```
import requests
import zipfile
import os
import arcpy
from arcpy.sa import Reclassify, Con, RemapValue, Raster, CostPath
import glob

arcpy.CheckOutExtension("Spatial")
aprx = arcpy.mp.ArcGISProject("CURRENT")
Dory_folder = r"C:\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory"
os.makedirs(Dory_folder, exist_ok=True)
arcpy.env.workspace = Dory_folder

def download_and_extract(url, zip_path, extract_to):
    try:
        response = requests.get(url)
        with open(zip_path, 'wb') as file:
            file.write(response.content)
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            zip_ref.extractall(extract_to)
        print(f"Downloaded and extracted to {extract_to}")
    except Exception as e:
        print(f"Failed to download or extract {url}: {e}")

def create_feature_class(folder, name, spatial_ref):
    path = os.path.join(folder, name)
    if not arcpy.Exists(path):
        arcpy.management.CreateFeatureclass(folder, name, "POINT",
spatial_reference=spatial_ref)
        print(f"Feature class {name} created.")
    else:
        print(f"Feature class {name} already exists.")
    return path



# Set Up Map  - Dory Farm and Picnic Area points

if not any(m.name == "Dory_Map" for m in aprx.listMaps()):
    dory_map = aprx.createMap("Dory_Map", "Map")
else:
    dory_map = aprx.listMaps("Dory_Map")[0]

# Set start and end points on the map (Dory's Farm and Picnic Area)
spatial_ref = arcpy.SpatialReference(4326)

# Dory's Farm
dory_farm_start_lat, dory_farm_start_long = 44.127985, -92.148796
dory_farm_start = create_feature_class(Dory_folder,
"Dory_farm_start.shp", spatial_ref)
```

```
# Add point to Dory's Farm feature class
if not arcpy.management.AddField(dory_farm_start, "Name", "TEXT",
field_length=50):
    print("Failed to add 'Name' field to Dory_farm_start")

with arcpy.da.InsertCursor(dory_farm_start, ['SHAPE@', 'Name']) as
cursor:
    cursor.insertRow([arcpy.Point(dory_farm_start_long,
dory_farm_start_lat), "Dory's Farm"])

dory_map.addDataFromPath(dory_farm_start)

# North Picnic Area
picnic_lat, picnic_long = 44.0547076, -92.0462307
picnic_area = create_feature_class(Dory_folder,
"North_Picnic_end.shp", spatial_ref)

# Add point to North Picnic feature class
if not arcpy.management.AddField(picnic_area, "Name", "TEXT",
field_length=50):
    print("Failed to add 'Name' field to North_Picnic_end")

with arcpy.da.InsertCursor(picnic_area, ['SHAPE@', 'Name']) as cursor:
    cursor.insertRow([arcpy.Point(picnic_long, picnic_lat), "North
Picnic"])

dory_map.addDataFromPath(picnic_area)


# County Boundary Data - for clipping
county_url = "https://resources.gisdata.mn.gov/pub/gdrs/data/pub/
us_mn_state_dot/bdry_counties/shp_bdry_counties.zip"
county_zip_path = os.path.join(Dory_folder, "mn_county.zip")
county_extract_dir = os.path.join(Dory_folder, "mn_county_extracted")
download_and_extract(county_url, county_zip_path, county_extract_dir)

county_shapefile_path = os.path.join(county_extract_dir,
"County_Boundaries_in_Minnesota.shp")


target_counties = ['(Winona', 'Wabasha', 'Olmsted)']
county_target_shapefile = os.path.join(Dory_folder,
"MN_County_Targets.shp")

if not arcpy.Exists(county_target_shapefile):
    arcpy.management.MakeFeatureLayer(county_shapefile_path,
"counties_lyr")
    where_clause = f"COUNTY_NAM IN {tuple(target_counties)}"
    arcpy.management.SelectLayerByAttribute("counties_lyr",
```

```
"NEW_SELECTION", where_clause)
    arcpy.management.CopyFeatures("counties_lyr",
county_target_shapefile)
    print(f"Saved selected counties to {county_target_shapefile}")
else:
    print(f"Target county shapefile {county_target_shapefile} already
exists.")

dory_map.addDataFromPath(county_target_shapefile)

# ------------------------

# Land Use
# She also doesn't like crossing water bodies if there isn't a bridge,
though sometimes she doesn't mind if she's wearing her waders.

land_url = "https://resources.gisdata.mn.gov/pub/gdrs/data/pub/
us_mn_state_dnr/biota_landcover_nlcd_mn_2016/
tif_biota_landcover_nlcd_mn_2016.zip"
land_zip_path = os.path.join(Dory_folder, "landcover.zip")
land_extract_dir = os.path.join(Dory_folder, "land_use_extracted")
download_and_extract(land_url, land_zip_path, land_extract_dir)

land_raster_path = os.path.join(land_extract_dir,
"NLCD_2016_Land_Cover.tif")

# Clip land use
land_use_clipped = os.path.join(Dory_folder,
"NLCD_2016_Land_Cover_Clip.tif")
if not arcpy.Exists(land_use_clipped):
    arcpy.management.Clip(
        in_raster=land_raster_path,
        out_raster=land_use_clipped,
        in_template_dataset=county_target_shapefile,
        clipping_geometry="ClippingGeometry",
        nodata_value="255",
        maintain_clipping_extent="NO_MAINTAIN_EXTENT"
    )
print(f"Clipped land cover raster saved to {land_use_clipped}")

# Reclassify muddy farm fields
# per metadata, 81, 82 are codes for these land uses
farm_fields_reclass_path = os.path.join(Dory_folder,
"Farm_Reclassified.tif")
if not arcpy.Exists(farm_fields_reclass_path):
    remap = RemapValue([
        [81, 10],  # Pasture
        [82, 10],  # Crops
    ])
    farm_fields_reclass = Reclassify(land_use_clipped, "Value", remap,
```

```
"NODATA")
    farm_fields_reclass.save(farm_fields_reclass_path)
    print(f"Farm fields reclassified and saved to
{farm_fields_reclass_path}")

dory_map.addDataFromPath(os.path.abspath(farm_fields_reclass_path))

# Hydrology Analysis
# Extract hydrology data for determining accessible water bodies and
bridge paths for Dory's navigation.

import os
import arcpy
from arcpy.sa import Reclassify, Con, RemapValue, Raster

# Define directories and paths
Dory_folder = r"C:\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory"
os.makedirs(Dory_folder, exist_ok=True)

# URLs for external data sources
hydro_url = "https://resources.gisdata.mn.gov/pub/gdrs/data/pub/
us_mn_state_dnr/water_dnr_hydrography/shp_water_dnr_hydrography.zip"
bridge_url = "https://resources.gisdata.mn.gov/pub/gdrs/data/pub/
us_mn_state_dot/trans_bridges/shp_trans_bridges.zip"

# Paths for downloads
hydro_zip_path = os.path.join(Dory_folder, "hydrography.zip")
hydro_extract_dir = os.path.join(Dory_folder, "hydrography_extracted")
bridge_zip_path = os.path.join(Dory_folder, "bridges.zip")
bridge_extract_dir = os.path.join(Dory_folder, "bridges_extracted")

# Download and extract hydrology and bridge data
download_and_extract(hydro_url, hydro_zip_path, hydro_extract_dir)
download_and_extract(bridge_url, bridge_zip_path, bridge_extract_dir)

hydro_shapefile_path = os.path.join(hydro_extract_dir,
"dnr_hydro_features_all.shp")
bridge_shapefile_path = os.path.join(bridge_extract_dir,
"Bridge_locations_in_Minnesota.shp")

# Paths for processed files
hydro_clipped_path = os.path.join(Dory_folder, "Hydro_Clipped.shp")
bridge_clipped_path = os.path.join(Dory_folder, "Bridges_Clipped.shp")
water_raster_path = os.path.join(Dory_folder, "Water_Raster.tif")
water_reclass_path = os.path.join(Dory_folder,
"Water_Reclassified.tif")
bridge_raster_path = os.path.join(Dory_folder, "Bridges_Raster.tif")
adjusted_water_reclass_path = os.path.join(Dory_folder,
"Adjusted_Water_Reclassified.tif")
```

```
# Clip hydrology and bridge data to the target area
(county_target_shapefile)
if not arcpy.Exists(hydro_clipped_path):
    arcpy.analysis.Clip(hydro_shapefile_path, county_target_shapefile,
hydro_clipped_path)
    print(f"Hydrography data clipped and saved to
{hydro_clipped_path}")

if not arcpy.Exists(bridge_clipped_path):
    arcpy.analysis.Clip(bridge_shapefile_path,
county_target_shapefile, bridge_clipped_path)
    print(f"Bridge data clipped and saved to {bridge_clipped_path}")

# Add "wb_code" field to hydrology data, if it doesn't exist
wb_code_field = "wb_code"
if wb_code_field not in [f.name for f in
arcpy.ListFields(hydro_clipped_path)]:
    arcpy.management.AddField(hydro_clipped_path, wb_code_field,
"SHORT")

# Water body class to code per metadata
wb_class_mapping = {
    'Artificial Basin': 1, 'Drained Lakebed': 2, 'Drained Wetland': 3,
    'Fish Hatchery Pond': 4, 'Industrial Waste Pond': 5, 'Inundation
Area': 6,
    'Intermittent Water': 7, 'Island or Land': 8, 'Lake or Pond': 9,
    'Mine or Gravel Pit': 10, 'Natural Ore Mine': 11, 'Natural
Taconite/Ore Mine': 12,
    'Partially Drained Lakebed': 13, 'Partially Drained Wetland': 14,
'Reservoir': 15,
    'Riverine island': 16, 'Riverine polygon': 17, 'Sewage/Filtration
Pd': 18,
    'Tailings Pond': 19, 'Wetland': 20
}

# Update 'wb_code' field in hydrology data based on mapping
with arcpy.da.UpdateCursor(hydro_clipped_path, ['wb_class',
wb_code_field]) as cursor:
    for row in cursor:
        row[1] = wb_class_mapping.get(row[0], 99)  # Use 99 for
unknown types
        cursor.updateRow(row)

# Convert hydrology data to raster using wb_code as value
if not arcpy.Exists(water_raster_path):
    cell_size = arcpy.management.GetRasterProperties(land_use_clipped,
"CELLSIZEX").getOutput(0)
    arcpy.conversion.PolygonToRaster(
        in_features=hydro_clipped_path, value_field=wb_code_field,
        out_rasterdataset=water_raster_path,
```

```
cell_assignment="MAXIMUM_COMBINED_AREA",
        cellsize=cell_size
    )
    print(f"Water raster created at {water_raster_path}")

# Reclassify water bodies - likeliness to be impassable
if not arcpy.Exists(water_reclass_path):
    remap = RemapValue([
        [1, 8], [2, 2], [3, 2], [4, 10], [5, 15],
        [6, 6], [7, 5], [8, 1], [9, 12], [10, 10],
        [11, 10], [12, 10], [13, 3], [14, 3], [15, 10],
        [16, 1], [17, 8], [18, 15], [19, 12], [20, 7],
        [99, 8]
    ])
    water_reclass = Reclassify(water_raster_path, "Value", remap)
    water_reclass.save(water_reclass_path)
    print(f"Reclassified water raster saved at {water_reclass_path}")

# Convert bridge data to raster and adjust water cost where bridges
exist
if not arcpy.Exists(bridge_raster_path):
    arcpy.conversion.FeatureToRaster(
        in_features=bridge_clipped_path, field="FID",
        out_raster=bridge_raster_path, cell_size=cell_size
    )
    print(f"Bridge raster created at {bridge_raster_path}")

# Set cost to 1 where bridges exist
if not arcpy.Exists(adjusted_water_reclass_path):
    adjusted_water_raster = Con(Raster(bridge_raster_path) > 0, 1,
Raster(water_reclass_path))
    adjusted_water_raster.save(adjusted_water_reclass_path)
    print(f"Adjusted water reclassification saved at
{adjusted_water_reclass_path}")

# Add final raster layers to map
dory_map.addDataFromPath(adjusted_water_reclass_path)

# ---------------

# Slope / Elevation
# Dory wants to take the path that is the most gradual in terms of
slope.

# Access DEM
slope_url = "https://resources.gisdata.mn.gov/pub/gdrs/data/pub/
us_mn_state_dnr/elev_30m_digital_elevation_model/
fgdb_elev_30m_digital_elevation_model.zip"
slope_zip_path = os.path.join(Dory_folder, "elevation.zip")
slope_extract_dir = os.path.join(Dory_folder, "elevation_extracted")
```

```
download_and_extract(slope_url, slope_zip_path, slope_extract_dir)

# Set paths to DEM data
dem_gdb_path = os.path.join(slope_extract_dir,
"elev_30m_digital_elevation_model.gdb")
dem_dataset_name = "digital_elevation_model_30m"
dem_path = os.path.join(dem_gdb_path, dem_dataset_name)

# Add DEM to the map
dory_map.addDataFromPath(dem_path)

# Clip DEM to target area
dem_clipped_path = os.path.join(Dory_folder, "DEM_Clipped.tif")
if not arcpy.Exists(dem_clipped_path):
    arcpy.management.Clip(
        in_raster=dem_path,
        out_raster=dem_clipped_path,
        in_template_dataset=county_target_shapefile,
        nodata_value="-9999",
        clipping_geometry="ClippingGeometry",
        maintain_clipping_extent="NO_MAINTAIN_EXTENT"
    )
    print(f"Clipped DEM saved at {dem_clipped_path}")
else:
    print(f"Clipped DEM already exists at {dem_clipped_path}")

# Add clipped DEM to the map
dory_map.addDataFromPath(dem_clipped_path)

# Calculate slope from the clipped DEM
slope_raster_path = os.path.join(Dory_folder, "Slope.tif")
if not arcpy.Exists(slope_raster_path):
    slope_raster = arcpy.sa.Slope(
        in_raster=dem_clipped_path,
        output_measurement="DEGREE",
        z_unit="METER",
        method="PLANAR"
    )
    slope_raster.save(slope_raster_path)
    print(f"Slope raster saved at {slope_raster_path}")
else:
    print(f"Slope raster already exists at {slope_raster_path}")

dory_map.addDataFromPath(slope_raster_path)

# Reclassify slope values based on degrees
# Slope classifications:
# 0-5 degrees = 1 (low)
# 5-15 degrees = 3 (medium)
# 15-30 degrees = 5 (high)
```

```
# 30-90 degrees = 10 (very high / steep)

slope_reclass_path = os.path.join(Dory_folder,
"Slope_Reclassified.tif")

# Reclassify slope based on defined values, see above
if not arcpy.Exists(slope_reclass_path):
    slope_raster = Raster(slope_raster_path)
    reclassified_slope = Con((slope_raster >= 0) & (slope_raster < 5),
1,
                             Con((slope_raster >= 5) & (slope_raster <
15), 3,
                                 Con((slope_raster >= 15) &
(slope_raster < 30), 5,
                                     Con((slope_raster >= 30) &
(slope_raster <= 90), 10))))
    reclassified_slope.save(slope_reclass_path)
    print(f"Reclassified slope raster saved at {slope_reclass_path}")
else:
    print(f"Reclassified slope raster already exists at
{slope_reclass_path}")

# Add reclassified slope raster to the map
dory_map.addDataFromPath(slope_reclass_path)

from arcpy.sa import *

# Raster objects
slope_reclass_raster = Raster(r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Slope_Reclassified.tif"
)
farm_fields_reclass_raster = Raster(r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Farm_Reclassified.tif")
adjusted_water_reclass_raster = Raster(r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Adjusted_Water_Reclassi
fied.tif")


cost_surface_weighted_path = r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Dory_Cost_Surface_Weigh
ted.tif"

# Apply weights and save
if not arcpy.Exists(cost_surface_weighted_path):
    cost_surface_weighted = (0.5 * slope_reclass_raster +
                             0.3 * farm_fields_reclass_raster +
                             0.2 * adjusted_water_reclass_raster)
    cost_surface_weighted.save(cost_surface_weighted_path)
    print(f"Saved to {cost_surface_weighted_path}")
else:
```

```
    print(f"Already exists at {cost_surface_weighted_path}")

# Add the weighted cost surface to the map
dory_map.addDataFromPath(cost_surface_weighted_path)

#Cost Distance


dory_farm_start = os.path.join(Dory_folder, "Dory_Farm_start.shp")
north_picnic_end = os.path.join(Dory_folder, "North_Picnic_end.shp")
cost_surface_weighted_path = os.path.join(Dory_folder,
"Dory_Cost_Surface_Weighted.tif")

# Check spatial references
cost_surface_sr =
arcpy.Describe(cost_surface_weighted_path).spatialReference
dory_farm_start_sr = arcpy.Describe(dory_farm_start).spatialReference
north_picnic_end_sr =
arcpy.Describe(north_picnic_end).spatialReference

cost_surface_sr

dory_farm_start_sr

north_picnic_end_sr

dory_farm_start_projected = os.path.join(Dory_folder,
"Dory_Farm_start_Projected.shp")

# Check if projection is required
if dory_farm_start_projected == os.path.join(Dory_folder,
"Dory_Farm_start_Projected.shp"):
    if not arcpy.Exists(dory_farm_start_projected):
        arcpy.management.Project(
            in_dataset=dory_farm_start,
            out_dataset=dory_farm_start_projected,
            out_coor_system=cost_surface_sr
        )
        print(f"Dory Farm projected to {dory_farm_start_projected}")
    dory_farm_start = dory_farm_start_projected
else:
    print("Dory Farm projection path confirmed.")


Dory_Farm_start_Projected = "C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Dory_Farm_start_Project
ed.shp"

# Project North Picnic
north_picnic_end_projected = os.path.join(Dory_folder,
```

```
"north_picnic_end_Projected.shp")

# Check if projection is required
if north_picnic_end_projected == os.path.join(Dory_folder,
"north_picnic_end_Projected.shp"):
    if not arcpy.Exists(north_picnic_end_projected):
        arcpy.management.Project(
            in_dataset=north_picnic_end,
            out_dataset=north_picnic_end_projected,
            out_coor_system=cost_surface_sr
        )
        print(f"north_picnic projected to
{north_picnic_end_projected}")
    north_picnic_end = north_picnic_end_projected
else:
    print("north_picnic projection path confirmed.")

north_picnic_end_projected  = "C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\north_picnic_end_Projec
ted.shp"

# Verify spatial references
print("Cost Surface:", cost_surface_sr.name)
print("Dory Farm:",
arcpy.Describe(dory_farm_start).spatialReference.name)
print("North Picnic:",
arcpy.Describe(north_picnic_end).spatialReference.name)

# output rasters
cost_distance_raster_path = os.path.join(Dory_folder,
"Cost_Distance.tif")
backlink_raster_path = os.path.join(Dory_folder, "Backlink.tif")


# Calculate Cost Distance

if not arcpy.Exists(cost_distance_raster_path):
    out_cost_distance = arcpy.sa.CostDistance(
        in_source_data=Dory_Farm_start_Projected,
        in_cost_raster=cost_surface_weighted_path,
        out_backlink_raster=backlink_raster_path
    )
    out_cost_distance.save(cost_distance_raster_path)
    print(f"Cost Distance saved at {cost_distance_raster_path}")
else:
    print("already exists.")


cost_distance_raster_path = r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Cost_Distance.tif"
```

```
backlink_raster_path = r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\Backlink.tif"
least_cost_path_raster_path = os.path.join(Dory_folder,
"Least_Cost_Path.tif")

# Check and calculate statistics for each raster
for raster_path in [cost_distance_raster_path, backlink_raster_path]:
    if arcpy.Exists(raster_path):
        try:
            arcpy.management.CalculateStatistics(raster_path)
            print(f"Calculated statistics for {raster_path}")
        except arcpy.ExecuteError as e:
            print(f"Failed to calculate statistics for {raster_path}:
{e}")
    else:
        print(f"Raster does not exist: {raster_path}")

#Uncertainty analysis

#Map Algebra ... weight equal 1

weights = [0.2, 0.4, 0.6, 0.8]
for slope_weight in weights:
    for farm_weight in weights:
        for water_weight in weights:
            total_weight = slope_weight + farm_weight + water_weight
            if total_weight == 1.0:
                cost_surface_varied = (slope_weight *
slope_reclass_raster +
                                       farm_weight *
farm_fields_reclass_raster +
                                       water_weight *
adjusted_water_reclass_raster)
                output_path = os.path.join(
                    Dory_folder,
                    f"Cost_Surface_slope_{slope_weight}
_farm_{farm_weight}_water_{water_weight}.tif"
                )
                cost_surface_varied.save(output_path)
                print(f"Saved varied cost surface at {output_path}")


# Loop through the saved cost surfaces


import glob
import arcpy
from arcpy.sa import CostDistance, CostPath

north_picnic_end_projected = r"C:
```

```
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\north_picnic_end_Projec
ted.shp"
cost_surface_files = glob.glob(os.path.join(Dory_folder,
"Cost_Surface_slope_*.tif")) #6 rasters from above

for cost_surface_path in cost_surface_files:
    # Define paths for outputs
    cost_distance_raster_path =
cost_surface_path.replace("Cost_Surface", "Cost_Distance")
    backlink_raster_path = cost_surface_path.replace("Cost_Surface",
"Backlink")
    least_cost_path_raster_path =
cost_surface_path.replace("Cost_Surface", "Least_Cost_Path")

    # Compute cost distance
    if not arcpy.Exists(cost_distance_raster_path):
        out_cost_distance = CostDistance(
            in_source_data=Dory_Farm_start_Projected,
            in_cost_raster=cost_surface_path,
            out_backlink_raster=backlink_raster_path
        )
        out_cost_distance.save(cost_distance_raster_path)
        print(f"Cost Distance saved at {cost_distance_raster_path}")

        # Calculate statistics for cost distance and backlink rasters

arcpy.management.CalculateStatistics(cost_distance_raster_path)
        arcpy.management.CalculateStatistics(backlink_raster_path)
        print(f"Statistics calculated for {cost_distance_raster_path}
and {backlink_raster_path}")

        # Verify that statistics are valid
        if arcpy.management.GetRasterProperties(backlink_raster_path,
"ANYNODATA").getOutput(0) == "0":
            print("Statistics verified as valid.")
        else:
            # Retry statistics calculation if they appear incomplete
            arcpy.management.CalculateStatistics(backlink_raster_path)
            time.sleep(2)  # Brief pause to ensure statistics finalize
            print("Recalculated statistics after verification.")

    # Compute least-cost path
    if not arcpy.Exists(least_cost_path_raster_path):
        try:
            out_least_cost_path = CostPath(
                in_destination_data=north_picnic_end_projected,
                in_cost_distance_raster=cost_distance_raster_path,
                in_cost_backlink_raster=backlink_raster_path
            )
            out_least_cost_path.save(least_cost_path_raster_path)
```

```
            print(f"Least-Cost Path saved at
{least_cost_path_raster_path}")
        except arcpy.ExecuteError as e:
            print(f"Error executing CostPath for
{least_cost_path_raster_path}: {e}")

north_picnic_end_projected = r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\north_picnic_end_Projec
ted.shp"

import time

north_picnic_end_projected = r"C:
\Mac\Home\Documents\ArcGIS\Projects\Lab_2\Dory\north_picnic_end_Projec
ted.shp"
cost_surface_files = glob.glob(os.path.join(Dory_folder,
"Cost_Surface_slope_*.tif"))

for cost_surface_path in cost_surface_files:
    # Define paths for outputs
    cost_distance_raster_path =
cost_surface_path.replace("Cost_Surface", "Cost_Distance")
    backlink_raster_path = cost_surface_path.replace("Cost_Surface",
"Backlink")
    least_cost_path_raster_path =
cost_surface_path.replace("Cost_Surface", "Least_Cost_Path")

    # Compute cost distance
    if not arcpy.Exists(cost_distance_raster_path):
        out_cost_distance = CostDistance(
            in_source_data=Dory_Farm_start_Projected,
            in_cost_raster=cost_surface_path,
            out_backlink_raster=backlink_raster_path
        )
        out_cost_distance.save(cost_distance_raster_path)
        print(f"Cost Distance saved at {cost_distance_raster_path}")

        # Calculate and verify statistics for cost distance and
backlink rasters
        for raster_path in [cost_distance_raster_path,
backlink_raster_path]:
            arcpy.management.CalculateStatistics(raster_path)
            time.sleep(2)  # Pause to ensure statistics finalize
            if arcpy.management.GetRasterProperties(raster_path,
"ANYNODATA").getOutput(0) != "0":
                print(f"Recalculating statistics for {raster_path}")
                arcpy.management.CalculateStatistics(raster_path)
                time.sleep(2)

    # Compute least-cost path
```

```
    if not arcpy.Exists(least_cost_path_raster_path):
        try:
            out_least_cost_path = CostPath(
                in_destination_data=north_picnic_end_projected,
                in_cost_distance_raster=cost_distance_raster_path,
                in_cost_backlink_raster=backlink_raster_path
            )
            out_least_cost_path.save(least_cost_path_raster_path)
            print(f"Least-Cost Path saved at
{least_cost_path_raster_path}")
        except arcpy.ExecuteError as e:
            print(f"Error executing CostPath for
{least_cost_path_raster_path}: {e}")
```