In [1]:
```
import ee
import geemap
import pandas as pd
import numpy as np
import os
import geopandas as gpd
```

In [2]:
```
ee.Initialize()
```

In [3]:
```
os.chdir('/Users/najah/work/internships/meghna//LT05_L1TP_145044_20100428_20161016_01_T1')
```

In [4]:
```
## adding points from local machine
Map = geemap.Map()
points_shp = './145044_20100428_roi/145044_20100428_500points.random_points/145044_20100428_500points.shp'
points = geemap.shp_to_ee(points_shp)
Map.addLayer(   points, {'color':'red'}, '500points')
Map.centerObject(points,8)
Map
```

```
Map(center=[23.126893770364333, 78.02312826412559], controls=(WidgetControl(options=['position', 'transparent_…
```

In [197]:
```
## distribution of points



gpd.read_file(points_shp)['class1'].value_counts()
```

Out[197]:
```
11    180
10    146
5      76
1      33
13     23
2      19
4      16
8       7
Name: class1, dtype: int64
```

In [20]:
```
cats = pd.Series(['Forest', 'Forest Fire', 'Forest Active Fire', 'Ag', "Ag Fire", 'Ag Active Fire', 'Shadow',
                  'Water', 'Cloud', 'No Data',' Others', 'Eutro', 'Unknown'
                 ])

numbers = pd.Series(list(range(1,14)))
d = {'class': numbers,'class_label': cats}
labels =pd.DataFrame(d)
labels
```

Out[20]:

	class	class_label
0	1	Forest
1	2	Forest Fire
2	3	Forest Active Fire
3	4	Ag
4	5	Ag Fire
5	6	Ag Active Fire
6	7	Shadow
7	8	Water
8	9	Cloud
9	10	No Data
10	11	Others
11	12	Eutro
12	13	Unknown

In [149]:
```
## adding the landsat image

image = ee.Image('LANDSAT/LT05/C01/T1_TOA/LT05_145044_20100428') \
            .select(['B1', 'B2', 'B3', 'B4', 'B5', 'B7'])

Map.centerObject(image)

rgbVis = {
  'min':0,
  'max' :0.4,
  'bands' : ['B5', 'B4', 'B3'],
  'gamma' : 1.2
}

Map.addLayer(image, rgbVis, 'original')
Map
```

```
Map(bottom=4023.0, center=[16.172472808397515, 96.04248046875001], controls=(WidgetControl(options=['position'…
```

In [150]:
```
# Calculate Slope and Elevation
elevation_raster = ee.Image('USGS/SRTMGL1_003')

elev = elevation_raster.select('elevation').rename('elev')
slope = ee.Terrain.slope(elevation_raster.select('elevation')).rename('slope')

##image meta data

image = image.addBands(elev).addBands(slope)
```

In [151]:
```python
# Creates a shadow band, with output 1 where pixels are illumunated and 0
# where they are shadowed. Takes as input an elevation band, azimuth and
# zenith of the light source in degrees, a neighborhood size, and whether or
# not to apply hysteresis when a shadow appears. Currently, this algorithm
# only works for Mercator projections, in which light rays are parallel.

 def getShadow(image):

  azimuth = image.get('SUN_AZIMUTH').getInfo()

  elevation = image.get('SUN_ELEVATION').getInfo()

  zenith = ee.Number(90).subtract(ee.Number(elevation)).getInfo()
  shadow = ee.Terrain.hillShadow(image, azimuth, zenith)

  return image.addBands(shadow)
image  = getShadow(image)
```

indices

In [152]:
```python
## creating the indices
def addIndices(image):

  ndvi = image.normalizedDifference(['B4', 'B3']).rename('ndvi'),

  ndmi = image.normalizedDifference(['B4', 'B5']).rename('ndmi'),

  nbr = image.normalizedDifference(['B4', 'B7']).rename('nbr'),

  ndwi = image.normalizedDifference(['B2', 'B4']).rename('ndwi'),


  bai = image.expression(
    '1/((0.1-red)**2+ (0.06-nir)**2)',{
      'nir': image.select('B4'),
      'red': image.select('B3')

    }).rename('bai'),


  mirbi = image.expression(
    '10.0 * S2 - 9.8 * S1 + 2.0',{
      'S1': image.select('B5'),
      'S2': image.select('B7')

    }).rename('mirbi'),

  baims = image.expression(
    '1.0/((0.05 - N) ** 2.0) + ((0.2 - S1) ** 2.0)',{
      'N': image.select('B4'),
      'S1': image.select('B5')
    }).rename('baims'),

  baiml = image.expression(
    '1.0/((0.05 - N) ** 2.0) + ((0.2 - S2) ** 2.0)',{
      'N': image.select('B4'),
      'S2': image.select('B7')

    }).rename('baiml'),

  gemi = image.expression(
    '((2*(B4**2-B3)**2+ (1.5*B4)+(.5*B3))/(B4+B3+.5))*(1-.25*((2*(B4**2-B3)**2+ (1.5*B4)+(.5*B3))/(B4+B3+.5)))-((R-.125)/(1-
        'R' : image.select('B3'),
        'B4': image.select('B4'),
        'B3':image.select('B3')
    }).rename('gemi')



  return image.addBands(ndvi).addBands(ndmi).addBands(nbr).addBands(ndwi).addBands(bai).addBands(baims).addBands(baiml).addE

image = addIndices(image)
```

In [168]:
```python
## mean values
geemap.image_mean_value(image.select('shadow')).getInfo()
```

Out[168]:
```
{'B1': 0.13438711952105473,
 'B2': 0.1359769791647964,
 'B3': 0.15044077551457466,
 'B4': 0.19902962299384205,
 'B5': 0.23270762806011241,
 'B7': 0.1819059863654902,
 'bai': 77.81245255450634,
 'baiml': 584.150274455091,
 'baims': 586.2317187368053,
 'elev': 447.35294980775825,
 'gemi': 0.37911833935444356,
 'mirbi': 1.5380727547614683,
 'nbr': 0.044763211784971495,
 'ndmi': -0.07666962400430449,
 'ndvi': 0.13594589189910725,
 'ndwi': -0.18180139004501383,
 'shadow': 1,
 'slope': 3.7969991376728967}
```

normalization and satandardisation

- no gemoetry defined for normalisation
- how do i normalise with a python function

In [97]:

```
def normalize(image):
  bandNames = image.bandNames()
  # Compute min and max of the image
  minDict = image.reduceRegion({
    'reducer': ee.Reducer.min(),
    'geometry': geometry,
    'scale': 10,
    'maxPixels': 1e9,
    'bestEffort': True,
    'tileScale': 16
  })
  maxDict = image.reduceRegion({
    'reducer': ee.Reducer.max(),
    'geometry': geometry,
    'scale': 10,
    'maxPixels': 1e9,
    'bestEffort': True,
    'tileScale': 16
  })
  mins = ee.Image.constant(minDict.values(bandNames))
  maxs = ee.Image.constant(maxDict.values(bandNames))

  normalized = image.subtract(mins).divide(maxs.subtract(mins))
  return normalized

#**************************************************************************
# Function to Standardize Image
# (Mean Centered Imagery with Unit Standard Deviation)
# https:#365datascience.com/tutorials/statistics-tutorials/standardization/
#**************************************************************************
def standardize(image):
  bandNames = image.bandNames()
  # Mean center the data to enable a faster covariance reducer
  # and an SD stretch of the principal components.
  meanDict = image.reduceRegion({
    'reducer': ee.Reducer.mean(),
    'geometry': geometry,
    'scale': 10,
    'maxPixels': 1e9,
    'bestEffort': True,
    'tileScale': 16
  })
  means = ee.Image.constant(meanDict.values(bandNames))
  centered = image.subtract(means)

  stdDevDict = image.reduceRegion({
    'reducer': ee.Reducer.stdDev(),
    'geometry': geometry,
    'scale': 10,
    'maxPixels': 1e9,
    'bestEffort': True,
    'tileScale': 16
  })
  stddevs = ee.Image.constant(stdDevDict.values(bandNames))

  standardized = centered.divide(stddevs)

  return standardized

standardizedImage = standardize(image)
normalizedImage = normalize(image)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [97], in <cell line: 60>()
     56   standardized = centered.divide(stddevs)
     58   return standardized
---> 60 standardizedImage = standardize(image)
     61 normalizedImage = normalize(image)

Input In [97], in standardize(image)
     32 bandNames = image.bandNames()
     33 # Mean center the data to enable a faster covariance reducer
     34 # and an SD stretch of the principal components.
     35 meanDict = image.reduceRegion({
     36   'reducer': ee.Reducer.mean(),
---> 37   'geometry': geometry,
     38   'scale': 10,
     39   'maxPixels': 1e9,
     40   'bestEffort': True,
     41   'tileScale': 16
     42 })
     43 means = ee.Image.constant(meanDict.values(bandNames))
     44 centered = image.subtract(means)

NameError: name 'geometry' is not defined
```

Main classfier

In [107]:

```
# #'formula': '10.0 * S2 - 9.8 * S1 + 2.0',
# baiml = image.expression(
#     '1.0/((0.05 - N) ** 2.0) + ((0.2 - S2) ** 2.0)',{
#       'N': image.select('B4'),
#       'S2': image.select('B7')

#     }).rename('baiml')
```

In [172]:
```python
# Use these bands for prediction.
bands = ['B1', 'B2', 'B3', 'B4', 'B5',  'B7', 'ndvi','ndmi' ,'nbr', 'ndwi',  'mirbi', 'baims', 'baiml', 'gemi','slope', 'ele

# This property of the table stores the land cover labels.
label = 'class1'

# Overlay the points on the imagery to get training.
# sample = image.select(bands).sampleRegions(
#     **{'collection': points, 'properties': [label], 'scale': 30}
# )

training = image.select(bands).sampleRegions(**{
  'collection' : points,
  'properties' : [label],
  'scale': 30
})
```

In [157]:
```python
training_gpd = geemap.ee_to_geopandas(training)

training_gpd.head()
```

Out[157]:

	geometry	B1	B2	B3	B4	B5	B7	baiml	baims	class1	elev	gemi	mirbi	nbr	r
0	None	0.145801	0.145188	0.172552	0.224978	0.285189	0.209870	32.661270	32.668430	11	528	0.383965	1.303847	0.034744	-0.11
1	None	0.140247	0.153892	0.177490	0.242910	0.266976	0.201443	26.871280	26.875763	4	555	0.390655	1.398063	0.093322	-0.04
2	None	0.129140	0.127779	0.130574	0.150261	0.198171	0.181779	99.480960	99.480632	5	318	0.355299	1.875719	-0.094924	-0.13
3	None	0.138859	0.142286	0.172552	0.227967	0.317568	0.229533	31.574257	31.587207	11	517	0.385886	1.183169	-0.003424	-0.16
4	None	0.156908	0.174203	0.204652	0.245899	0.301379	0.229533	26.058488	26.067893	11	443	0.366534	1.341825	0.034423	-0.10

In [183]:
```python
## building the classifer

# numberOfTrees: The number of decision trees to create.
#   variablesPerSplit: The number of variables per
#       split. If unspecified, uses the square root of
#       the number of variables.
#   minLeafPopulation: Only create nodes whose training
#       set contains at least this many points.
#   bagFraction: The fraction of input to bag per tree.
#   maxNodes: The maximum number of leaf nodes in each tree. If
#       unspecified, defaults to no limit.
#   seed: The randomization seed.
classifier = ee.Classifier.smileRandomForest( numberOfTrees = 500, variablesPerSplit = 5).train(training, label, bands)
```

In [192]:
```python
# Classify the image with the same bands used for training.
result = image.select(bands).classify(classifier)

# # Display the clusters with random colors.
Map.addLayer(result.randomVisualizer(), {}, 'classfied')
Map.centerObject(result)
Map
```

```
Map(bottom=420.0, center=[79.68718415450823, -76.28906250000001], controls=(WidgetControl(options=['position',…
```

accuracy

In [185]:
```python
in_shp = './145044_20100428_roi/145044_20100428_200points.random_points/145044_20100428_200points.shp'
```

In [186]:
```python
in_fc = geemap.shp_to_ee(in_shp)


out_csv = ('./145044_20100428_models/RF/145044_20100428_200points_validation.csv')
geemap.extract_values_to_points(in_fc, result, out_csv)
validation_df = pd.read_csv(out_csv)

validation_df['class1'].value_counts()
```

```
Generating URL ...
Downloading data from https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/tables/60abc1faf03565fcdb181f4
a66a7b36a-fa90ebc0eb31d96b58fb073a1bd5bac9:getFeatures
Please wait ...
Data downloaded to /Users/najah/work/internships/meghna/LT05_L1TP_145044_20100428_20161016_01_T1/145044_20100428_models/RF/
145044_20100428_200points_validation.csv
```

Out[186]:
```
11    72
10    68
5     21
1     15
13    10
4      9
2      3
8      2
Name: class1, dtype: int64
```

In [187]:
```python
validation_df['first'].value_counts()
```

Out[187]:
```
11.0    79
5.0     25
1.0     17
4.0      7
13.0     2
8.0      1
Name: first, dtype: int64
```

In [188]:
```python
cf = pd.crosstab(validation_df['first'], validation_df['class1'],rownames=['Actual'], colnames=['Predicted'], margins=True).
cf
```

Out[188]:

Predicted	1	2	4	5	8	10	11	13
Actual								
1	11	0	0	0	1	0	5	0
2	0	0	0	0	0	0	0	0
4	0	0	5	0	0	0	2	0
5	0	3	1	18	0	0	3	0
8	0	0	0	0	1	0	0	0
10	0	0	0	0	0	0	0	0
11	4	0	2	3	0	0	60	10
13	0	0	1	0	0	0	1	0

add hiilshade forest fire missclassified 11 14 haze 13 unknown 15 leaf fallen off 11 Others sandy land 13 light fires 1 truly unknown

In [189]:

```
cf / cf.sum(axis=0)
```

Out[189]:

Predicted	1	2	4	5	8	10	11	13
Actual								
1	0.733333	0.0	0.000000	0.000000	0.5	NaN	0.070423	0.0
2	0.000000	0.0	0.000000	0.000000	0.0	NaN	0.000000	0.0
4	0.000000	0.0	0.555556	0.000000	0.0	NaN	0.028169	0.0
5	0.000000	1.0	0.111111	0.857143	0.0	NaN	0.042254	0.0
8	0.000000	0.0	0.000000	0.000000	0.5	NaN	0.000000	0.0
10	0.000000	0.0	0.000000	0.000000	0.0	NaN	0.000000	0.0
11	0.266667	0.0	0.222222	0.142857	0.0	NaN	0.845070	1.0
13	0.000000	0.0	0.111111	0.000000	0.0	NaN	0.014085	0.0

In [190]:

```
np.diag(cf).sum() / cf.to_numpy().sum()
```

Out[190]:

```
0.7251908396946565
```

In [99]:

```
cf.loc['Total',:] = cf.sum(axis=0)
#Total sum per row:
cf.loc[:,'Total'] = cf.sum(axis=1)
cf
```

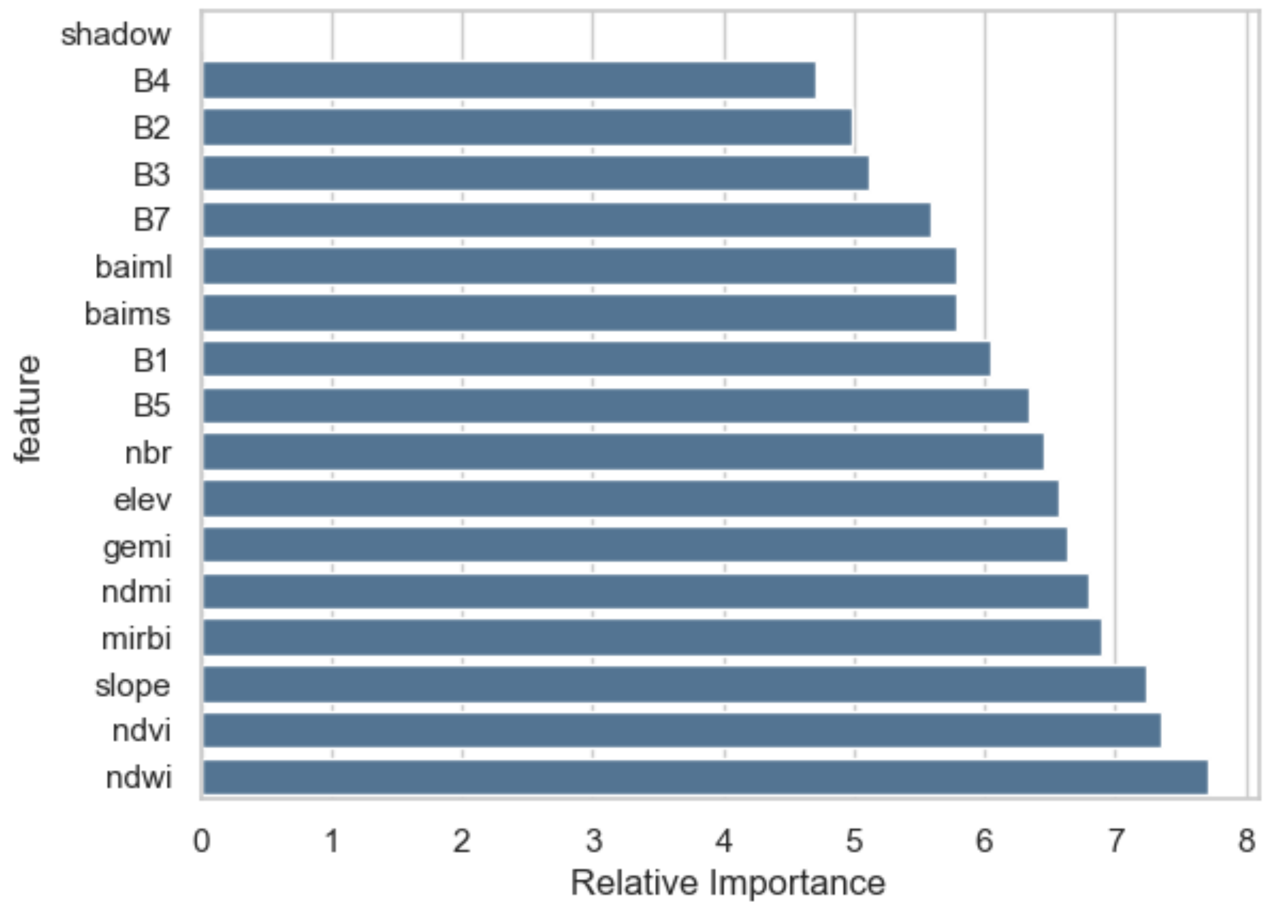
Out[99]:

Predicted	1	2	4	5	8	10	11	13	Total
Actual									
1	11.0	0.0	0.0	0.0	1.0	0.0	5.0	0.0	17.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	2.0	0.0	6.0	0.0	0.0	0.0	3.0	0.0	11.0
5	0.0	3.0	1.0	17.0	0.0	0.0	3.0	0.0	24.0
8	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	2.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	2.0	0.0	1.0	4.0	0.0	0.0	59.0	9.0	75.0
13	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	2.0
Total	15.0	3.0	9.0	21.0	2.0	0.0	71.0	10.0	131.0

feature importance

In [191]:

```
importance = ee.Dictionary(classifier.explain().get('importance'))

vals = importance.getInfo().values()

variables = importance.getInfo().keys()

imp = pd.DataFrame(vals, variables).reset_index().rename(columns = {'index': 'feature', 0: 'value'}).sort_values('value')

## calculating relative importance

imp['rel_imp'] = imp['value']*100/imp['value'].sum()

import seaborn as sns
import matplotlib.pyplot as plt
#plt.style.use('ggplot')
sns.set_theme(style="whitegrid")

ax = sns.barplot(data = imp,
           y = 'feature',
           x  = 'rel_imp',
           color = '#49759c')
ax.set(xlabel='Relative Importance')

plt.show()
```



In [141]:
```
js_snippet = """
}
"""
geemap.js_snippet_to_py(js_snippet)
```

split train and test

In [ ]:

In [148]:
```
## classifier

training = image.select(bands).sampleRegions(**{
  'collection' : training_points,
  'properties' : [label],
  'scale': 30
})
```

In [158]:
```
##Adds a column of deterministic pseudorandom numbers.
training = training.randomColumn()

split = 0.7

training_points = training.filter(ee.Filter.lt('random', split))
validation_points = training.filter(ee.Filter.gte('random', split))

training_points.first().getInfo()

validation_points.first().getInfo()
```

Out[158]:
```
{'type': 'Feature',
 'geometry': None,
 'id': '11_0',
 'properties': {'B1': 0.151354119181633,
  'B2': 0.16839967668056488,
  'B3': 0.19971393048763275,
  'B4': 0.25785401463508606,
  'B5': 0.3216153085231781,
  'B7': 0.25481507182121277,
  'baiml': 23.14938932152985,
  'baims': 23.161174912698275,
  'class1': 8,
  'mirbi': 1.3963206946849822,
  'nbr': 0.005927688907831907,
  'ndmi': -0.11003394424915314,
  'ndvi': 0.12706327438354492,
  'ndwi': -0.20986172556877136,
  'random': 0.72664157329318}}
```

In [176]:
```
classifier = ee.Classifier.smileRandomForest(100).train(training_points, 'class1', bands)
```

In [177]:
```
# Classify the image with the same bands used for training.
result = image.select(bands).classify(classifier)

# # Display the clusters with random colors.
Map.addLayer(result.randomVisualizer(), {}, 'classfied')
Map.centerObject(result)
Map
```

```
Map(bottom=229801.0, center=[21.86311093593943, 76.7732172877952], controls=(WidgetControl(options=['position'…
```

In [183]:
```
validation_points_gdp = geemap.ee_to_geopandas(validation_points)
validation_points_gdp['class1'].value_counts()
```

Out[183]:
```
11    42
5     14
1      8
2      6
8      4
13     4
4      3
Name: class1, dtype: int64
```

In [178]:
```
validated = validation_points.classify(classifier)
```

In [179]:
```
test_accuracy= validated.errorMatrix('class1', 'classification')
```

In [180]:
```
test_accuracy.accuracy().getInfo()
```

Out[180]:
```
0.6419753086419753
```

In [190]:
```
cf2 = pd.DataFrame(test_accuracy.getInfo())
cf2.loc['Total',:] = cf2.sum(axis=0)
#Total sum per row:
cf2.loc[:,'Total'] = cf2.sum(axis=1)
cf2
```

Out[190]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	8.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	6.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	3.0
5	0.0	0.0	0.0	0.0	0.0	13.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	14.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	1.0	1.0	0.0	0.0	4.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	1.0	1.0	0.0	0.0	7.0	0.0	0.0	0.0	0.0	0.0	32.0	0.0	1.0	42.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	4.0
Total	0.0	7.0	1.0	0.0	0.0	26.0	0.0	0.0	2.0	0.0	1.0	42.0	0.0	2.0	81.0

In [192]:
```
cf2= cf2.loc[:,(cf2 != 0).any(axis=0)]
cf2 =cf2.loc[(cf2 != 0).any(axis=1),:]
```

In [193]:
```
cf2.reindex(index=cf.index, columns=cf.columns)
```

Out[193]:

Predicted	1	2	4	5	8	10	11	13
Actual								
1	5.0	0.0	NaN	0.0	0.0	0.0	3.0	0.0
2	0.0	0.0	NaN	4.0	0.0	0.0	1.0	1.0
4	1.0	0.0	NaN	0.0	0.0	0.0	2.0	0.0
5	0.0	0.0	NaN	13.0	0.0	0.0	1.0	0.0
8	0.0	0.0	NaN	0.0	2.0	1.0	1.0	0.0
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	1.0	1.0	NaN	7.0	0.0	0.0	32.0	1.0
13	0.0	0.0	NaN	2.0	0.0	0.0	2.0	0.0

In [ ]: