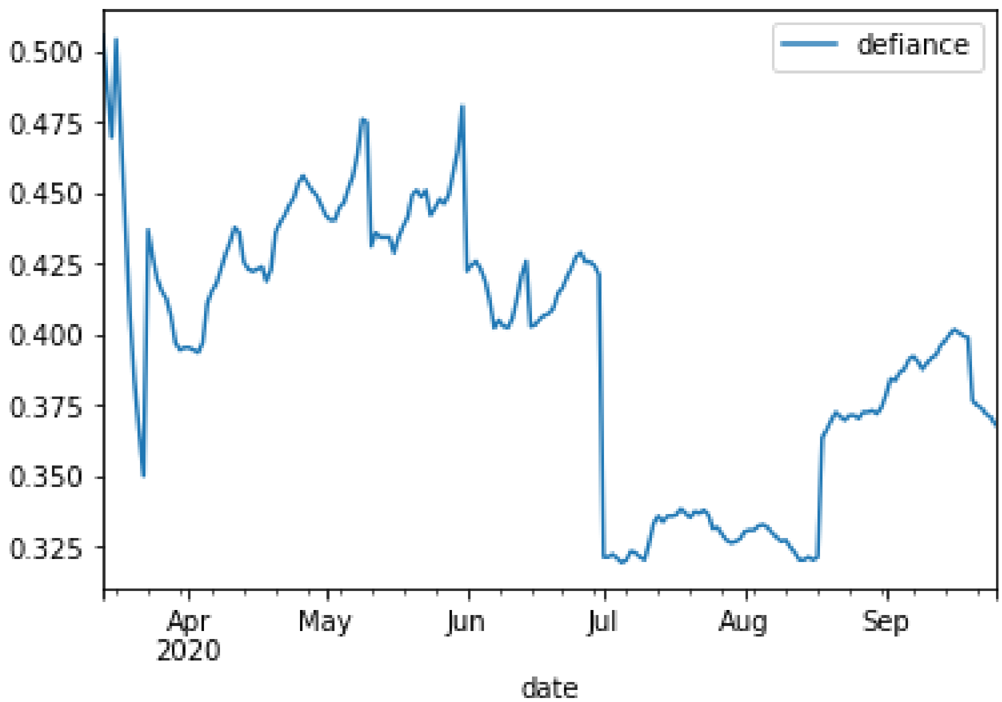




Initially you get a big jump as people adjust to the new situation. This is a different period of behaviour to later in the pandemic, so let's start on the 13th March, when the first government measures to restrict in-person meetings were announced.

[61]:
```
target = target[target['date'] >= '2020-03-13']
target.plot('date', 'defiance')
```

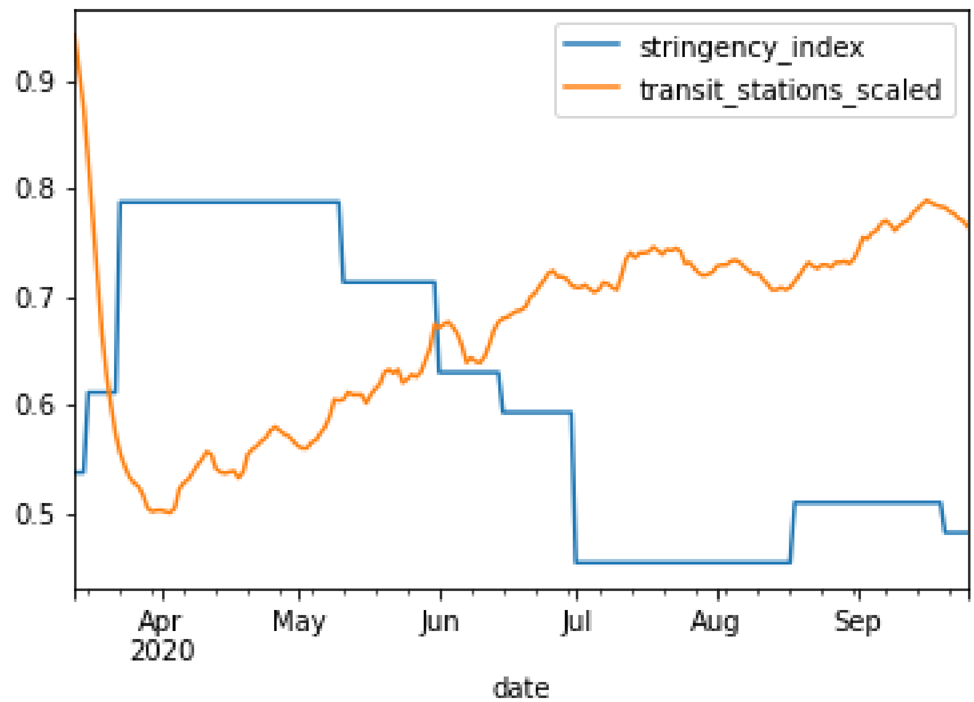
[61]:
```
<AxesSubplot:xlabel='date'>
```

