Historical impacts of the Dawesville Cut on some aspects of water quality in the Peel-Harvey estuary: as inferred from remote sensing data using modern techniques.

Final Project for ENVT4408, Semester 2, 2024.

Ross Marriott, Student No. 19324501

School of Agriculture and Environment, The University of Western Australia.

Date: 28 October, 2024

Version 1 written in Python 3 using ESRI ArcPro v3.2.2 Notebooks.

Background

The Peel-Harvey system and the Dawesville Cut

The Peel-Harvey estuary system is comprised of the Harvey estuary and the Peel inlet, located at the southern margin of the sprawling metropolitan area of the capital city of Perth, Western Australia. Algal blooms within this system have been problematic historically, particularly during summer months of the 1980s and early 1990s. The algal blooms had resulted in noxious gas production and decaying algal wracks along various shoreline locations, which had bothered neighbouring residential communities.

The Dawesville Cut (also known as the Dawesville Channel) is a man-made channel that was constructed 1990 - April 1994 for the purpose of alleviating these problems by increasing the mixing of higher salinity coastal waters with the estuary and inlet. However, algal blooms continue to be problematic in some parts of the system.

Concurrent advancements in Remote Sensing methods

In the intervening period since the Dawesville Cut was constructed, there have been many advances in the field of Remote Sensing. Multi- and hyper-spectral sensors mounted on various remote sensing platforms, including satellites, planes, vessels, and drones have been used to capture and characterise the spectral signatures of a range of phenomena over space and time. The Landsat series of satellites, which commenced in 1972, is the longest-running of these programs. Much of these data are now freely available, and recent advances in geographic information system software and methods has led to some researchers revisiting the analysis of past events. For instance, Ho *et al*, (2017) recently used multispectral data captured by the Landsat 5 Thematic Mapper (TM) sensor during 1984-2011 to analyse historical phytoplankton blooms in Lake Erie in the United States. Bugnot *et al*, (2018) used ratios of reflectance data from different visible light spectra (blue, green, red) recorded by Landsat sensors to model historical environmental changes in Australian estuaries.

Accordingly, it is possible that historical changes in ratios of visible light reflectance for the Peel-Harvey estuary, before and after the Dawesville Cut was constructed, could reflect impacts of this intervention for improving water quality within this system.

References cited:

Bugnot, A.B., Lyons, M.B., Scanes, P., Clark, G.F., Fyfe, S.K., Lewis, E.L., & Johnson, E.L. (2018). A novel framework for the use of remote sensing for monitoring catchments at continental scales. *Journal of Environmental Management* **217**: 939-950. https://doi.org/10.1016/j.jenvman.2018.03.058

Ho, J.C., Stumpf, R.P., Bridgeman, T.B, & Michalak, A.M. (2017). Using Landsat to extend the historical record of lacustrine phytoplankton blooms: A Lake Erie case study. *Remote Sensing of Environment* **191**: 273-285. http://dx.doi.org/10.1016/j.rse.2016.12.013

Aim

To evaluate the likely impacts of the Dawesville Cut on water quality in the Peel-Harvey system by comparing and contrasting spatial patterns in the ratios of visible light spectra reflectance, before and after the Cut was constructed.

Objectives

1. Identify indices to measure historical changes in water quality within the estuary and inlet over time (likely turbidity and density of phytoplankton / organic particulates).

2. Explore seasonal changes in each index within the estuary and inlet.

3. Has there been a change which may possibly attributed to the Cut? Compare the water quality indices measured on satellite images taken two years before (i.e., 1988) and after (i.e., 1996) construction of the Dawesville Cut.

4. If there has been a change, assess spatially where these changes occurred and interpret findings (refer to StoryMap presentation for details).

Import libraries

In [ ]:
```
import os, sys, arcpy, math
from arcpy import env
from arcpy.sa import *
from arcgis.gis import GIS
import pandas as pd
import numpy as np
```

Confirm license

In [ ]:
```
gis = GIS("pro")
arcpy.CheckOutExtension("Spatial")
```

Define path variables, load shape file

Set path to inFolder based on the location of all subfolders containing downloaded raster images (from EO Browser, https://www.sentinel-hub.com/explore/eobrowser/) and outputs from running this script. Save all required imagery (separate high resolution TIFF files for Landsat 5 Bands 1-4 to a subfolder called "Input_rasters").

AOI = Shape file created beforehand in ArcGIS Pro by manually digitising around the shoreline of the Peel-Harvey estuary. This shape file will be used to mask out the surrounding land.

In [ ]:
```
inFolder = r"C:\Users\Owner\Documents\ENVT4408\Project\Data"
inFolder1 = inFolder + "\\Input_rasters" # Rasters from Landsat 5 Bands 1-4.
AOI = inFolder + "\\" + "AOI.shp"
# Checks:
print(AOI)
print(inFolder1)
```

Load and rename input rasters and apply a suitable projection

All were downloaded with WGS 1984 GCS and require projection to GDA2020 Zone 50 (EPSG7850) to ensure distances are in meters, with direction preserved.

In [5]:
```
arcpy.Describe(AOI).spatialReference
```

Out[5]:

name (Projected Coordinate System)	GDA2020_MGA_Zone_50
factoryCode (WKID)	7850
linearUnitName (Linear Unit)	Meter

spatialReference.GCS

name (Geographic Coordinate System)	GDA2020
factoryCode (WKID)	7844
angularUnitName (Angular Unit)	Degree
datumName (Datum)	GDA2020

AOI is already in this PCS so we do not need to reproject this shapefile.

The following code loops through all input rasters to firstly rename copies of them, and then to reproject those copies into the required PCS, ready for analysis.

In [ ]:

```
# Create an output geodatabase to store results (if one has not before)
outFolder = inFolder
pcs_db = os.path.join(outFolder, "pcsout.gdb")
if arcpy.Exists(pcs_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "pcsout.gdb")

# Set the PCS we want to reproject to
out_coordinate_system = arcpy.SpatialReference('GDA2020_MGA_Zone_50')

# Run the for-loop to reproject copies of input rasters and save to geodatabase
# Rename outputs to concise but informative names (e.g., "Mar_1988_Blue_pcs")
arcpy.env.overwriteOutput = True
for file in os.listdir(inFolder1):
    path = inFolder1 + "\\" + file
    Month = file[5:7]
    if Month == "03":
        Month = "Mar"
    elif Month == "07":
        Month = "Jul"
```

```
    elif Month == "09":
        Month = "Sep"
    else:
        Month = "Dec"
    Year = file[0:4]
    Type = file[-12]
    if Type == "1":
        Type = "Blue"
    elif Type == "2":
        Type = "Green"
    elif Type == "3":
        Type = "Red"
    else:
        Type = "NIR" # Near infra-red band images were also saved but not used.
    newfile_nam = Month + "_" + Year + "_" + Type
    newfile = pcs_db + "\\" + newfile_nam
    # Note: do not need to use raster file extension if saving to a gdb
    arcpy.management.ProjectRaster(path, newfile, out_coordinate_system, "BILINEAR")
    print(newfile_nam + " saved to pcsout.gdb")

# Reset default behaviour incase we inadvertently overwrite
arcpy.env.overwriteOutput = False

print("Finished copying renamed, reprojected rasters")

# Check the projection has worked for the last raster in the loop
print(newfile)
arcpy.Describe(newfile).spatialReference
```

To constrain our analyses to only the water bodies, we clip each raster to the AOI.

In [ ]:
```
# Create an output geodatabase to store results
clp_db = os.path.join(outFolder, "clippedout.gdb")
if arcpy.Exists(clp_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "clippedout.gdb")

# Run the for loop to clip the rasters to AOI and save in clippedout.gdb
arcpy.env.overwriteOutput = True
arcpy.env.workspace = pcs_db
```

```
for file in arcpy.ListDatasets():
    in_raster = pcs_db + "\\" + file
    out_raster = clp_db + "\\" + file + "_clp"
    arcpy.Clip_management(in_raster, "#", out_raster, AOI, "#", "ClippingGeometry", "MAINTAIN_EXTENT")
    print(file + "_clp saved to " + clp_db)

# Reset default behaviour incase we inadvertently overwrite
arcpy.env.overwriteOutput = False

print("Finished clipping the rasters")
```

File management: Store the clipped rasters in separate geodatabases for computations, grouped by Month and Year.

In [ ]:
```
arcpy.env.overwriteOutput = True
arcpy.env.workspace = clp_db

# Define lists for referencing within loop
yearVals = ['1988','1996'] # Pre- and Post-construction of the Cut.
monthVals = ['Mar','Jul','Sep','Dec']
# Use nested loop to create a "timeVals" list combining the above 2
# lists in the format of Month_Year, comprising all of the time points
# that we will be analysing:
timeVals = list()
for i in list(range(0,len(yearVals))):
    for j in list(range(0,len(monthVals))):
        timeVals.append(monthVals[j] + "_" + yearVals[i])
print(timeVals)

# Make geodatabases to group clipped rasters within
for t in timeVals:
    t = t + ".gdb"
    if arcpy.Exists(os.path.join(outFolder, t)):
        pass
    else:
        arcpy.CreateFileGDB_management(outFolder, t)

# Allocate each raster to its respective geodatabase
for file in arcpy.ListDatasets():
    path = clp_db + "\\" + file
    Year = file[4:8]
    Month = file[0:3]
```

```
    t_group = Month + "_" + Year + ".gdb"
    new_db = os.path.join(outFolder, t_group)
    newfile = new_db + "\\" + file
    arcpy.management.CopyRaster(path, newfile, pixel_type="64_BIT")

arcpy.env.overwriteOutput = False
print("Finished sorting files")
```

Spatial analyses

Calculate the spectral index rasters and save

In [ ]:
```
# Create an output geodatabases to store results
g_on_r_db = os.path.join(outFolder, "g_on_rout.gdb")
if arcpy.Exists(g_on_r_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "g_on_rout.gdb")

b_on_r_db = os.path.join(outFolder, "b_on_rout.gdb")
if arcpy.Exists(b_on_r_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "b_on_rout.gdb")
print("Spectral index geodatabases created")

# Create list of geodatabases for previous groupings of rasters by Month and Year
group_dbs = list()
for i in list(range(0,len(timeVals))):
    group_dbs.append(timeVals[i] + ".gdb")
print(group_dbs)

# Run the for loop to calculate the 2 indices for each group, then save
arcpy.env.overwriteOutput = True
for g in list(range(0,len(group_dbs))):
    gdb_file = group_dbs[g]
    Month_Year = gdb_file[0:8]
    print("Processing rasters for " + Month_Year + " group")
    gdb_path = os.path.join(outFolder, gdb_file)
    arcpy.env.workspace = gdb_path
    in_rasters = arcpy.ListDatasets()
```

```
    for file in in_rasters:
        Type = file[9:-4]
        if Type == "Red":
            red_rast = file
        elif Type == "Blue":
            blu_rast = file
        elif Type == "Green":
            gre_rast = file
        else:
            nir_rast = file # not used.
    # 1. Calculate Green on Red index
    g_on_r = g_on_r_db + "\\" + Month_Year + "_g_on_r"
    g_on_r_rast = Divide(gre_rast, red_rast)
    arcpy.management.CopyRaster(g_on_r_rast, g_on_r, pixel_type="64_BIT")
    print("Green on Red calculated")
    # 2. Calculate Blue on Red index
    b_on_r = b_on_r_db + "\\" + Month_Year + "_b_on_r"
    b_on_r_rast = Divide(blu_rast, red_rast)
    arcpy.management.CopyRaster(b_on_r_rast, b_on_r, pixel_type="64_BIT")
    print("Blue on Red calculated")

arcpy.env.overwriteOutput = False

print("Finished calculating spectral indices")
```

Compare Green on Red and Blue on Red indices across time using Image Regression.

We will compare values at a random sample of 100 points within the estuary, defined by the AOI, at each successive time point with the earliest (March 1988). The random samples will be taken a minimum distance of 500m apart, to reduce prospect for spatial autocorrelation affecting results and ensure that the points have good representation across the estuary. The resulting output, for each index, will be a table of values for every sampled point, for each time point (columns). Out of convenience, we then convert this table into an Excel file and export it so that we may analyse it in a different software (R). (Note: this could also be done here using Python, but I have chosen R to do this because of my greater familiarity with statistical functions in that software). A correlation matrix will be constructed to compare the strength of pairwise linear associations between index values obtained across the same locations between different times (**(i)** between different months in the same year to assess seasonal differences; and **(ii)** between different years for the same month to assess changes before and after construction of the Cut). For each month, a linear regression will also be fitted to compare values at the sampled points between the 1988 raster (X axis) and 1996 raster (Y axis), and the slope and intercept estimates, with their 95% confidence intervals, will be saved, for plotting. Image Regression is useful to complement the correlation analyses because we can make more specific inferences about comparison of the spatial distribution of index values over time. For instance, an estimate of slope not significantly different from 1 and intercept not significantly different from 0 would suggest that the spatial distribution of values has not changed over time (contingent upon the statistical assumptions of this simple model being met).

Note: Since the random seed has not been set, running the following cell will result in a different random sample of points, and therefore different estimates from the resulting correlation and regression analyses. The assumption is that a different random sample would not substantively change the results, considering that all such random samples would suitably represent the spatial distribution of index values in the estuary and inlet captured at each time point. One extension of this study would be to investigate the sensitivity to results of this assumption.

In [ ]:
```
# Create the random sample of 100 points.

# NOTE: If you re-run and overwrite randompts.shp you're likely to get different
#       results, generated from a new set of randomly allocated points. That is why
#       arcpy.env.overwriteOutput is not set to True for this cell.

arcpy.management.CreateRandomPoints(outFolder, "randompts", AOI,
                                    number_of_points_or_field = 100,
                                    minimum_allowed_distance = 500)
```

In [ ]:
```
# Sample points from each raster, separately for each index, and save to table.

randompts = outFolder + "\\" + "randompts.shp"

# Specify variables for output feature classes
sample_g_on_r = g_on_r_db + "\\" + "sample_g_on_r"
sample_b_on_r = b_on_r_db + "\\" + "sample_b_on_r"

arcpy.env.overwriteOutput = True

# Green on Red index
arcpy.env.workspace = g_on_r_db
raster_list = arcpy.ListDatasets()
Sample(raster_list, randompts, sample_g_on_r)

print("Finished sampling values from Green on Red rasters")

# Blue on Red index
arcpy.env.workspace = b_on_r_db
raster_list = arcpy.ListDatasets()
Sample(raster_list, randompts, sample_b_on_r)
```

```
arcpy.env.overwriteOutput = False

print("Finished sampling values from Blue on Red rasters")
```

In [ ]:
```
# Convert output to an Excel table and export.

# Create an output folder location to store results
regress_dir = os.path.join(outFolder, "image_regress_out")
if os.path.exists(regress_dir):
    pass
else:
    os.mkdir(regress_dir)

arcpy.env.overwriteOutput = True

g_on_r_out_xls = regress_dir + "\\" + "g_on_r_100_randpts.xls"
b_on_r_out_xls = regress_dir + "\\" + "b_on_r_100_randpts.xls"

arcpy.conversion.TableToExcel(sample_g_on_r, g_on_r_out_xls)
arcpy.conversion.TableToExcel(sample_b_on_r, b_on_r_out_xls)

arcpy.env.overwriteOutput = False

print("Excel files with raster values ready and saved in " + regress_dir)
```

Although we have done the image regressions, which analyse whether the distribution of values has changed, it would also be good to look at how the average values have changed. We can do this in a convenient way by extracting from the metadata for each raster the mean and standard deviation of the pixel values (using the "GetRasterProperties" function).

In [ ]:
```
# Calculate the mean and standard deviation of g_on_r and b_on_r rasters, and save.

# Create an output folder location to store results
means_dir = os.path.join(outFolder, "mean_stdev_indices_out")
if os.path.exists(means_dir):
    pass
else:
    os.mkdir(means_dir)

arcpy.env.overwriteOutput = True
```

```python
# Green on Red index
g_on_r_names = list()
g_on_r_means = list()
g_on_r_stdevs = list()
arcpy.env.workspace = g_on_r_db
raster_list = arcpy.ListDatasets()
for in_raster in raster_list:
    file = g_on_r_db + "\\" + in_raster
    meanval = arcpy.management.GetRasterProperties(file, "MEAN")
    stdevval = arcpy.management.GetRasterProperties(file, property_type = "STD")
    g_on_r_names.append(in_raster[0:8])
    g_on_r_means.append(meanval)
    g_on_r_stdevs.append(stdevval)

g_on_r_stats = pd.DataFrame({'Month_Year': g_on_r_names,
                            'Mean': g_on_r_means, 'StDev': g_on_r_stdevs})
g_on_r_statsout = means_dir + "\\" + "mean_stdev_g_on_r.xlsx"
g_on_r_stats.to_excel(g_on_r_statsout)
print("Green on Red index statistics computed for each raster:\n")
print(g_on_r_stats)

## Blue on Red index
b_on_r_names = list()
b_on_r_means = list()
b_on_r_stdevs = list()
arcpy.env.workspace = b_on_r_db
raster_list = arcpy.ListDatasets()
for in_raster in raster_list:
    file = b_on_r_db + "\\" + in_raster
    meanval = arcpy.management.GetRasterProperties(file, "MEAN")
    stdevval = arcpy.management.GetRasterProperties(file, "STD")
    b_on_r_names.append(in_raster[0:8])
    b_on_r_means.append(meanval)
    b_on_r_stdevs.append(stdevval)

b_on_r_stats = pd.DataFrame({'Month_Year': b_on_r_names,
                            'Mean': b_on_r_means, 'StDev': b_on_r_stdevs})
b_on_r_statsout = means_dir + "\\" + "mean_stdev_b_on_r.xlsx"
b_on_r_stats.to_excel(b_on_r_statsout)
print("\nBlue on Red index statistics computed for each raster:\n")
print(b_on_r_stats)
```

```
arcpy.env.overwriteOutput = False
```

Using map algebra we subtract the 1988 from the 1996 raster for each month and spectral index to obtain rasters showing the net change over that time period in the index value on a map, for each respective month.

Note: for convenience, I have used a Python dictionary structure to temporarily store together rasters of the same year and then refer to them on lines 38 and 39 using the corresponding Month key, to ensure that I am peforming the subtraction on rasters for the same month.

In [ ]:

```
# For each index and month, calculate the difference rasters and store them.

# Create an output geodatabases to store results
g_on_rdiff_db = os.path.join(outFolder, "g_on_rdiffout.gdb")
if arcpy.Exists(g_on_rdiff_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "g_on_rdiffout.gdb")

b_on_rdiff_db = os.path.join(outFolder, "b_on_rdiffout.gdb")
if arcpy.Exists(b_on_rdiff_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "b_on_rdiffout.gdb")

arcpy.env.overwriteOutput = True

# Green on Red index
g_on_r_1988 = {} # create empty dictionaries
g_on_r_1996 = {}
arcpy.env.workspace = g_on_r_db
raster_list = arcpy.ListDatasets()
for in_raster in raster_list:
    file = g_on_r_db + "\\" + in_raster
    Month = in_raster[0:3]
    Year = in_raster[4:8]
    # Store each raster for that year with its
    # corresponding 'Month' key in dictionary:
    if Year == "1988":
        g_on_r_1988[Month] = file
    else:
```

```
        g_on_r_1996[Month] = file
# Calculate the difference rasters
for i in list(range(0,len(monthVals))):
    Month = monthVals[i]
    # We use dictionary key to extract rasters
    # for the same Month in different years:
    Month_88 = g_on_r_1988[Month]
    Month_96 = g_on_r_1996[Month]
    # Calculate the difference between years
    # controlling for time of year (same Month):
    Month_diff = Minus(Month_96, Month_88)
    # Save outputs to name-specific geodatabase:
    Month_diffout = g_on_rdiff_db + "\\" + "g_on_rdiff_" + Month
    arcpy.management.CopyRaster(Month_diff, Month_diffout, pixel_type="64_BIT")
    print("g_on_rdiff_" + Month + " raster calculated and saved to " + g_on_rdiff_db)

# Blue on Red index
b_on_r_1988 = {} # create empty dictionaries
b_on_r_1996 = {}
arcpy.env.workspace = b_on_r_db
raster_list = arcpy.ListDatasets()
for in_raster in raster_list:
    file = b_on_r_db + "\\" + in_raster
    Month = in_raster[0:3]
    Year = in_raster[4:8]
    # Store each raster for that year with its
    # corresponding 'Month' key in dictionary:
    if Year == "1988":
        b_on_r_1988[Month] = file
    else:
        b_on_r_1996[Month] = file
# Calculate the difference rasters
for i in list(range(0,len(monthVals))):
    Month = monthVals[i]
    # We use dictionary key to extract rasters
    # for the same Month in different years:
    Month_88 = b_on_r_1988[Month]
    Month_96 = b_on_r_1996[Month]
    # Calculate the difference between years
    # controlling for time of year (same Month):
    Month_diff = Minus(Month_96, Month_88)
    # Save outputs to name-specific geodatabase:
```

```
    Month_diffout = b_on_rdiff_db + "\\" + "b_on_rdiff_" + Month
    arcpy.management.CopyRaster(Month_diff, Month_diffout, pixel_type="64_BIT")
    print("b_on_rdiff_" + Month + " raster calculated and saved to " + b_on_rdiff_db)

arcpy.env.overwriteOutput = False

print("Finished calculating difference rasters. Make sure to check histogram distributions \nprior to next step.")
```

Finally we address the question, where are the differences in index value, for each month? We use hotspot method to show on the change maps where the increases and decreases have occurred.

In [10]:

```
# Run hotspot analysis on the difference rasters, for each index and month.

# Create an output geodatabases to store results
g_on_rhotsp_db = os.path.join(outFolder, "g_on_rhotspout.gdb")
if arcpy.Exists(g_on_rhotsp_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "g_on_rhotspout.gdb")

b_on_rhotsp_db = os.path.join(outFolder, "b_on_rhotspout.gdb")
if arcpy.Exists(b_on_rhotsp_db):
    pass
else:
    arcpy.CreateFileGDB_management(outFolder, "b_on_rhotspout.gdb")

arcpy.env.overwriteOutput = True

# Green on Red index
arcpy.env.workspace = g_on_rdiff_db
raster_list = arcpy.ListDatasets()
for in_raster in raster_list:
    file = g_on_rdiff_db + "\\" + in_raster
    Month = in_raster[-3:]
    # Create points from the raster
    g_on_r_pts = g_on_rhotsp_db + "\\" + "g_on_r_" + Month + "_pts"
    arcpy.RasterToPoint_conversion(file, g_on_r_pts, raster_field="Value")
    # Create hotspot from points
    g_on_r_hotsp = g_on_rhotsp_db + "\\" + "g_on_r_" + Month + "_hotsp"
    arcpy.HotSpots_stats(g_on_r_pts, "grid_code", g_on_r_hotsp,
                         Conceptualization_of_Spatial_Relationships="FIXED_DISTANCE_BAND",
```

```
                          Distance_Method="EUCLIDEAN_DISTANCE", Standardization="NONE",
                          Distance_Band_or_Threshold_Distance="", Self_Potential_Field="",
                          Weights_Matrix_File="",
                          Apply_False_Discovery_Rate__FDR__Correction="NO_FDR")
    print("Hotspot map for " + in_raster + " created and saved")

# Blue on Red index
arcpy.env.workspace = b_on_rdiff_db
raster_list = arcpy.ListDatasets()
for in_raster in raster_list:
    file = b_on_rdiff_db + "\\" + in_raster
    Month = in_raster[-3:]
    # Create points from the raster
    b_on_r_pts = b_on_rhotsp_db + "\\" + "b_on_r_" + Month + "_pts"
    arcpy.RasterToPoint_conversion(file, b_on_r_pts, raster_field="Value")
    # Create hotspot from points
    b_on_r_hotsp = b_on_rhotsp_db + "\\" + "b_on_r_" + Month + "_hotsp"
    arcpy.HotSpots_stats(b_on_r_pts, "grid_code", b_on_r_hotsp,
                          Conceptualization_of_Spatial_Relationships="FIXED_DISTANCE_BAND",
                          Distance_Method="EUCLIDEAN_DISTANCE", Standardization="NONE",
                          Distance_Band_or_Threshold_Distance="", Self_Potential_Field="",
                          Weights_Matrix_File="",
                          Apply_False_Discovery_Rate__FDR__Correction="NO_FDR")
    print("Hotspot map for " + in_raster + " created and saved")

arcpy.env.overwriteOutput = False

print("Finished calculating hotspot vector files from the difference rasters")
```

```
Hotspot map for g_on_rdiff_Mar created and saved
Hotspot map for g_on_rdiff_Jul created and saved
Hotspot map for g_on_rdiff_Sep created and saved
Hotspot map for g_on_rdiff_Dec created and saved
Hotspot map for b_on_rdiff_Mar created and saved
Hotspot map for b_on_rdiff_Jul created and saved
Hotspot map for b_on_rdiff_Sep created and saved
Hotspot map for b_on_rdiff_Dec created and saved
Finished calculating hotspot vector files from the difference rasters
```