```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 vegetation_indices
                                 A QGIS plugin
 This plugin calculates vegetation indices
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/
                              -------------------
        begin                : 2024-01-23
        git sha              : $Format:%H$
        copyright            : (C) 2024 by
        email                :
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
import sys
import os.path
from .QSVI_Plugin_dialog import vegetation_indicesDialog
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication
from qgis.PyQt.QtGui import QIcon
from qgis.PyQt.QtWidgets import QAction, QFileDialog, QCheckBox, QApplication,
QWidget, QPushButton
from .resources import *
from qgis.core import *
from qgis.gui import *
from qgis.utils import iface
from qgis.analysis import QgsRasterCalculatorEntry, QgsRasterCalculator
from qgis.core import QgsRasterLayer, QgsProject

class vegetation_indices:
    """QGIS Plugin Implementation."""
    def __init__(self, iface):
        """Constructor.
        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
```

```python
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'vegetation_indices_{}.qm'.format(locale))
        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            QCoreApplication.installTranslator(self.translator)
        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&vegetation_indices')
        # Check if plugin was started the first time in current QGIS session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None
    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.
        We implement this ourselves since we do not inherit QObject.
        :param message: String for translation.
        :type message: str, QString
        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('vegetation_indices', message)

    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.
        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str
        :param text: Text that should be shown in menu items for this action.
        :type text: str
        :param callback: Function to be called when the action is triggered.
        :type callback: function
        :param enabled_flag: A flag indicating if the action should be enabled
```

```
            by default. Defaults to True.
        :type enabled_flag: bool
        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool
        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool
        :param status_tip: Optional text to show in a popup when mouse pointer
            hovers over the action.
        :type status_tip: str
        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget
        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.
        :returns: The action that was created. Note that the action is also
            added to self.actions list.
        :rtype: QAction
        """
        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)
        if status_tip is not None:
            action.setStatusTip(status_tip)
        if whats_this is not None:
            action.setWhatsThis(whats_this)
        if add_to_toolbar:
            # Adds plugin icon to Plugins toolbar
            self.iface.addToolBarIcon(action)
        if add_to_menu:
            self.iface.addPluginToRasterMenu(
                self.menu,
                action)
        self.actions.append(action)
        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""
        icon_path = ':/plugins/QSVI_Plugin/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'This plugin calculates vegetation indices'),
            callback=self.run,
            parent=self.iface.mainWindow())
        # will be set False in run()
        self.first_start = True

    def update_raster_list(self):
```

```python
        self.clear_lists(
            self.dlg.cmb_red,
            self.dlg.cmb_green,
            self.dlg.cmb_blue,
            self.dlg.cmb_nir,
            self.dlg.cmb_b5,
            self.dlg.cmb_lst,
            self.dlg.cmb_select_index
        )
        layers = list()
        layers.append("Not Set")
        layers = layers + [lay.name() for lay in
self.iface.mapCanvas().layers()]
        self.add_layers_to_raster_list(
            layers,
            self.dlg.cmb_blue,
            self.dlg.cmb_green,
            self.dlg.cmb_red,
            self.dlg.cmb_nir,
            self.dlg.cmb_b5,
            self.dlg.cmb_lst
        )

    def add_layers_to_raster_list(self, layers, *boxes):
        for box in boxes:
            box.addItems(layers)
        index_list = ["NDVI","ARVI","EVI","LAI","CVI","UTFVI","TDI"]
        self.dlg.cmb_select_index.addItems(index_list)

    def clear_lists(self, *boxes):
        for box in boxes:
            box.clear()
        self.dlg.cmb_select_index.clear()

    def saveRaster(self):
        filename = QFileDialog.getExistingDirectory(self.dlg, "Select folder")
        self.dlg.lineEdit_save_path.setText(filename)

    def blue(self):
        layers = [layer for layer in
QgsProject.instance().mapLayers().values()]
        raster_layers = []
        for layer in layers:
            if layer.type() == QgsMapLayer.RasterLayer:
                raster_layers.append(layer.name())
        self.dlg.cmb_blue.addItems(raster_layers)

    def green(self):
        layers = [layer for layer in
QgsProject.instance().mapLayers().values()]
        raster_layers = []
        for layer in layers:
            if layer.type() == QgsMapLayer.RasterLayer:
                raster_layers.append(layer.name())
        self.dlg.cmb_green.addItems(raster_layers)

    def red(self):
        layers = [layer for layer in
QgsProject.instance().mapLayers().values()]
        raster_layers = []
        for layer in layers:
            if layer.type() == QgsMapLayer.RasterLayer:
                raster_layers.append(layer.name())
        self.dlg.cmb_red.addItems(raster_layers)

    def vnir(self):
        layers = [layer for layer in
QgsProject.instance().mapLayers().values()]
        raster_layers = []
        for layer in layers:
            if layer.type() == QgsMapLayer.RasterLayer:
                raster_layers.append(layer.name())
        self.dlg.cmb_vnir.addItems(raster_layers)

    def nir(self):
        layers = [layer for layer in
QgsProject.instance().mapLayers().values()]
        raster_layers = []
        for layer in layers:
            if layer.type() == QgsMapLayer.RasterLayer:
                raster_layers.append(layer.name())
        self.dlg.cmb_nir.addItems(raster_layers)

    def lst(self):
        print ("defLst")
        layers = [layer for layer in
QgsProject.instance().mapLayers().values()]
        raster_layers = []
        for layer in layers:
            if layer.type() == QgsMapLayer.RasterLayer:
                raster_layers.append(layer.name())
        self.dlg.cmb_lst.addItems(raster_layers)

    def getBlue(self):
        layer = None
        layername = self.dlg.cmb_blue.currentText()
        for lyr in QgsProject.instance().mapLayers().values():
            if lyr.name() == layername:
                layer = lyr
                break
        return layer

    def getGreen(self):
        layer = None
        layername = self.dlg.cmb_green.currentText()
        for lyr in QgsProject.instance().mapLayers().values():
            if lyr.name() == layername:
                layer = lyr
                break
        return layer

    def getRed(self):
        layer = None
        layername = self.dlg.cmb_red.currentText()
        for lyr in QgsProject.instance().mapLayers().values():
            if lyr.name() == layername:
                layer = lyr
                break
        return layer

    def getVNir(self):
        layer = None
        layername = self.dlg.cmb_vnir.currentText()
        for lyr in QgsProject.instance().mapLayers().values():
            if lyr.name() == layername:
                layer = lyr
                break
        return layer

    def getNir(self):
        layer = None
        layername = self.dlg.cmb_nir.currentText()
        for lyr in QgsProject.instance().mapLayers().values():
            if lyr.name() == layername:
                layer = lyr
                print (lyr.name())
                print (layername)
                print(layer)
                break
        return layer

    def getLst(self):
        layer = None
        layername = self.dlg.cmb_lst.currentText()
        for lyr in QgsProject.instance().mapLayers().values():
            if lyr.name() == layername:
                layer = lyr
                print (lyr.name())
                print (layername)
                print(layer)
                break
        return layer
    #Code OK!
    def calc_ndvi(self):
        lyr1 = self.getRed()
        lyr2 = self.getNir()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save NDVI
Output", f"{self.dlg.lineEdit_save_path.text()}/ndvi.tif", "Raster Files
(*.tif *.asc *.img)")

            red_entry = QgsRasterCalculatorEntry()
            red_entry.ref = 'red_band@1'
            red_entry.raster = lyr1
            red_entry.bandNumber = 1

            nir_entry = QgsRasterCalculatorEntry()
            nir_entry.ref = 'nir_band@1'
            nir_entry.raster = lyr2
            nir_entry.bandNumber = 1

            ndvi_formula = '(nir_band@1 - red_band@1) / (nir_band@1 +
red_band@1)'

            ndvi_calculator = QgsRasterCalculator(
                ndvi_formula,
                output_path,
                'GTiff',
                lyr1.extent(),
                lyr1.width(),
                lyr1.height(),
                [red_entry, nir_entry]
                )
            ndvi_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                ndvi_layer = QgsRasterLayer(output_path, 'NDVI', 'gdal')
                QgsProject.instance().addMapLayer(ndvi_layer)
            self.iface.messageBar().pushMessage("NDVI Output Created
Successfully",level = Qgis.Success,duration = 10)
        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase
select output path.",level = Qgis.Warning,duration = 10)
    #Code OK!
    def calc_arvi(self):
        lyr1 = self.getRed()
        lyr2 = self.getNir()
        lyr3 = self.getBlue()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save ARVI
Output", f"{self.dlg.lineEdit_save_path.text()}/arvi.tif", "Raster Files
(*.tif *.asc *.img)")

            red_entry = QgsRasterCalculatorEntry()
            red_entry.ref = 'red_band@1'
            red_entry.raster = lyr1
            red_entry.bandNumber = 1

            nir_entry = QgsRasterCalculatorEntry()
            nir_entry.ref = 'nir_band@1'
            nir_entry.raster = lyr2
            nir_entry.bandNumber = 1

            blue_entry = QgsRasterCalculatorEntry()
            blue_entry.ref = 'blue_band@1'
            blue_entry.raster = lyr3
            blue_entry.bandNumber = 1

            arvi_formula = '(nir_band@1 - (2*red_band@1)+blue_band@1) /
(nir_band@1 + (2*red_band@1)+blue_band@1)'

            arvi_calculator = QgsRasterCalculator(
                arvi_formula,
                output_path,
                'GTiff',
                lyr1.extent(),
                lyr1.width(),
                lyr1.height(),
                [red_entry, nir_entry, blue_entry]
                )

            arvi_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                arvi_layer = QgsRasterLayer(output_path, 'ARVI', 'gdal')
                QgsProject.instance().addMapLayer(arvi_layer)
            self.iface.messageBar().pushMessage("ARVI Output Created
Successfully",level = Qgis.Success,duration = 10)
        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase
select output path.",level = Qgis.Warning,duration = 10)
    #Code OK!
    def calc_evi(self):
        lyr1 = self.getRed()
        lyr2 = self.getNir()
        lyr3 = self.getBlue()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save EVI
Output", f"{self.dlg.lineEdit_save_path.text()}/evi.tif", "Raster Files (*.tif
*.asc *.img)")

            red_entry = QgsRasterCalculatorEntry()
            red_entry.ref = 'red_band@1'
            red_entry.raster = lyr1
            red_entry.bandNumber = 1

            nir_entry = QgsRasterCalculatorEntry()
            nir_entry.ref = 'nir_band@1'
            nir_entry.raster = lyr2
            nir_entry.bandNumber = 1

            blue_entry = QgsRasterCalculatorEntry()
            blue_entry.ref = 'blue_band@1'
            blue_entry.raster = lyr3
            blue_entry.bandNumber = 1

            evi_formula = '2.5*((nir_band@1 -
red_band@1)/(nir_band@1+6*red_band@1-7.5*+blue_band@1+1))'

            evi_calculator = QgsRasterCalculator(
                evi_formula,
                output_path,
                'GTiff',
                lyr1.extent(),
                lyr1.width(),
                lyr1.height(),
                [red_entry, nir_entry, blue_entry]
                )

            evi_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                evi_layer = QgsRasterLayer(output_path, 'EVI', 'gdal')
                QgsProject.instance().addMapLayer(evi_layer)
            self.iface.messageBar().pushMessage("EVI Output Created
Successfully",level = Qgis.Success,duration = 10)
        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase
select output path.",level = Qgis.Warning,duration = 10)
    #Code OK!
    def calc_lai(self):
        lyr1 = self.getRed()
        lyr2 = self.getNir()
        lyr3 = self.getBlue()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save LAI 
Output", f"{self.dlg.lineEdit_save_path.text()}/lai.tif", "Raster Files (*.tif 
*.asc *.img)")

            red_entry = QgsRasterCalculatorEntry()
            red_entry.ref = 'red_band@1'
            red_entry.raster = lyr1
            red_entry.bandNumber = 1

            nir_entry = QgsRasterCalculatorEntry()
            nir_entry.ref = 'nir_band@1'
            nir_entry.raster = lyr2
            nir_entry.bandNumber = 1

            blue_entry = QgsRasterCalculatorEntry()
            blue_entry.ref = 'blue_band@1'
            blue_entry.raster = lyr3
            blue_entry.bandNumber = 1

            lai_formula = '3.618*((2.5*((nir_band@1 - 
red_band@1)/(nir_band@1+6*red_band@1-7.5*+blue_band@1+1)))-0.118)'

            lai_calculator = QgsRasterCalculator(
                lai_formula,
                output_path,
                'GTiff',
                lyr1.extent(),
                lyr1.width(),
                lyr1.height(),
                [red_entry, nir_entry, blue_entry]
                )

            lai_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                lai_layer = QgsRasterLayer(output_path, 'LAI', 'gdal')
                QgsProject.instance().addMapLayer(lai_layer)

            self.iface.messageBar().pushMessage("LAI Output Created 
Successfully",level = Qgis.Success,duration = 10)

        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase 
select output path.",level = Qgis.Warning,duration = 10)
    #Code OK!
    def calc_cvi(self):
        lyr1 = self.getRed()
        lyr2 = self.getNir()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save CVI
Output", f"{self.dlg.lineEdit_save_path.text()}/cvi.tif", "Raster Files (*.tif
*.asc *.img)")

            red_entry = QgsRasterCalculatorEntry()
            red_entry.ref = 'red_band@1'
            red_entry.raster = lyr1
            red_entry.bandNumber = 1

            nir_entry = QgsRasterCalculatorEntry()
            nir_entry.ref = 'nir_band@1'
            nir_entry.raster = lyr2
            nir_entry.bandNumber = 1

            cvi_formula = '(nir_band@1 / red_band@1)-1'

            cvi_calculator = QgsRasterCalculator(
                cvi_formula,
                output_path,
                'GTiff',
                lyr1.extent(),
                lyr1.width(),
                lyr1.height(),
                [red_entry, nir_entry]
                )

            cvi_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                cvi_layer = QgsRasterLayer(output_path, 'CVI', 'gdal')
                QgsProject.instance().addMapLayer(cvi_layer)

            self.iface.messageBar().pushMessage("CVI Output Created
Successfully",level = Qgis.Success,duration = 10)

        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase
select output path.",level = Qgis.Warning,duration = 10)
    #Code OK!
    def calc_utfvi(self):
        lyr1 = self.getLst()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save UTFVI
Output", f"{self.dlg.lineEdit_save_path.text()}/utfvi.tif", "Raster Files
(*.tif *.asc *.img)")
```

```
            lst_entry = QgsRasterCalculatorEntry()
            lst_entry.ref = 'lst_band@1'
            lst_entry.raster = lyr1
            lst_entry.bandNumber = 1
            MEAN = float(self.dlg.lineEdit_rh.text())
            utfvi_formula = '(lst_band@1 - '+str(MEAN)+') / '+str(MEAN)
            #utfvi_formula = '(lst_band@1 - STATISTICS_MEANlst_band@1) / 
STATISTICS_MEANlst_band@1'
            utfvi_calculator = 
QgsRasterCalculator(utfvi_formula,output_path,'GTiff',lyr1.extent(),lyr1.width
(),lyr1.height(),[lst_entry])

            utfvi_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                utfvi_layer = QgsRasterLayer(output_path, 'UTFVI', 'gdal')
                QgsProject.instance().addMapLayer(utfvi_layer)

            self.iface.messageBar().pushMessage("UTFVI Output Created 
Successfully",level = Qgis.Success,duration = 10)

        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase 
select output path.",level = Qgis.Warning,duration = 
10)
    #Code OK!
    def calc_tdi(self):
        lyr1 = self.getLst()

        if self.dlg.lineEdit_save_path.text() != "":
            output_path, _ = QFileDialog.getSaveFileName(None, "Save TDI 
Output", f"{self.dlg.lineEdit_save_path.text()}/tdi.tif", "Raster Files (*.tif 
*.asc *.img)")

            lst_entry = QgsRasterCalculatorEntry()
            lst_entry.ref = 'lst_band@1'
            lst_entry.raster = lyr1
            lst_entry.bandNumber = 1
            RH = float(self.dlg.lineEdit_rh.text())
            tdi_formula = 'lst_band@1-0.55*(1-0.01*'+str(RH)+')*(lst_band@1-
14.5)'
            tdi_calculator = 
QgsRasterCalculator(tdi_formula,output_path,'GTiff',lyr1.extent(),lyr1.width()
,lyr1.height(),[lst_entry])

            tdi_calculator.processCalculation()
            if self.dlg.chk_load_output.isChecked():
                tdi_layer = QgsRasterLayer(output_path, 'TDI', 'gdal')
                QgsProject.instance().addMapLayer(tdi_layer)
```

```python
            self.iface.messageBar().pushMessage("TDI Output Created 
Successfully",level = Qgis.Success,duration = 10)

        else:
            self.iface.messageBar().pushMessage("Output path is NULL! Plase 
select output path.",level = Qgis.Warning,duration = 10)

    def calculation(self):
        if self.dlg.cmb_select_index.currentText() == "NDVI":
            self.calc_ndvi()
        if self.dlg.cmb_select_index.currentText() == "ARVI":
            self.calc_arvi()
        if self.dlg.cmb_select_index.currentText() == "EVI":
            self.calc_evi()
        if self.dlg.cmb_select_index.currentText() == "LAI":
            self.calc_lai()
        if self.dlg.cmb_select_index.currentText() == "CVI":
            self.calc_cvi()
        if self.dlg.cmb_select_index.currentText() == "UTFVI":
            self.calc_utfvi()
        if self.dlg.cmb_select_index.currentText() == "TDI":
            self.calc_tdi()

    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginRasterMenu(
                self.tr(u'&vegetation_indices'),
                action)
            self.iface.removeToolBarIcon(action)

    def run(self):
        """Run method that performs all the real work"""
        # Create the dialog with elements (after translation) and keep 
reference
        # Only create GUI ONCE in callback, so that it will only load when the 
plugin is started
        if self.first_start == True:
            self.first_start = False
            self.dlg = vegetation_indicesDialog()
        # show the dialog
        self.update_raster_list()
        self.dlg.lineEdit_save_path.clear()
        self.dlg.btn_klasor_sec.clicked.connect(self.saveRaster)
        self.dlg.btn_OK.clicked.connect(self.calculation)
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            # Do something useful here - delete the line containing pass and
            # substitute with your code.
            # self.final()
            #self.iface.messageBar().pushMessage("Output Created 
Successfully", level=Qgis.Success, duration=3)
            # self.dlg.tb_output.clicked.disconnect(self.saveRaster)
            pass
```