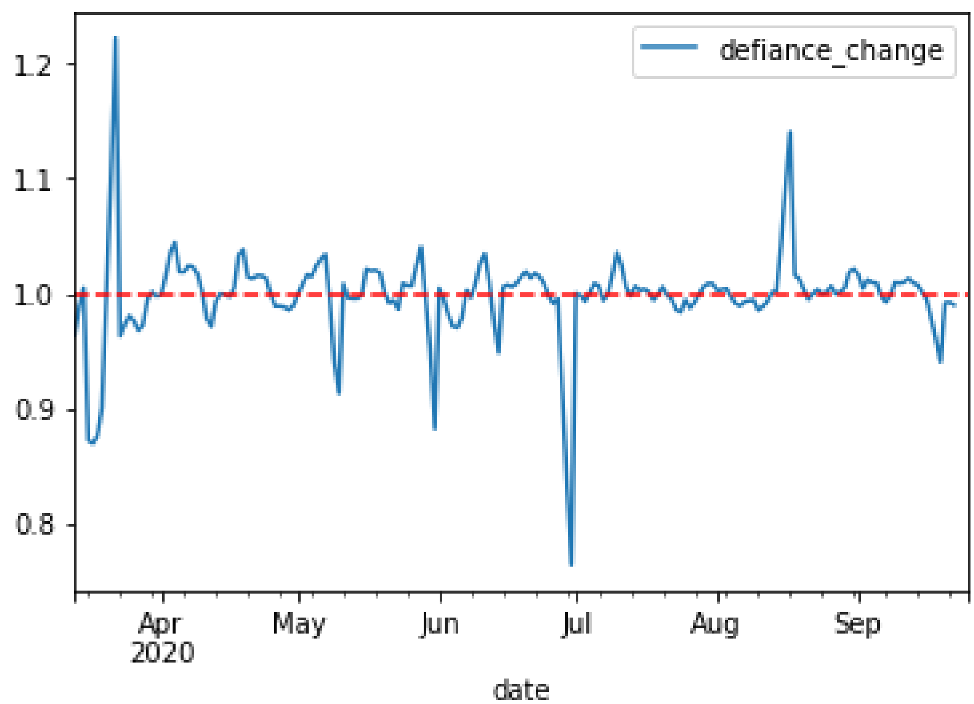


[62]:
```
# How can we spot when there is a spike? Compare the average of the defiance␣
↪over the next 3 days to the current day (as a ratio) - if the ratio is high␣
↪then there will be a sustained spike over the next 3 days

target['defiance_change'] = pd.Series(target['defiance']).rolling(window = 3,␣
↪min_periods = 1).mean().shift(-3) / target['defiance']
```

[63]:
```
target.plot('date', ['stringency_index', 'transit_stations_scaled'])
target.plot('date', 'defiance_change')
plt.axhline(y = 1, color = 'r', linestyle = '--')
```

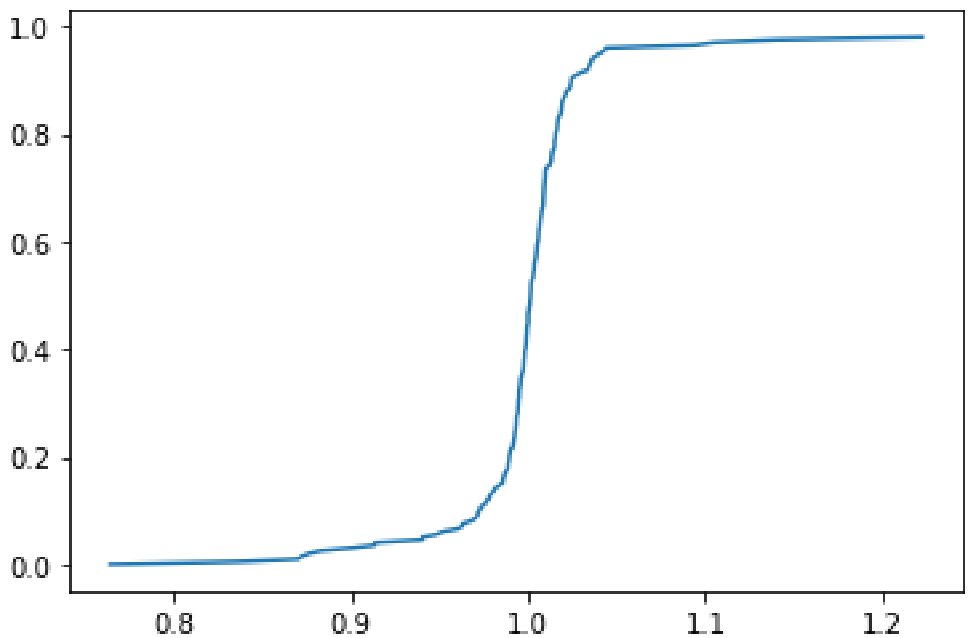
[63]: <matplotlib.lines.Line2D at 0x7ff4cd2fdb20>


stringency_index
transit_stations_scaled
0.9
0.8
0.7
0.6
0.5
Apr
2020
May
Jun
Jul
Aug
Sep
date


defiance_change
1.2
1.1
1.0
0.9
0.8
Apr
2020
May
Jun
Jul
Aug
Sep
date

[64]:
```python
# cdf of defiance_change
def cdf(x, plot=True, *args, **kwargs):
    x, y = sorted(x), np.arange(len(x)) / len(x)
    return plt.plot(x, y, *args, **kwargs) if plot else (x, y)

cdf(target['defiance_change']) # to understand distribution of change
```

[64]:
```
[<matplotlib.lines.Line2D at 0x7ff4cc00e3a0>]
```



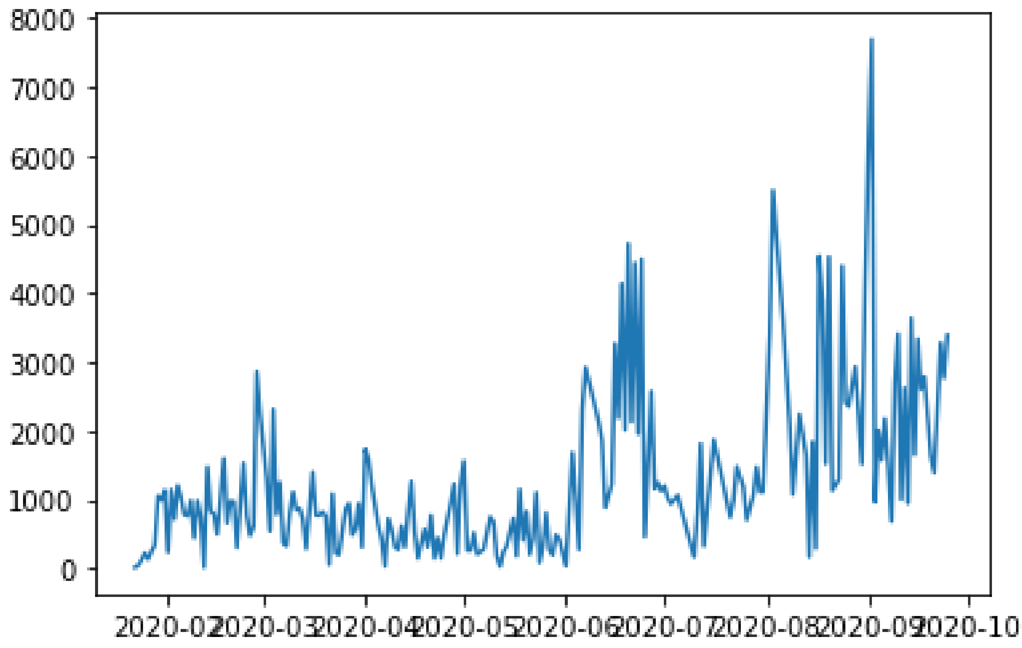
[65]:
```python
sum(target['defiance_change'] >= 1)/target.shape[0] # % of days with defiance␣
↪increasing on average in the next week - nicely balanced
```

[65]:
```
0.5329949238578681
```

Understanding Kaggle data set

[66]:
```python
# Number of posts over time

plt.plot(df[['date', 'full_text']].groupby('date').agg('count'))
```

[66]:
```
[<matplotlib.lines.Line2D at 0x7ff4cbd1eb80>]
```



[67]:
```
# What are the posts about? Relevant topics include covid and travel words

covid_key_words = ['covid', 'corona', 'virus', 'lockdown', 'pcr', 'cases',␣
 ↪'deaths', 'vaccine']
travel_key_words = ['train', 'bus', 'car', 'tram', 'meet', 'journey',␣
 ↪'transport', 'drive']
```

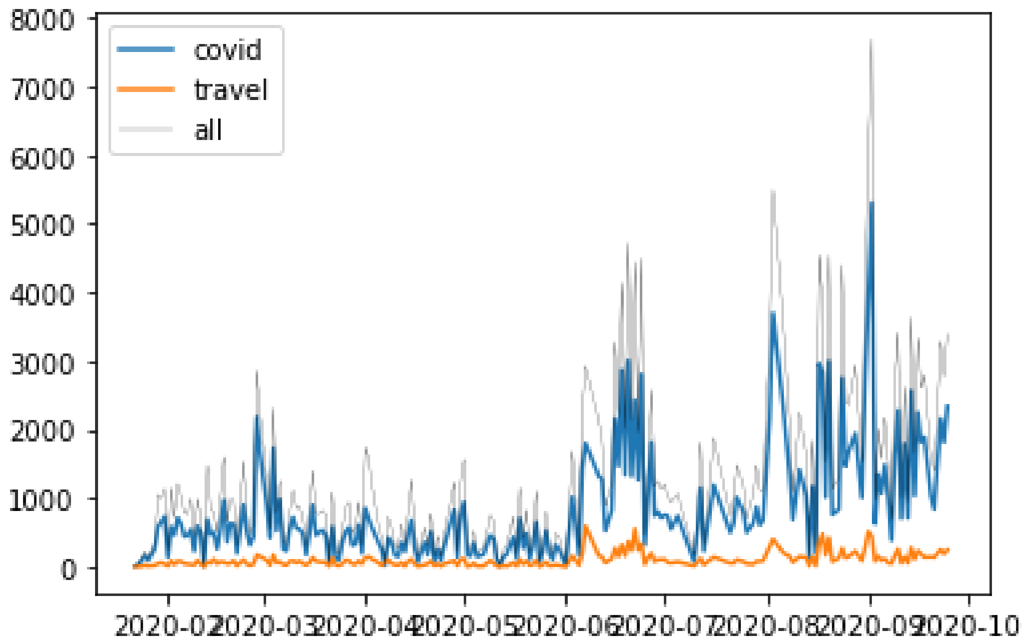
[68]:
```
# Clean text of tweets to lower-case, remove punctuation (including e.g.␣
 ↪COVID-19 to covid 19)

df['text_translation_clean'] = df['text_translation'].str.replace('[^\w\s]','␣
 ↪') # remove punctuation (may introduce double spaces but that doesn't matter␣
 ↪for now)
df['text_translation_clean'] = df['text_translation_clean'].str.lower()
```

[69]:
```
df['covid'] = df['text_translation_clean'].astype(str).apply(lambda x: any([k␣
 ↪in x for k in covid_key_words]))
df['travel'] = df['text_translation_clean'].astype(str).apply(lambda x: any([k␣
 ↪in x for k in travel_key_words]))
```

[70]:
```
plt.plot(df[df['covid']][['date', 'full_text']].groupby('date').agg('count'),␣
 ↪label = 'covid')
plt.plot(df[df['travel']][['date', 'full_text']].groupby('date').agg('count'),␣
 ↪label = 'travel')
```

```
plt.plot(df[['date', 'full_text']].groupby('date').agg('count'), linewidth = 0.
↪2, color = 'black', label = 'all')
plt.legend()
```

[70]: <matplotlib.legend.Legend at 0x7ff4cd30a1f0>



[71]:
```
# Most posts are about covid, a few are about travel:
print('% of posts about Covid: ' + str(round(100 * sum(df['covid'])/df.
↪shape[0], 0)) + '%')
print('% of posts about Travel: ' + str(round(100 * sum(df['travel'])/df.
↪shape[0], 0)) + '%')
```

```
% of posts about Covid: 64.0%
% of posts about Travel: 8.0%
```

Feature Engineering

[72]:
```
# Daily volumes of posts, split by category

daily_df = df.groupby(['date', 'weekday']).agg({'full_text': 'count',
                                                'covid': 'sum',
                                                'travel': 'sum',
                                                })\
    .reset_index().rename(columns = {'full_text':'posts'})
```

```
# We have to normalise covid and travel by the total number of posts, since␣
↪this changes each day
daily_df['covid'] = daily_df['covid'] / daily_df['posts']
daily_df['travel'] = daily_df['travel'] / daily_df['posts']
```

[73]:
```
# Volumes from the previous week, again split by category

daily_df['weekly_covid'] = pd.Series(daily_df['covid']).rolling(window = 7,␣
↪min_periods = 1).sum()
daily_df['weekly_travel'] = pd.Series(daily_df['travel']).rolling(window = 7,␣
↪min_periods = 1).sum()
```

[74]:
```
# Daily trends: day-on-day change

daily_df['covid_dod'] = daily_df['covid']/daily_df['covid'].shift(1)
daily_df['travel_dod'] = daily_df['travel']/daily_df['travel'].shift(1)
```

[75]:
```
# Weekly trends: weekly_posts / (weekly_posts 7 days ago)

daily_df['covid_wow'] = daily_df['weekly_covid']/daily_df['weekly_covid'].
↪shift(7)
daily_df['travel_wow'] = daily_df['weekly_travel']/daily_df['weekly_travel'].
↪shift(7)
```

[76]:
```
# Seasonality: dummy variable for the weekend

daily_df[['date', 'weekday']].head() # 22nd Jan 2020 was a Wednesday and␣
↪weekday = 2, so weekend is weekday = 5 or 6
daily_df['weekend'] = daily_df['weekday'].apply(lambda x: (x == 5) or (x == 6))
```

[77]:
```
# Sentiment / subjectivity

sentiment_df = df.assign(covid_sentiment = df['covid'] *␣
↪df['sentiment_pattern'],
                         covid_subjectivity = df['covid'] *␣
↪df['subjective_pattern'])\
    .groupby('date').agg({'covid_sentiment': ['mean', 'std'],
                          'covid_subjectivity': ['mean']})\
    .reset_index()
sentiment_df.columns = ['date', 'covid_sentiment_mean', 'covid_sentiment_std',␣
↪'covid_subjectivity_mean']

daily_df = daily_df.merge(sentiment_df, on = 'date', how = 'inner')
```

