Using Python to Map of the Impacts of Geology and Fluvial Processes on the Development of Drainage Networks on Mars

Principles of Geocomputing 5561

> The synopsis of this project is to map out linear geological features (ligaments) on the surface of Mars to show evidence of past fluvial/drainage activity (with some emphasis on Valles Marineris and Olympus Mons). Using original Mars DEMs and three raster analytic layers, to find evidence; these include: watershed, hillshade, and slope, there is substantial geological evidence that these networks determined the fluvial processes. Having performed the digitization and analysis in ArcMap for a prior undergraduate project, the goal here is to simply automate the maps with Python and visualize them. The whole project itself utilized statistical analysis in simulating three-dimensional imagery, calculating indices, and assessing differences in terrain, atmospheric pressure, oxidation, and night/day cycles. In conjunction with Earth-Sciences Department at Bemidji State University.
>
> Alexander Danielson danie861

In [1]:

```
import rasterio #Open imagery to be displayed
import numpy as np #Allows for arrays and matrices
import matplotlib.pyplot as plt #Allows to plot the raster/tif files
import geopandas #Display the rows and columns in the shapefiles
import matplotlib as mpl
import geopandas as gpd #Plot out geological linear features
from rasterio.plot import plotting_extent #Sets the plotting extent of geological linea
from matplotlib.colors import BoundaryNorm, LinearSegmentedColormap
import numpy as np



mars = rasterio.open('Mars_MGS_MOLA_Shade_global_463m.tif') #Opens Mars imagery from NA
watershed = rasterio.open('Watersh_Flow12.tif') #Watersheds of Mars to delinate areas o
slopemars = rasterio.open('slopemars1.tif') #Slope for steepness of terrain
hillshade = rasterio.open('hillelemars1.tif') #Hillshade for downhill direction of terr
ligament = ('Ligament_Lines.shp') #Assigned ligaments
slopelinear = ('SlopeLinear.shp')
hillshadeline = ('HillshadeLinear.shp')

mars = mars.read() #Assigned variables to map
watershed = watershed.read()
slopemars = slopemars.read()
hillshade = hillshade.read()

print(mars.shape) #Prints out the dimensions of the Mars DEM
print(np.amin(mars[0]))
print(np.amax(mars[0]))
print(np.amax(mars[0]) + abs(np.amin(mars[0])))

print(watershed.shape) #Prints out the dimensions of the Mars Watershed
```

```
print(np.amin(watershed[0]))
print(np.amax(watershed[0]))
print(np.amax(watershed[0]) + abs(np.amin(watershed[0])))

print(slopemars.shape) #Prints out the dimensions of the Mars Tharsis Area via Slope
print(np.amin(slopemars[0]))
print(np.amax(slopemars[0]))
print(np.amax(slopemars[0]) + abs(np.amin(slopemars[0])))

print(hillshade.shape) #Prints out the dimensions of the Mars Tharsis Area via Hillshad
print(np.amin(hillshade[0]))
print(np.amax(hillshade[0]))
print(np.amax(hillshade[0]) + abs(np.amin(hillshade[0])))
```

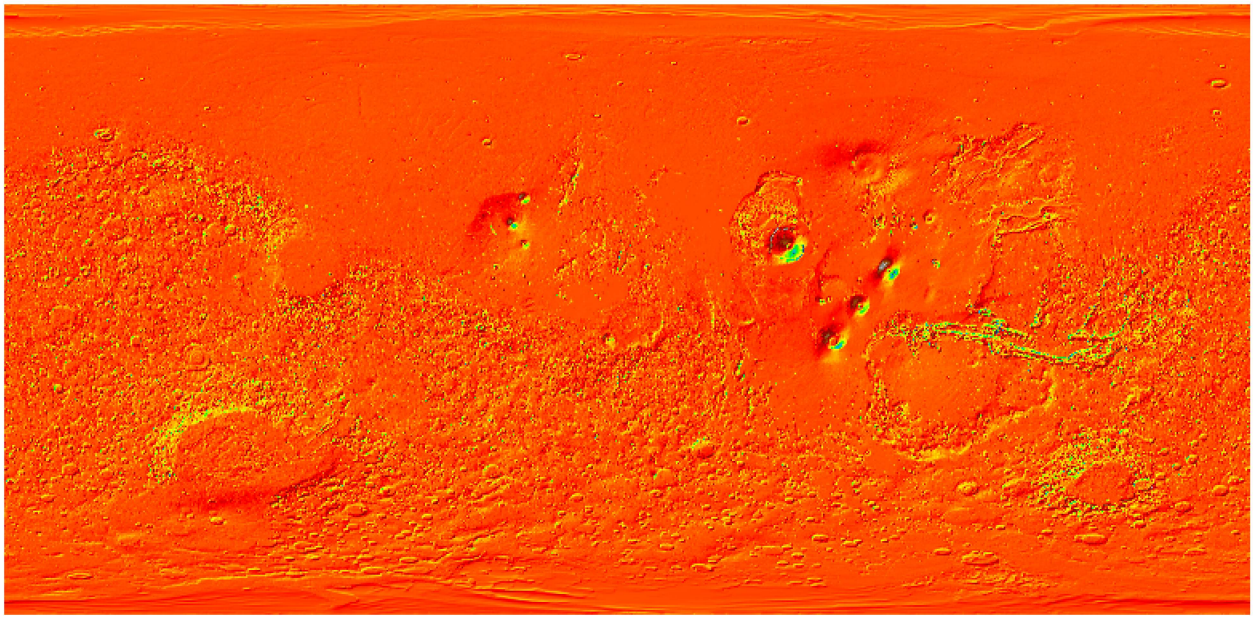
```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
In  [1]:
Line 1:     import rasterio #Open imagery to be displayed

ModuleNotFoundError: No module named 'rasterio'
---------------------------------------------------------------------------
```

In [3]:

```
# Reads the shapefiles for the geological features to be displayed
gdfligament = gpd.read_file('Ligament_Lines.shp')
gdfhillshade = gpd.read_file('HillshadeLinear.shp')
gdfslope = gpd.read_file('SlopeLinear.shp')
```

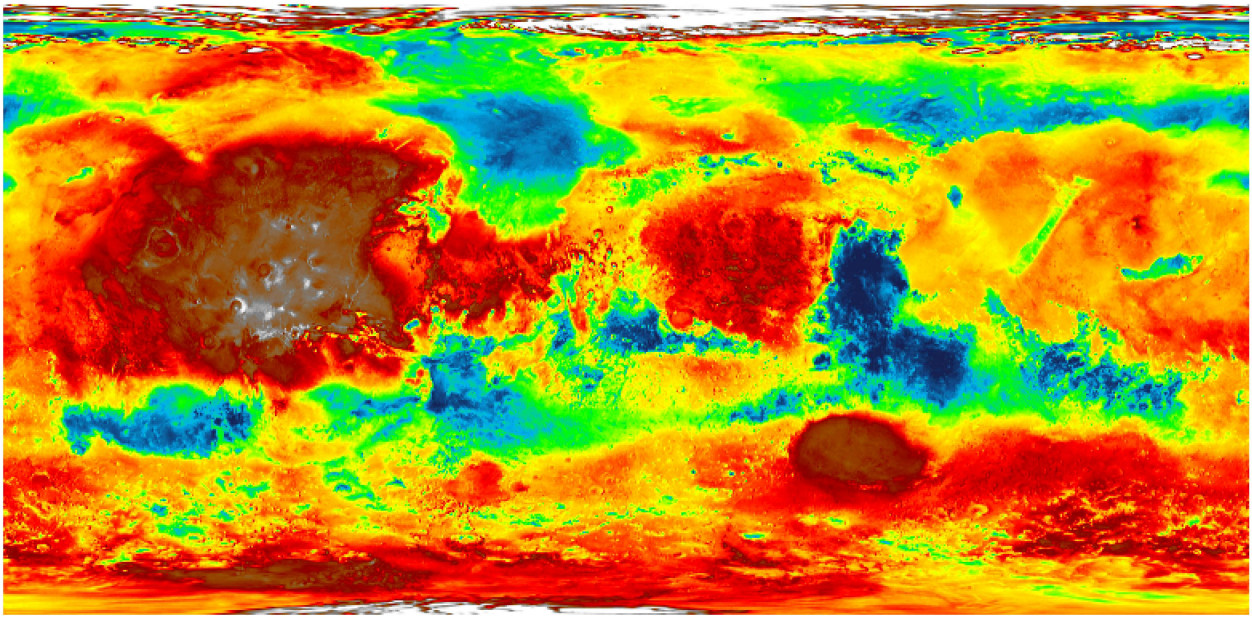
In [4]:

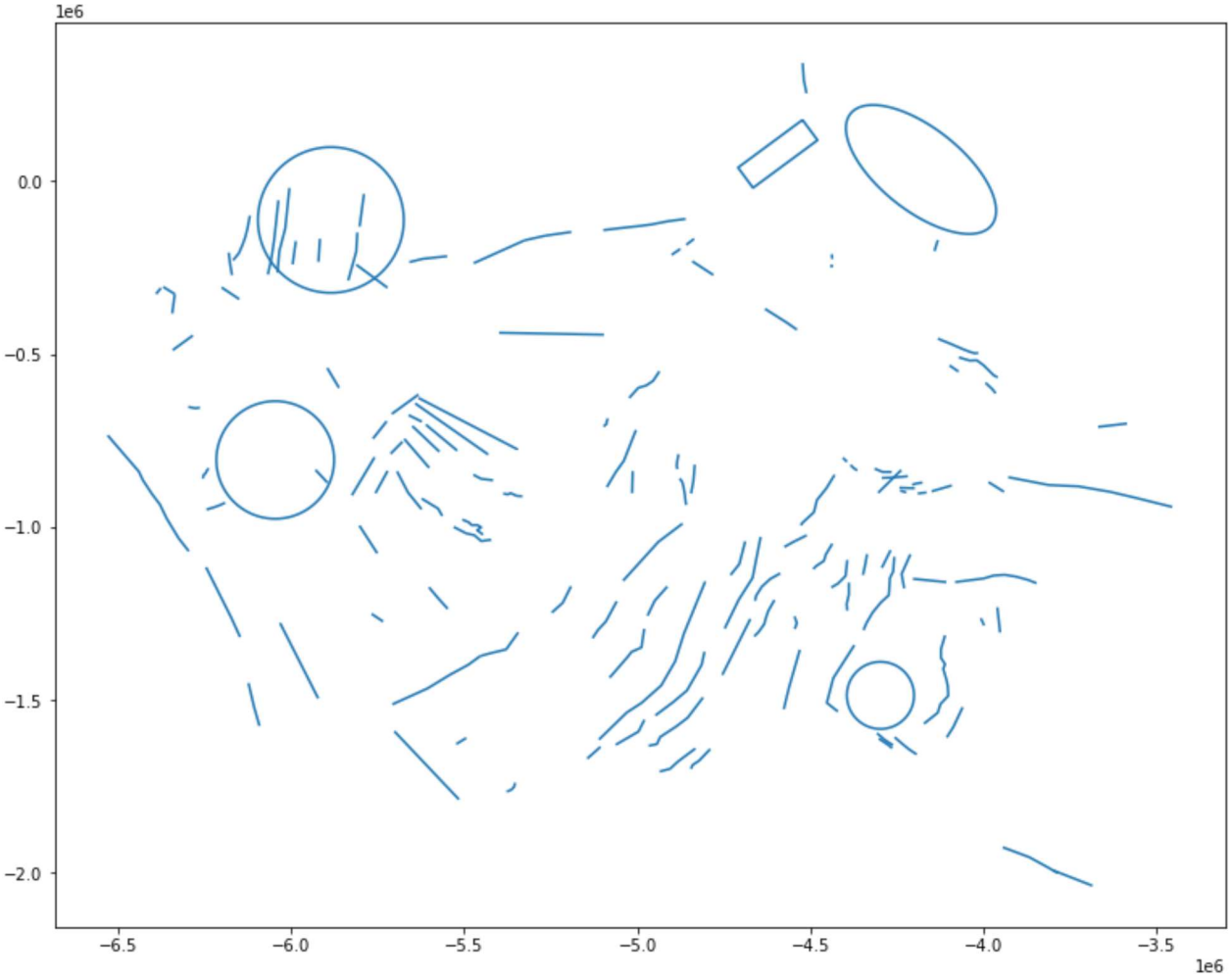
```
from matplotlib.colors import BoundaryNorm, LinearSegmentedColormap #Maps the raster la


custom_cmap = LinearSegmentedColormap.from_list('mars', ['#162252', #Creates a colored
                                                         '#104E8B',
                                                         '#00B2EE',
                                                         '#00FF00',
                                                         '#FFFF00',
                                                         '#FFA500',
                                                         '#FF0000',
                                                         '#8b0000',
                                                         '#964B00',
                                                         '#808080',
                                                         '#FFFFFF'], N=2221)

bounds = np.arange(-8210, 14000, 10) #Sets boundaries for raster
norm = BoundaryNorm(bounds, custom_cmap.N)

fig, ax = plt.subplots() #Makes subplots of area
fig.set_size_inches(14, 7)

ax.imshow(mars[0], cmap=custom_cmap) #Shows assigned color map of Mars
ax.axis('off')
newax = fig.add_axes([0.82, 0.13, 0.08, 0.08], anchor='NE')
newax.axis('off')
plt.show()
```



In [5]:

```
from matplotlib.colors import BoundaryNorm, LinearSegmentedColormap



custom_cmap = LinearSegmentedColormap.from_list('watershed', ['#162252', #Shows watersh
                                                              '#104E8B',
                                                              '#00B2EE',
                                                              '#00FF00',
                                                              '#FFFF00',
                                                              '#FFA500',
                                                              '#FF0000',
                                                              '#8b0000',
                                                              '#964B00',
                                                              '#808080',
                                                              '#FFFFFF'], N=2221)

bounds = np.arange(-8210, 14000, 10)
norm = BoundaryNorm(bounds, custom_cmap.N)

fig, ax = plt.subplots()
fig.set_size_inches(14, 7)

ax.imshow(watershed[0], cmap=custom_cmap)
ax.axis('off')
newax = fig.add_axes([0.82, 0.13, 0.08, 0.08], anchor='NE')
newax.axis('off')
gdfligament.plot(figsize = (13, 15)) #Creates plot of geological linear feature for ide
plt.show()
```

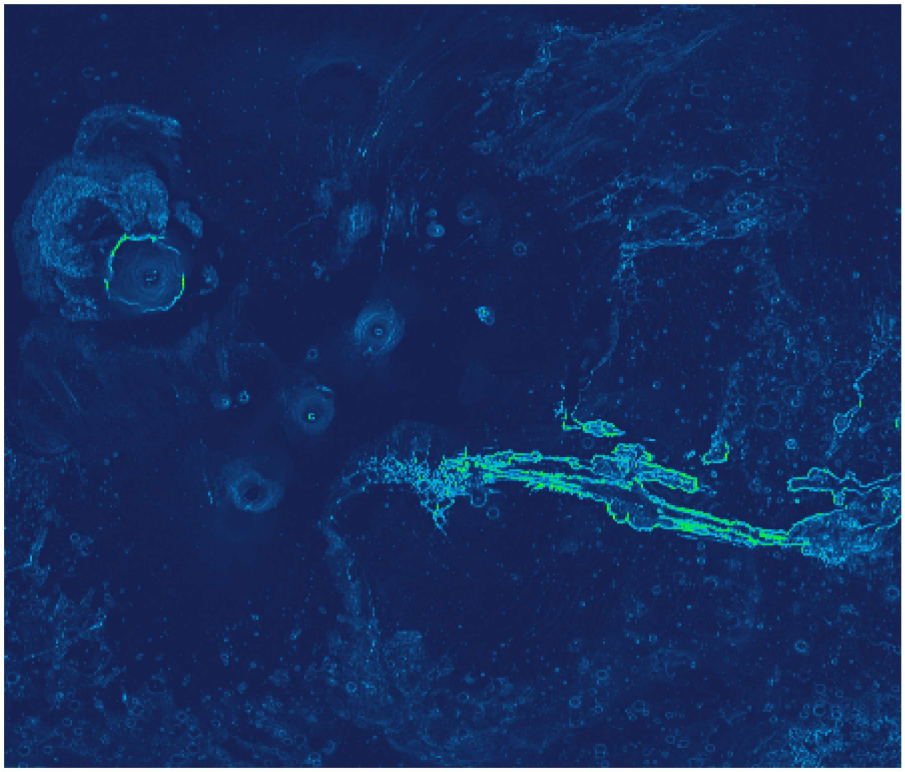


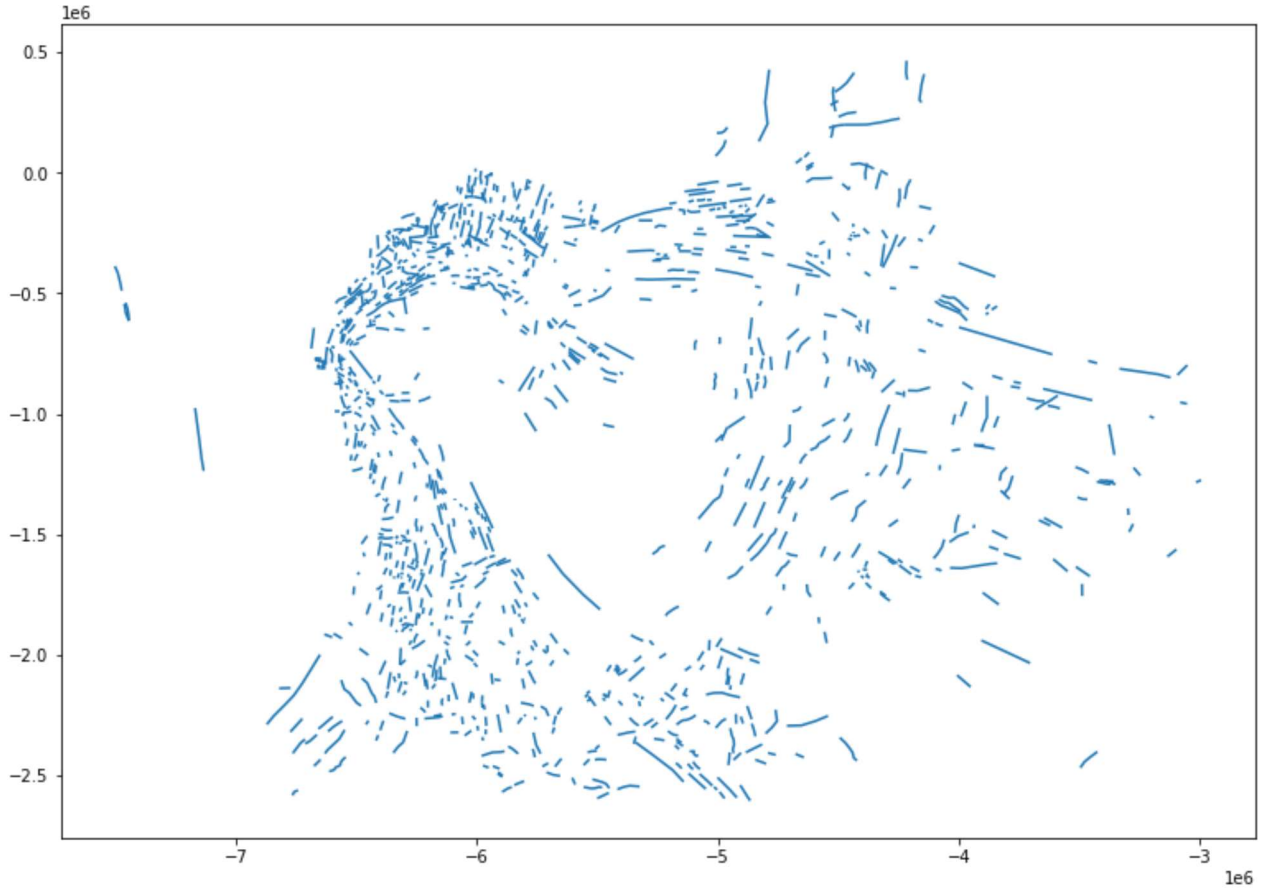


In [6]:

```
from matplotlib.colors import BoundaryNorm, LinearSegmentedColormap


custom_cmap = LinearSegmentedColormap.from_list('slopemars', ['#162252',
                                                              '#104E8B',
                                                              '#00B2EE',
                                                              '#00FF00',
```

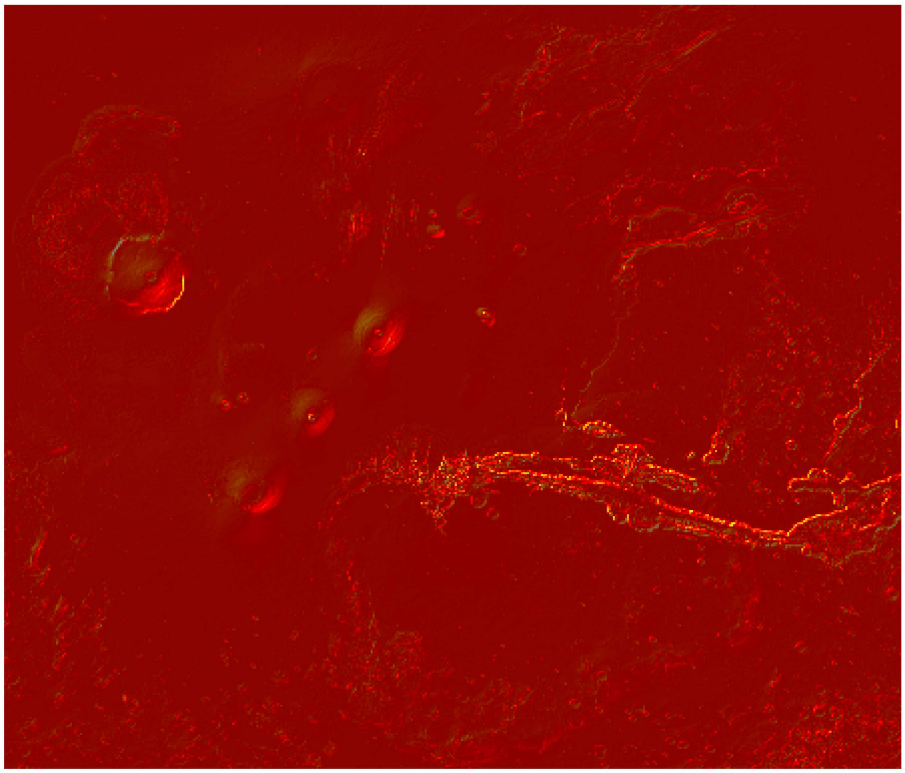
```
                                                    '#FFFF00',
                                                    '#FFA500',
                                                    '#FF0000',
                                                    '#8b0000',
                                                    '#964B00',
                                                    '#808080',
                                                    '#FFFFFF'], N=2221)

bounds = np.arange(-8210, 14000, 10)
norm = BoundaryNorm(bounds, custom_cmap.N)

fig, ax = plt.subplots()
fig.set_size_inches(14, 7)

ax.imshow(slopemars[0], cmap=custom_cmap)
ax.axis('off')
newax = fig.add_axes([0.82, 0.13, 0.08, 0.08], anchor='NE')
newax.axis('off')
gdfslope.plot(figsize = (13, 15)) #Displays slope geological linear features along slop
plt.show()
```

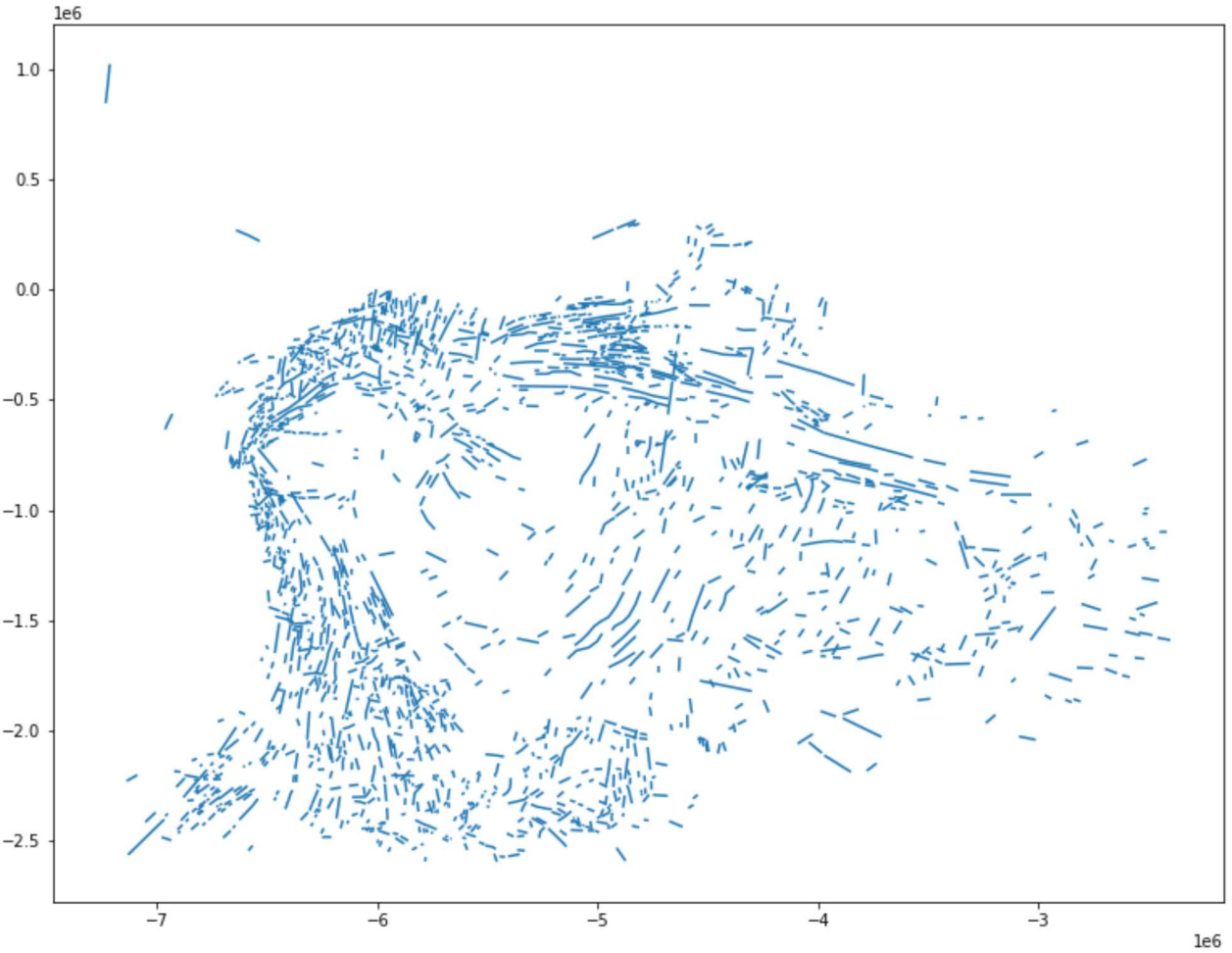




In [7]:

```
from matplotlib.colors import BoundaryNorm, LinearSegmentedColormap


custom_cmap = LinearSegmentedColormap.from_list('hillshade', ['#162252',
                                                              '#104E8B',
                                                              '#00B2EE',
                                                              '#00FF00',
                                                              '#FFFF00',
                                                              '#FFA500',
                                                              '#FF0000',
                                                              '#8b0000',
                                                              '#964B00',
                                                              '#808080',
                                                              '#FFFFFF'], N=2221)

bounds = np.arange(-8210, 14000, 10)
norm = BoundaryNorm(bounds, custom_cmap.N)

fig, ax = plt.subplots()
fig.set_size_inches(14, 7)

ax.imshow(hillshade[0], cmap=custom_cmap)
ax.axis('off')
newax = fig.add_axes([0.82, 0.13, 0.08, 0.08], anchor='NE')
newax.axis('off')
gdfhillshade.plot(figsize = (13, 15)) #Displays hillshade geological linear features al
plt.show()
```





In [8]:
```
gdf['geometry'] #Displays feature lengths
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_320/2308936042.py in <module>
----> 1 gdf['geometry'] #Displays feature lengths

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

In [ ]:
```
gdf.dtypes #Data type of ligaments
```

In [ ]:
```
gdf[2:5]
```

In [9]:
```
colors_undersea = plt.cm.ocean(np.linspace(0.2, 0.8, 821)) #Assigns variables to be plo
undersea_map = LinearSegmentedColormap.from_list('mars', colors_undersea, N=821)

colors_land = plt.cm.terrain(np.linspace(0.25, 1, 1400))
land_map = LinearSegmentedColormap.from_list('watershed', colors_land, N=1400)

colors = np.vstack((colors_undersea, colors_land))
terrain_map = LinearSegmentedColormap.from_list('hillshade', colors, N=2221)

bounds = np.arange(-8210, 14000, 10)
norm = BoundaryNorm(bounds, terrain_map.N)
plt.show()
```

In [11]:
```
def change_sea_level(new_sea_level): #Shows a colormap for Mars to delienate the differ
    new_sea_level = new_sea_level / 10
    undersea = int(800 + new_sea_level)
    land = int(1400 - new_sea_level)
    colors_undersea = plt.cm.ocean(np.linspace(0.2, 0.8, undersea)) #Based on ocean dep
    undersea_map = LinearSegmentedColormap.from_list('watershed', colors_undersea, N=un

    colors_land = plt.cm.terrain(np.linspace(0.25, 1, land)) #Based on land surface
    land_map = LinearSegmentedColormap.from_list('mars', colors_land, N=land)

    colors = np.vstack((colors_undersea, colors_land)) #Based on terrian/angles
    terrain_map = LinearSegmentedColormap.from_list('slopemars', colors, N=2221)

    bounds = np.arange(-8210, 14000, 10)
    norm = BoundaryNorm(bounds, terrain_map.N)
    return terrain_map, norm


terrain_map0, norm0 = change_sea_level(new_sea_level = 0)
terrain_map3000, norm3000 = change_sea_level(new_sea_level = 3000)
terrain_map_3000, norm_3000 = change_sea_level(new_sea_level = -3000)
plt.show()
```

In [ ]:
```
terrain_map1500, norm1500 = change_sea_level(new_sea_level = 1500) #Plotting Mars Terra

fig, ax = plt.subplots()
fig.set_size_inches(14, 7)

ax.imshow(mars[0], cmap=terrain_map1500, norm=norm1500)
ax.axis('off')
```

```
newax = fig.add_axes([0.82, 0.13, 0.08, 0.08], anchor='NE')
newax.axis('off')

plt.show()
```

In [ ]:

```
pip install earthpy #Attempt to use a different Python package by clipping the vector f
```

In [ ]:

```
pip install rioxarray #Attempt to use a different Python package clip the vector to the
```

In [ ]:

```
# Inspiration: https://www.tensorflow.org/tutorials/generative/dcgan
# https://www.kaggle.com/code/joxcat/simple-cat-generator
# Data source: http://madm.dfki.de/downloads
# The link above provided us with EuroSAT data
# It is a dataset containing 27000 satellite images with different classes

# Import modules below
# Uses TensorFlow GAN to generate cat pictures with a goal of
# Currently working on changing the dataset to satellite imagery to produce
# our own.

import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers
import time
from IPython import display

# Set our image size
img_size = 64

# We are using the mnist dataset from TF datasets as a template
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
print(train_images.shape, train_labels.shape)

# Here, I am defining the distribution strategy in an attempt to use GPUs
strategy = tf.distribute.MirroredStrategy()

# Print out the number of devices we are working with, Tim's PC has 1 GPU
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

# Reading in our cat dataset from our own hardware (Tim's PC)
# Below, I was trying out multiple datasets to see how to reshape the data
# I have learned that the parameters have to create the product of the array
# Once I figured that out, I learned that the model doesn't accept the shape
# So now, we have to figure out how to change the model to accept the shape
cat_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    #"../cat_gan/cats",
    #"../cat_gan/eurosat",
    "../cat_gan/test",
    labels="inferred",
    label_mode="int",
    class_names=['Residential'], # folder named cats in the path
    #class_names=['AnnualCrop','Forest','HerbaceousVegetation','Highway','Industrial','
    #             'PermanentCrop','Residential','River','SeaLake'],
    #color_mode="grayscale", # output color
    color_mode="rgb", # output color
```

```
    batch_size=32,
    image_size=(img_size, img_size),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
)

cat_train_labels = []
cat_train_images = []

for images, labels in cat_dataset:
    for i in range(len(images)):
      cat_train_images.append(images[i])
      cat_train_labels.append(labels[i])


# Formatting our dataset into a numpy array
c_images = np.array(cat_train_images)
print(c_images.shape[0])
c_images = c_images.reshape(c_images.shape[0],img_size,img_size)

# This was to reformat the new satellite imagery dataset without erroring out
# Not 100% sure if that's correct, or what we want to do, but it seems like
# the right direction.
#c_images = c_images.reshape(c_images.shape[0],192,img_size)
print(c_images.shape)


c_labels = np.array(cat_train_labels)
print(c_labels.shape)
c_labels = c_labels.reshape(c_labels.shape[0])
print(c_labels.shape)

# Saves our model when training.
from numpy import save
save('images.npy', c_images)
save('labels.npy', c_labels)


train_images = c_images
train_labels = c_labels

train_images = train_images.reshape(train_images.shape[0], img_size, img_size, 3).astyp
train_images = (train_images - 127.5) / 127.5 # Normalize the images to [-1, 1]

# More parameters to set, buffer size and batch size
BUFFER_SIZE = 40000
BATCH_SIZE = 256

# Define the train dataset model here by slicing the train images and shuffling them
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).b

# Generator model from TensorFlow GAN tutorial that we are using
# Added strategy.scope() as parallel code
# Here is where we are running into problems
# For instance, we are getting:
# ValueError: Input 0 of layer "sequential_8" is incompatible with the layer:
```

```
# expected shape=(None, 56, 56, 1), found shape=(256, 64, 64, 3)
# We are not sure what parameters to change within the model to correct it
with strategy.scope():
    def make_generator_model():

        model = tf.keras.Sequential()
        model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
        model.add(layers.BatchNormalization())
        model.add(layers.LeakyReLU())

        model.add(layers.Reshape((7, 7, 256)))
        assert model.output_shape == (None, 7, 7, 256) # Note: None is the batch size

        model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', u
        assert model.output_shape == (None, 7, 7, 128)
        model.add(layers.BatchNormalization())
        model.add(layers.LeakyReLU())

        model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', us
        assert model.output_shape == (None, 14, 14, 64)
        model.add(layers.BatchNormalization())
        model.add(layers.LeakyReLU())

        model.add(layers.Conv2DTranspose(1, (10, 10), strides=(4, 4), padding='same', u
        print(model.output_shape)
        assert model.output_shape == (None, 56, 56, 1) #generator model

        return model

# Setting the model into a variable so it will be used throughout the code
generator = make_generator_model()

# This shows us what the model is creating everytime we run it
# Makes the noise random so it shows us what its creating
noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

# The code below just shows us what the generated image looks like, but we
# don't need it in our case.
#plt.imshow(generated_image[0, :, :, 0], cmap='gray')

# This is the discriminator model, also from TensorFlow GAN
with strategy.scope():
    def make_discriminator_model():
        model = tf.keras.Sequential()
        model.add(layers.Conv2D(64, (10, 10), strides=(2, 2), padding='same',
                                         input_shape=[56, 56, 1]))
        model.add(layers.LeakyReLU())
        model.add(layers.Dropout(0.3))

        model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
        model.add(layers.LeakyReLU())
        model.add(layers.Dropout(0.3))

        model.add(layers.Flatten())
        model.add(layers.Dense(1))

        return model

# Creating the discriminator model here so it can be used later
```

```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)


generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)


# Parameters to define, we can change how many epochs, examples to create, etc
EPOCHS = 1
noise_dim = 100
num_examples_to_generate = 12

# Creates the random seed to generate
seed = tf.random.normal([num_examples_to_generate, noise_dim])


@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminat


# The actual function that takes out dataset and epochs to train our model
def train(dataset, epochs):
    for epoch in range(epochs):
        start = time.time()
    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as we go
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
```

```
                                seed)

    # Save the model every 15 epochs
    #if (epoch + 1) % 15 == 0:
        #checkpoint.save(file_prefix = checkpoint_prefix)
    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

# Generate after the final epoch
    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epochs,
                             seed)


# A function created to save the images that the model outputs into the root
# directory of where the program is stored.

def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)
  fig = plt.figure(figsize=(4,4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()

# RUN THE SCRIPT
train(train_dataset, EPOCHS)
```

> The comments (#) denote where keycode/detail was implemented. A lot of the same interactions of code have been repeated, this it for simple visualization proposes as this isn't meant to be a complex operation. Only to show the juxtaposition of geological interactions on the surface and to show how the ligament features compare in conforming to shaping the fluvial valleys on the planet. As stated in the synopsis, most work was rendered in ArcGIS and the visualization and colormaps give inklings into how the terrain on Mars is geologically congruent to that of Earth and that they share a similar geological history.