[78]:
```
# Covid case and death numbers (also from Our World In Data)
```

```
daily_df = daily_df.merge(stringency[['date', 'new_cases', 'new_deaths']], on =␣
↪'date', how = 'inner')
```

[79]:
```
# Add target

target['y'] = target['defiance_change'] > 1
daily_df = daily_df.merge(target[['date', 'y']], on = 'date', how = 'inner')
daily_df.head()
```

[79]:
```
        date  weekday  posts  …  new_cases  new_deaths      y
0 2020-03-13      4.0    723  …      155.0         5.0  False
1 2020-03-14      5.0    285  …      176.0         2.0  False
2 2020-03-15      6.0    843  …      278.0         8.0   True
3 2020-03-16      0.0   1401  …      292.0         4.0  False
4 2020-03-17      1.0    778  …      346.0        19.0  False

[5 rows x 18 columns]
```

[80]:
```
daily_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 160 entries, 0 to 159
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   date                     160 non-null    datetime64[ns]
 1   weekday                  160 non-null    float64
 2   posts                    160 non-null    int64
 3   covid                    160 non-null    float64
 4   travel                   160 non-null    float64
 5   weekly_covid             160 non-null    float64
 6   weekly_travel            160 non-null    float64
 7   covid_dod                160 non-null    float64
 8   travel_dod               160 non-null    float64
 9   covid_wow                160 non-null    float64
 10  travel_wow               160 non-null    float64
 11  weekend                  160 non-null    bool
 12  covid_sentiment_mean     160 non-null    float64
 13  covid_sentiment_std      160 non-null    float64
 14  covid_subjectivity_mean  160 non-null    float64
 15  new_cases                160 non-null    float64
 16  new_deaths               159 non-null    float64
 17  y                        160 non-null    bool
dtypes: bool(2), datetime64[ns](1), float64(14), int64(1)
memory usage: 21.6 KB
```

[81]:
```
daily_df.describe()
```

[81]:

```
           weekday       posts  …    new_cases  new_deaths
count  160.000000   160.00000  …   160.000000  159.000000
mean     3.006250  1435.67500  …   586.650000   36.031447
std      2.001562  1343.71378  …   546.788488   52.009057
min      0.000000    30.00000  …    36.000000    0.000000
25%      1.000000   481.00000  …   175.000000    2.000000
50%      3.000000  1043.50000  …   476.500000    8.000000
75%      5.000000  1897.00000  …   846.750000   53.000000
max      6.000000  7694.00000  …  2779.000000  234.000000

[8 rows x 15 columns]
```

Took inner merges to ensure no NAs, but there is still one row with no deaths, and one with inf travel_dod (since previous day had zero travel)

[82]:

```python
daily_df['new_deaths'].fillna(0, inplace = True) # fill with 0
daily_df.replace(np.inf, np.ma.masked_invalid(daily_df['travel_wow']).mean(),
 ↪inplace = True) # fill with mean of rest of column
```

Train/test split

When analysing features, only want to use the train set, so split here.

Not enough data to use a validation set (see READ ME).

[84]:

```python
train_rows = np.floor(daily_df.shape[0] * 0.75)
test_rows = daily_df.shape[0] - train_rows

daily_df['train_test'] = ['train'] * int(train_rows) + ['test'] * int(test_rows)
```

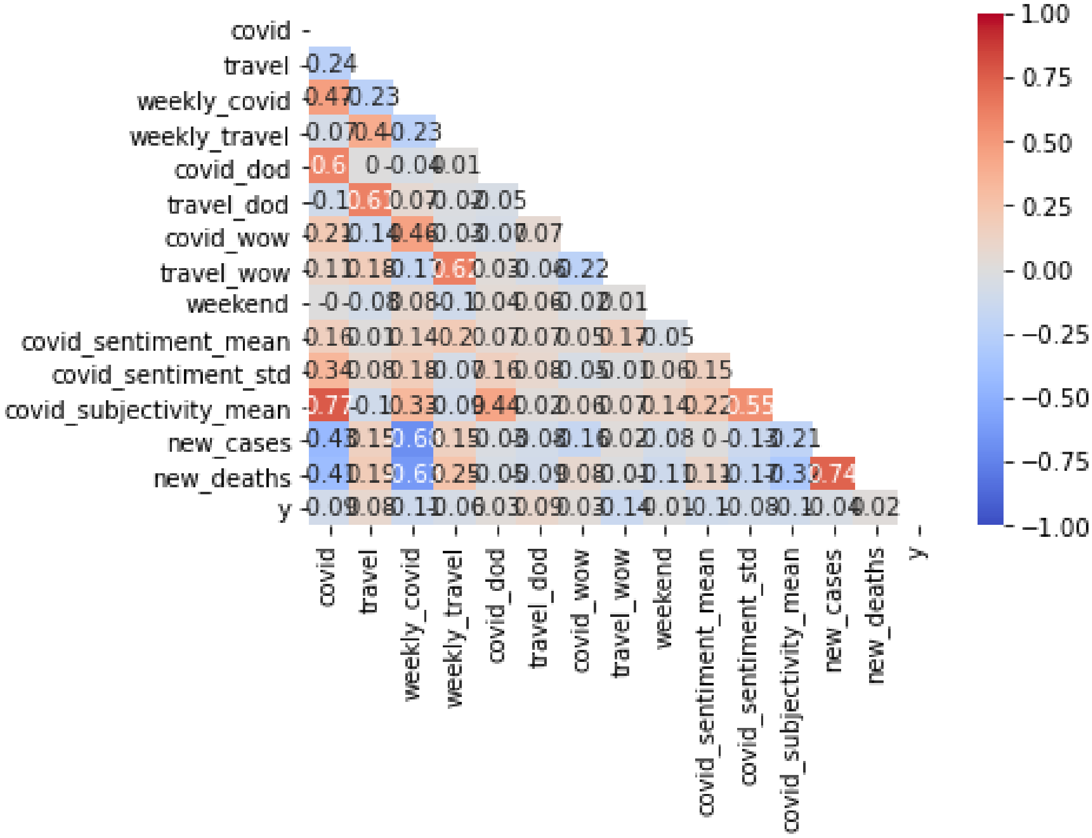
Feature Analysis

[85]:

```python
analysis_df = daily_df[daily_df['train_test'] == 'train'].drop(['date',
 ↪'weekday', 'posts', 'train_test'], axis = 1) # posts and weekday aren't
 ↪features we will use
```

[86]:

```python
# Feature correlation heatmap:

plt.rcParams['axes.facecolor'] = 'white'

corr_matrix = analysis_df.corr(method = 'spearman').round(2)
mask = np.triu(np.ones_like(corr_matrix, dtype = bool))
sns.heatmap(corr_matrix, annot = True, vmax = 1, vmin = -1, center = 0, cmap =
 ↪'coolwarm', mask = mask)
plt.show()
```



[87]:
```
# Change feature selected here to analyse different features

feature = 'new_deaths'
```

[88]:
```
sorted_mat = corr_matrix.unstack().sort_values(ascending = False)
sorted_df = pd.DataFrame(sorted_mat).reset_index()
sorted_df[(sorted_df['level_0'] == feature) & (sorted_df['level_1'] !=␣
↪feature)] # correlated features
```

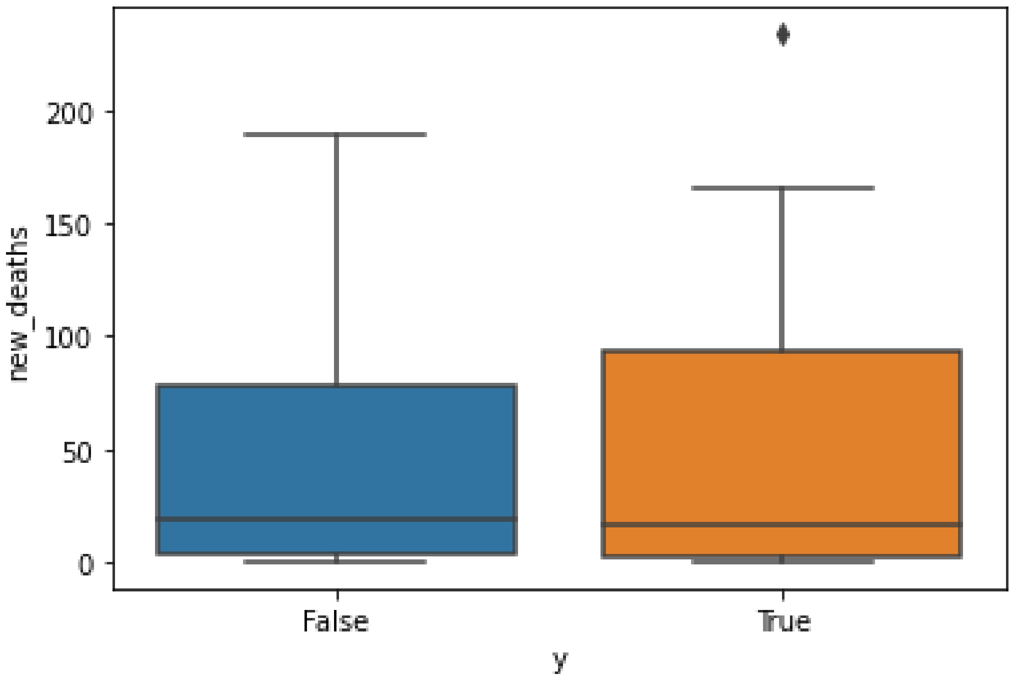
[88]:

	level_0	level_1	0
18	new_deaths	new_cases	0.74
40	new_deaths	weekly_travel	0.25
47	new_deaths	travel	0.19
72	new_deaths	covid_sentiment_mean	0.11
80	new_deaths	covid_wow	0.08
111	new_deaths	y	0.02
133	new_deaths	travel_wow	-0.01
150	new_deaths	covid_dod	-0.05
174	new_deaths	travel_dod	-0.09
190	new_deaths	weekend	-0.11

```
201  new_deaths      covid_sentiment_std -0.17
215  new_deaths  covid_subjectivity_mean -0.32
217  new_deaths                    covid -0.41
221  new_deaths             weekly_covid -0.63
```

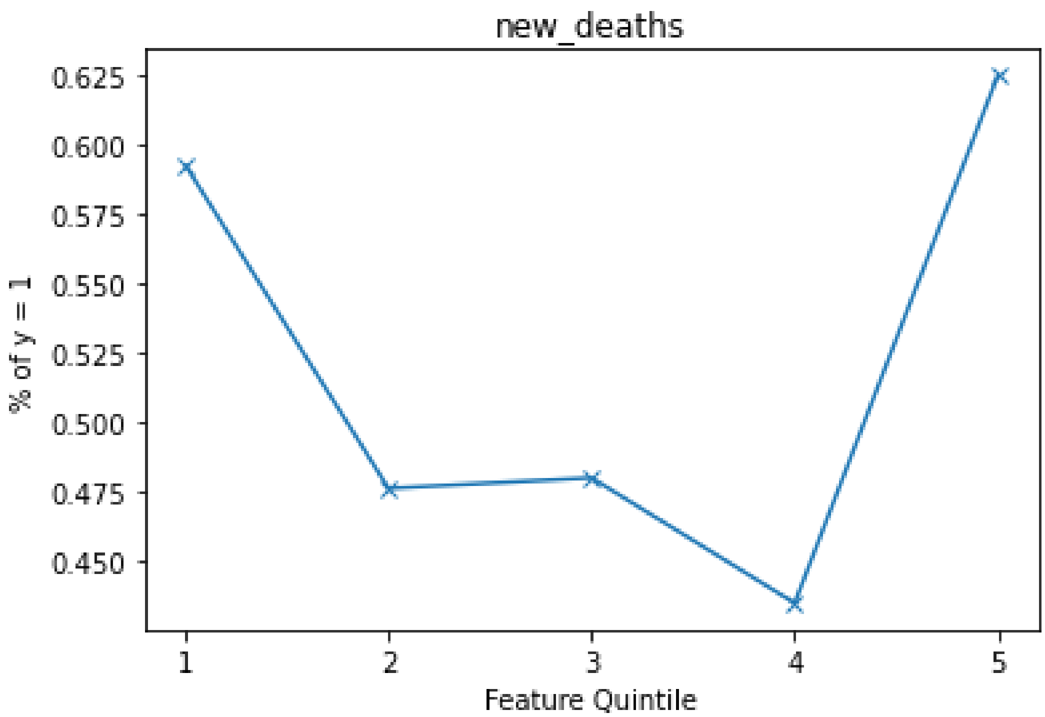
[89]:
```
# View distribution by y:

sns.boxplot(x = 'y', y = feature, data = analysis_df)
```

[89]: <AxesSubplot:xlabel='y', ylabel='new_deaths'>



[90]:
```
# Group feature values and summarise target for each:

cuts = 5

df_ntile = analysis_df[[feature, 'y']]
df_ntile['ntile'] = pd.qcut(df_ntile[feature], q = cuts, labels = range(1, cuts␣
 ↪+ 1)) # split into tiles by group

df_ntile = df_ntile.groupby('ntile').agg({'y': np.mean}).reset_index()
plt.plot([x for x in range(1,(cuts+1))], df_ntile['y'], linestyle = 'solid',␣
 ↪marker = 'x')
plt.xticks(np.arange(1, cuts + 1, 1.0))
plt.title(feature)
```

```
plt.xlabel('Feature Quintile')
plt.ylabel('% of y = 1')
plt.show()
```



Feature Selection

[91]:
```
daily_df['weekend'] = np.where(daily_df['weekend'], 1, 0) # for xgboost model
```

[92]:
```
# Comment in/out lines to include features in the model
# Features are based off analysis, but adjusted to reduce overfit / depending␣
↪on feature importance in the model

features = [
    'covid',
    'travel',

    # 'weekly_covid',
    # 'weekly_travel',

    # 'covid_dod',
    # 'travel_dod',

    # 'covid_wow',
    'travel_wow',
```

```
    # 'weekend',

    # 'covid_sentiment_mean',
    # 'covid_sentiment_std',
    # 'covid_subjectivity_mean',

    # 'new_cases',
    'new_deaths'
]
```

Modelling

[93]:
```
# Split features and target

X = daily_df[features]
y = daily_df['y']
```

[94]:
```
# Split into train and test

X_train, X_test = X[daily_df['train_test'] == 'train'],␣
 ↪X[daily_df['train_test'] == 'test']
y_train, y_test = y[daily_df['train_test'] == 'train'],␣
 ↪y[daily_df['train_test'] == 'test']
```

[95]:
```
# Fit model

# Parameters are based off grid search, but adjusted to reduce overfit
np.random.seed(123)

model = KNeighborsClassifier()
model.fit(X = X_train, y = y_train)

y_pred = model.predict(X_test)
y_pred
```

[95]:
```
array([ True, False,  True,  True, False, False, False, False,  True,
       False, False,  True, False,  True,  True,  True,  True,  True,
       False,  True,  True, False,  True, False,  True,  True, False,
       False,  True,  True,  True,  True,  True,  True,  True, False,
        True, False, False, False])
```

[96]:
```
y_pred_train = model.predict(X_train)

confusion_matrix_train = metrics.confusion_matrix(y_train, y_pred_train)

print('Train:')
```

```
print(confusion_matrix_train)
print('Accuracy = ' + str(round(100 * metrics.accuracy_score(y_train,␣
↪y_pred_train), 1)) + '%')
print('Recall = ' + str(round(100 * metrics.recall_score(y_train,␣
↪y_pred_train), 1)) + '%')
print('Precision = ' + str(round(100 * metrics.precision_score(y_train,␣
↪y_pred_train), 1)) + '%')
```

```
Train:
[[39 18]
 [16 47]]
Accuracy = 71.7%
Recall = 74.6%
Precision = 72.3%
```

[97]:
```
confusion_matrix_test = metrics.confusion_matrix(y_test, y_pred)

print('Test:')
print(confusion_matrix_test)
print('\nAccuracy = ' + str(round(100 * metrics.accuracy_score(y_test, y_pred),␣
↪1)) + '%')
print('\nRecall = ' + str(round(100 * metrics.recall_score(y_test, y_pred), 1))␣
↪+ '%')
print('\nPrecision = ' + str(round(100 * metrics.precision_score(y_test,␣
↪y_pred), 1)) + '%')
```

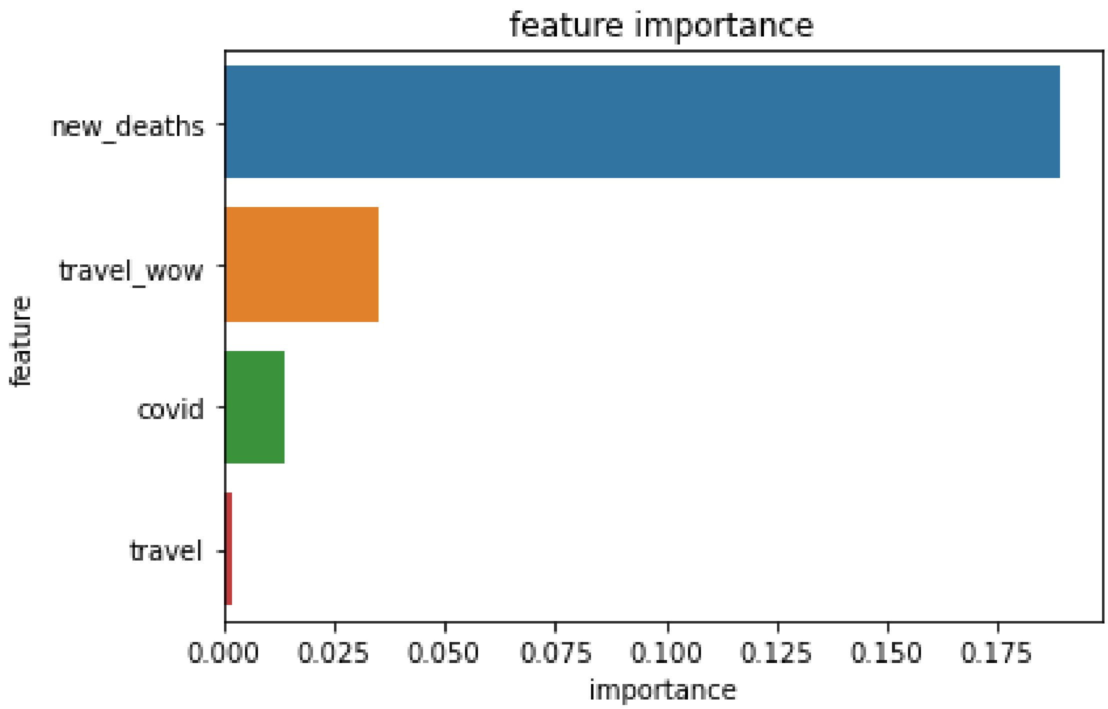
```
Test:
[[ 6  8]
 [11 15]]

Accuracy = 52.5%

Recall = 57.7%

Precision = 65.2%
```

Feature Importance

Can also be used for feature selection

[98]:
```
feature_importance = permutation_importance(model, X_train, y_train, n_repeats␣
↪= 20, random_state = 123).importances_mean
feature_importance_df = pd.DataFrame({'feature': features,
                                      'importance': feature_importance})\
    .sort_values('importance', ascending = False)

sns.barplot(x = 'importance', y = 'feature', data = feature_importance_df).
↪set_title('feature importance')
```

[98]: Text(0.5, 1.0, 'feature importance')



[ ]: