Université de Genève

Faculté des Sciences de la Société

Département de géographie et environnement
Certificat complémentaire en géomatique

Evaluation du potentiel de valorisation du Swiss Data Cube pour les parcelles culturales genevoises

Etudiant : Alexander Folz

Superviseurs :
Bruno Chatenoux (Directeur de mémoire, UNEP/GRID-Genève)
Gregory Giuliani (Co-directeur de mémoire, UNEP/GRID-Genève)
Justine Grespan (Superviseure, OCAN)

30.10.2019

Résumé

Dans le contexte d'une agriculture moderne vulnérable à diverses instabilités naturelles et sociales, cette étude a montré le potentiel du Swiss Data Cube pour créer des indicateurs saisonniers de suivi environnemental des parcelles culturales genevoises. Le ratio de l'indice NDVI et NDWI d'une culture permet d'informer sur sa susceptibilité à un stress hydrique. L'analyse de la tendance saisonnière 1984-2019 a montré qu'il existait des groupes de culture avec une tendance caractéristique : les cultures arboricoles, horticoles, la vigne et le blé de printemps qui dénotent une tendance positive du NDVI et du NDWI en 36 ans ; les prairies, pâturages, autres espaces peu intensifs et cultures maraîchères témoignant d'une importante augmentation du NDVI et d'une hausse du NDWI plus modeste, voire nulle ; les grandes cultures, dont l'activité photosynthétique n'a que modestement augmenté et l'humidité foliaire est resté proche de constant. Résultant d'une augmentation de la température de surface, les tendances NDVI et NDWI reflètent des traits physiologiques et phénologiques des cultures distincts : la vigne témoigne d'une résistance à la sécheresse plus élevée, tandis que les prairies et les grandes cultures sont plus sensibles à un déficit hydrique. Ces différences de tendances sont notamment expliquées par le fait qu'une culture d'automne est atteinte par une augmentation de la température plus importante qu'une culture d'été. Malgré le potentiel de cette étude pour informer sur le statut environnemental des parcelles culturales genevoises, les résultats issus de cette étude sont exploratoires et nécessitent la validation d'études statistiques ultérieures.

Tables des matières

Abréviations

ARD	Analysis Ready Data
DGAN	Direction générale de l'agriculture et de la nature
GIEC	Groupe d'experts intergouvernemental sur l'évolution du climat
GRID	Global Resource Information Database
JRC	European Commission Joint Research Center
NDVI	Normalized Difference Vegetation Index
NDWI	Normalized Difference Water Index
NIR	Near Infrared
NPCRI	Normalized Pigment Chlorophyll Ratio Index
OCAN	Office Cantonal de l'Agriculture et de la Nature
OFEV	l'Office fédéral de l'environnement
PNUE	Programme des Nations Unies pour l'environnement
SWIR	Short-wave Infrared

Liste des Figures

I. Présentation des organisations hôtes

Le GRID – Genève (GRID – Global Resource Information Database) est une instance du groupe de centres d'information internationaux de la Division Scientifique du Programme des Nations Unies pour l'environnement (PNUE). Le GRID – Genève est un partenariat entre le Programme des Nations Unies pour l'Environnement, l'Office fédéral de l'environnement (OFEV) et l'université de Genève (UniGE). Le mandat du centre est de transformer les données en information afin de supporter la prise de décision liée aux questions environnementales. Un des domaines d'excellence du centre est la télédétection, opérationnalisée pour la Suisse par le Swiss Data Cube, une base de données regroupant une archive d'images satellites temporelles de la Suisse à haute résolution.

L'OCAN (Office Cantonal de l'Agriculture et de la Nature) est garant de la préservation des terres agricoles à l'échelle du Canton de Genève et supporte le développement d'une agriculture autosuffisante, écologique et rémunératrice.

C'est au sein de ces deux institutions qu'il m'a été offert la possibilité d'effectuer un stage de 3 mois sur le potentiel du Swiss Data Cube dans la création d'indicateurs environnementaux de suivi des parcelles culturales genevoises.

II. Introduction

Dans un contexte d'augmentation des températures de la surface terrestre - de 1.53°C par rapport à la moyenne de 1850-1900 - de dégradation des sols, et de surexploitation des ressources naturelles d'une part (Arneth et al. , 2019), d'augmentation de la demande en produits alimentaires d'origine animale (ibid. , 2019) et de précarisation des agriculteurs d'autre part – seul 13.9% du prix final d'un produit agricole à la vente en supermarché revient à l'agriculteur en moyenne à l'échelle mondiale - (Oxfam, 2018 Figure 8.), le système de production alimentaire actuel est controversé.

En effet, l'agriculture paysanne occupe en moyenne seulement 25% des terres agricoles mondiales, mais produit dans de nombreux pays une part conséquente de la nourriture consommée – tel qu'au Brésil ou en Russie – (Grain, 2014 ; FAO, 2009). L'élevage animal, nécessitant pâturages et denrées alimentaires comme fourrage, mobilise 70% des terres agricoles mondiales (Steinfeld et al., 2006 ; Grain, 2014) et 27% de l'eau utilisée par l'homme annuellement (dont 23% de l'eau évapo-transpirée par les cultures et pâturages ainsi que 12% de l'eau douce consommée à l'échelle mondiale) (Mekonnen & Hoekstra, 2011 ; Mekonnen & Hoekstra, 2012 ; Hoekstra & Mekonnen, 2012).

Au total, selon le GIEC (Groupe d'experts intergouvernemental sur l'évolution du climat), l'agriculture moderne contribue entre 21 et 37% des émissions mondiales de gaz à effet de serre (Arneth et al. , 2019). En Suisse, l'import net[1] de denrées alimentaires représente 45% des émissions de gaz à effet de serre de l'alimentation suisse (Bretscher et al., 2014). Une agriculture paysanne autosuffisante garante de l'approvisionnement alimentaire local apparaît donc essentielle pour le maintien d'une société cohérente avec l'environnement naturel. Afin de contribuer à la transition vers une agriculture résiliente à ces interdépendances, le potentiel de l'imagerie satellitaire pour suivre l'état environnemental des parcelles agricoles gagne à être évalué.

[1] Ces émissions de gaz à effet de serre importées correspondent aux émissions de gaz à effet de serre générées par l'importation de denrées alimentaires déduites de celles générées par l'exportation.

Suivant les directives de la Confédération, les exploitants des parcelles agricoles au bénéfice de paiements directs sont depuis 2017 dans l'obligation de géoréférencer chaque année leurs parcelles en précisant leur culture principale. En 2018, la surface agricole utile était de 8'959 hectares, soit une nette diminution par rapport à l'année 2017, 9'373 hectares. Près de la moitié de la surface agricole utile est occupée par des parcelles de blé d'automne, de vignes et de prairies extensives (Figure 1).

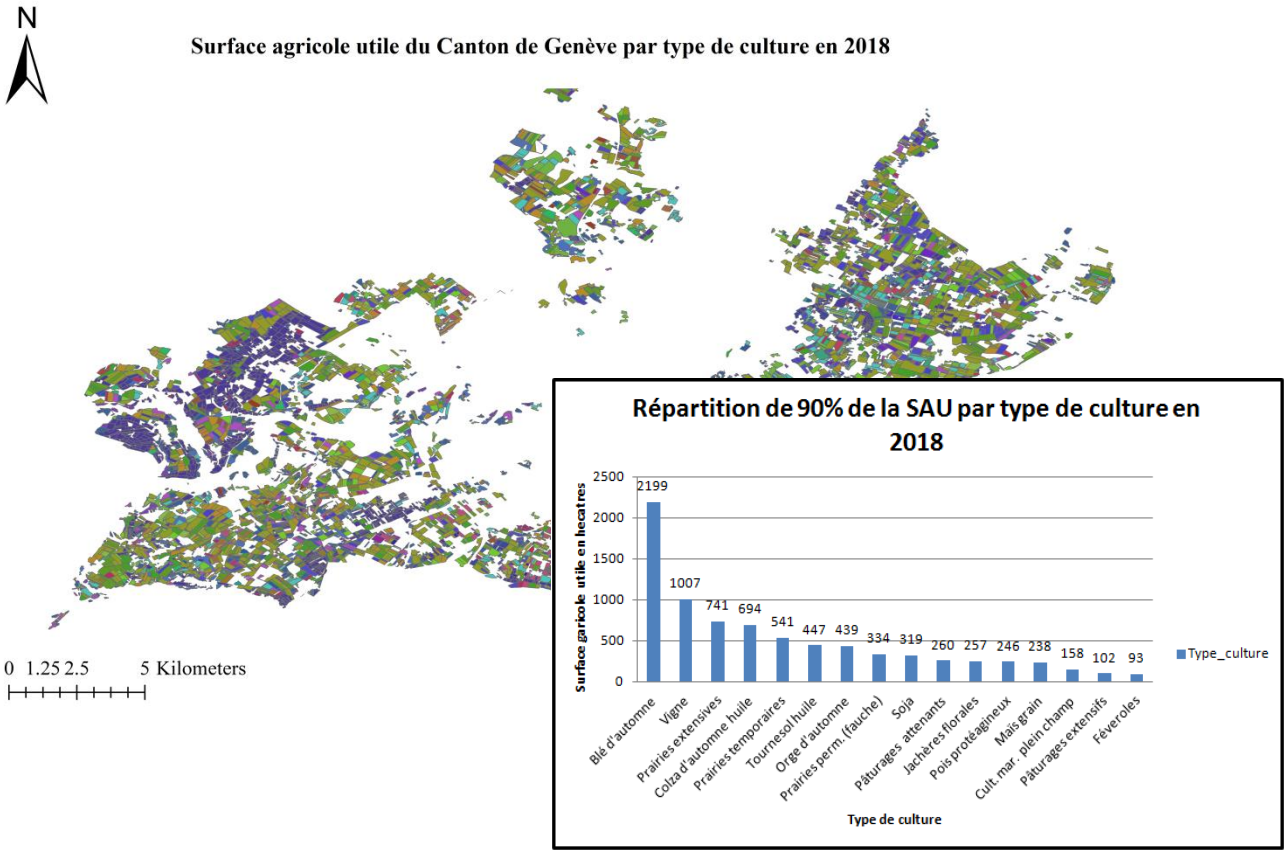


Figure 1: Répartition de 90% de la surface agricole utile par type de culture dans le Canton de Genève en 2018

Dans ce contexte, ce stage vise à valoriser la base de données spatiales GeoACORDA de l'office cantonal de l'agriculture et de la nature (OCAN) en calculant des indices de suivi environnemental (santé du couvert végétal, humidité foliaire) des parcelles culturales genevoises à l'aide du Swiss Data Cube.

Les objectifs de ce stage sont d'une part, de créer une base de données temporelle saisonnière de 1984 à 2019 d'indices environnementaux pour chaque parcelle culturale genevoise dans un but d'aide aux exploitants. D'autre part, ce travail vise à déterminer la tendance temporelle de ces indices pour chaque parcelle et type de culture.

III. Méthodes et données

Le Swiss Data Cube est un serveur stockant, pour toute la Suisse, les images satellitaires Landsat de 1984 à 2019, à 30 m de résolution (Surface Reflectance Landsat Analysis Ready Data), et de Sentinel 2 dès 2015, à 10 m de résolution (Sentinel 2 2A product classification). Dans le cadre de ce stage, seules ont été utilisées les images des satellites Landsat 5, 7 et 8, couvrant la période 1984-2019. Une scène satellite Landsat a une résolution de 30m, chaque pixel contient donc de l'information pour 900 m² de superficie (<0.1 ha). L'emprise spatiale utilisée dans cette analyse - latitude (46.129, 46.373) et longitude (5.9555, 6.319) - couvre le canton de Genève, ainsi que les régions limitrophes du canton de Vaud et de France voisine (Figure 2).

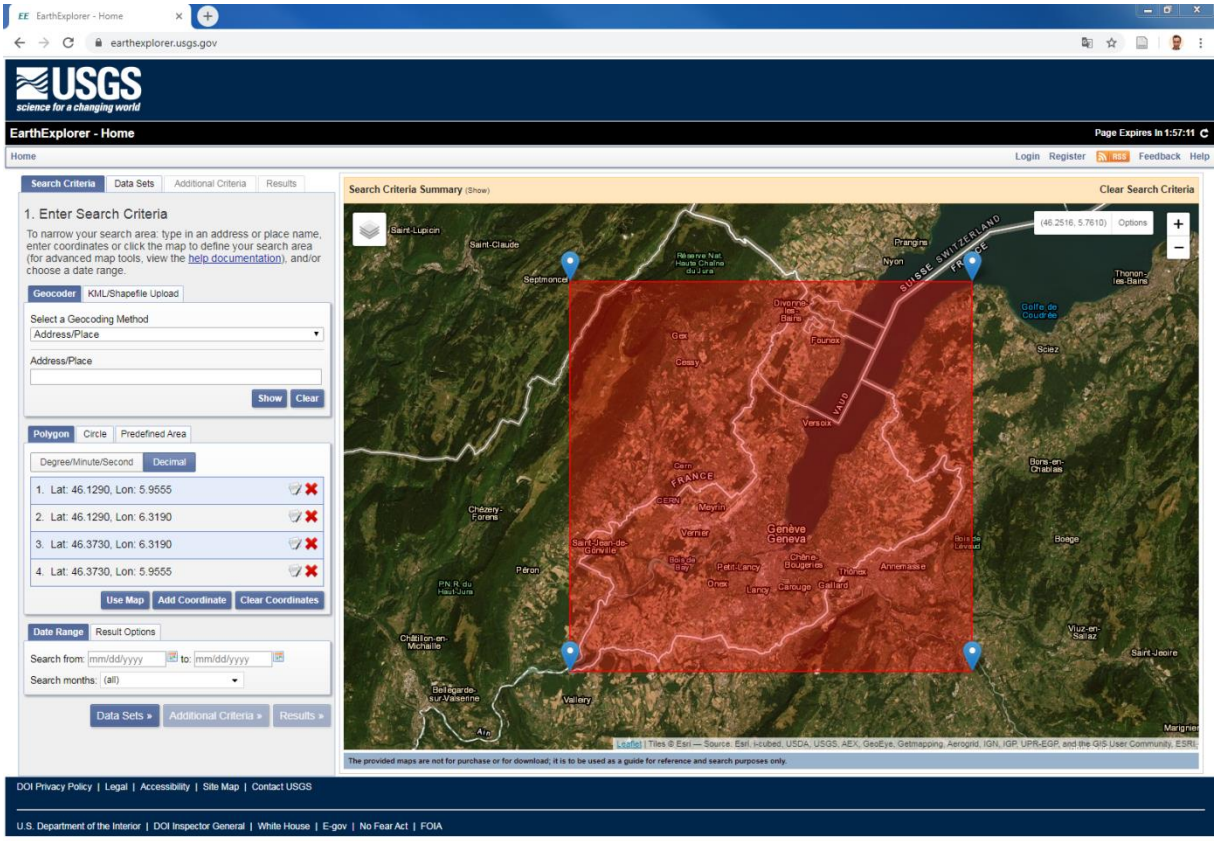


Figure 2: Emprise géographique des images Landsat utilisées

Chaque pixel d'une image Landsat (ARD) stocke les valeurs de réflexion de surface de la lumière solaire dans plusieurs bandes de longueur d'onde, allant du visible (0.43-0.67 µm) à l'infrarouge (0.85-12.51 µm). Les réflexions des différentes bandes de longueur d'onde varient en fonction du type de surface qui réfléchit la lumière solaire. Ainsi, en se basant sur les

différentes propriétés physiques de chaque couverture du sol, il est possible de combiner certaines bandes afin de dériver des indices de suivi environnemental :

L'analyse de la littérature a permis d'identifier les différents indices de télédétection utilisés pour caractériser la vigueur de la végétation (Xue & Su, 2017). L'indice de végétation par différence normalisé (NDVI) montre une forte corrélation avec la structure de la canopée foliaire, l'indice de surface foliaire, ainsi qu'avec l'activité photosynthétique de la canopée foliaire (Gamon et al, 1995 cited in : Xue & Su, 2017). Par conséquent, cet indice a été utilisé pour mesurer l'état de santé de la végétation dans de nombreuses recherches (Xue & Su, 2017). Ainsi, en raison de contraintes de comparabilité et de limites de temps, seul l'indice de végétation par différence normalisé (NDVI) a été retenu. Récemment, des indices issus de la télédetection pour mesurer le contenu en eau des feuilles ont été développés, variant en précision, et par conséquent, en complexité : tel que l'évapotranspiration actuelle d'une couverture du sol en appliquant le modèle de balance d'énergie de surface METRIC - ce modèle nécessite notamment des données de stations météorologiques et topographiques en entrée -; ou comme l'indice d'humidité foliaire par différence normalisé (NDWI). Etant donné la durée limitée de ce stage, l'indice NDWI a été retenu pour mesurer le contenu en eau foliaire des parcelles culturales genevoises.

L'indice NDVI permet de mesurer le degré d'activité photosynthétique de la végétation, soit l'intensité de « verdure » de la surface foliaire. (Tucker, 1979). L'indice NDVI est calculé comme suit :

$$NDVI = \frac{NIR - R}{NIR + R}$$

Où NIR réfère à la bande de longueur d'onde de l'infrarouge proche (0.7-0.9 µm) et R réfère la bande de longueur du rouge dans le spectre du visible (0.6-0.7 µm). En effet, la structure interne des feuilles réfléchit le spectre du proche infrarouge, tandis que le pigment de chlorophylle présent dans la feuille absorbe fortement le rouge et le bleu dans le spectre du visible dans le processus de la photosynthèse, et réfléchit le vert, ce qui donne à la végétation sa couleur caractéristique[2] (Tucker, 1979).

[2] Levy R., 2019. Measuring vegetation (NDVI & EVI): Normalized Difference Vegetation Index (NDVI). NASA Goddard Space Flight Center.
https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring_vegetation_2.php

Les propriétés optiques d'une feuille établie par spectrophotométrie en laboratoire, ainsi qu'un modèle testant la variation de réflexion du spectre électromagnétique ont montré qu'au-delà de 8 feuilles empilées les unes sur les autres, la réflexion dans l'infrarouge restait constante, tandis que la réflexion dans le spectre rouge visible se stabilisait après une couche de feuilles.(Lillesaeter, 1982)

L'indice NDWI permet de mesurer le contenu en eau liquide de la végétation, ainsi que la densité et « verdure » de la surface foliaire. (Gao, 1997 ; JRC, 2011). L'indice NDWI est calculé comme suit :

$$NDWI = \frac{NIR - SWIR}{NIR + SWIR}$$

Où NIR réfère à la bande de longueur d'onde de l'infrarouge proche (0.7-0.9 µm) et SWIR1 (Short-Wave Infrared 1) réfère à la bande de longueur d'onde de l'infrarouge moyen 1 (1.55-1.7 µm). La bande spectrale NIR capture l'effet de diffusion de la lumière au contact de la structure interne des feuilles, – le mésophylle spongieux - et la bande SWIR1 capture également, dans une moindre mesure, la réflexion du mésophylle spongieux, mais absorbe surtout le contenu en eau liquide des feuilles. Ainsi, la différence normalisée des deux bandes permet d'approximer l'absorption de la bande SWIR1 par l'eau liquide de la feuille (Gao, 1997). L'indice NDWI permet de suivre le degré de flétrissement de la plante, potentiellement aussi lié à d'autres facteurs que des épisodes de sécheresse tel que le changement de couverture du sol, la prolifération d'insectes ravageurs ou micro-organismes etc. (JRC, 2011). A condition d'être comparé à des sources météorologiques lors d'épisodes de sécheresses, cet indice permet de manière robuste de suivre à haute résolution le stress hydrique de la végétation au cours du temps (ibid, 2011). Gu (2007) a notamment montré sur une zone de prairies du centre des Grandes Plaines aux Etats-Unis que le NDWI était plus sensible que le NDVI lors de sécheresses, soit la variation négative du NDWI par rapport à la moyenne 2001-2005 était deux fois plus importante que celle du NDVI à la même période.
Selon Gao (1997), la capacité de l'indice NDWI à mesurer le contenu en eau liquide des feuilles fait de celui-ci un indicateur complémentaire au NDVI dans l'évaluation du statut de la végétation.

Ce stage a permis (1) d'établir une base de données spatiale de ces indices environnementaux par parcelle agricole et type de culture, comprenant :

- Les moyennes saisonnières pour chaque année de 1984 à 2019
- Les déviations standards saisonnières pour chaque année de 1984 à 2019
- Les moyennes mensuelles pour chaque année de 1984 à 2019
- Les déviations standards mensuelles pour chaque année de 1984 à 2019

Ainsi que (2) une tendance linéaire non paramétrique de ces indices saisonniers pour la période 1984-2019.

Afin de calculer les indices NDVI and NDWI saisonniers, le script python Jupyter Notebook *AF_NDVI_NDWI_seasonal_stats_generator. ipynb* a été exécuté pour générer un fichier NETCDF par saison pour chaque année (Figure 3, haut). Le script *AF_NDVI_NDWI_monthly_stats_generator.ipynb* a été exécuté pour générer les fichiers mensuels *(*Annexe 3*)*.

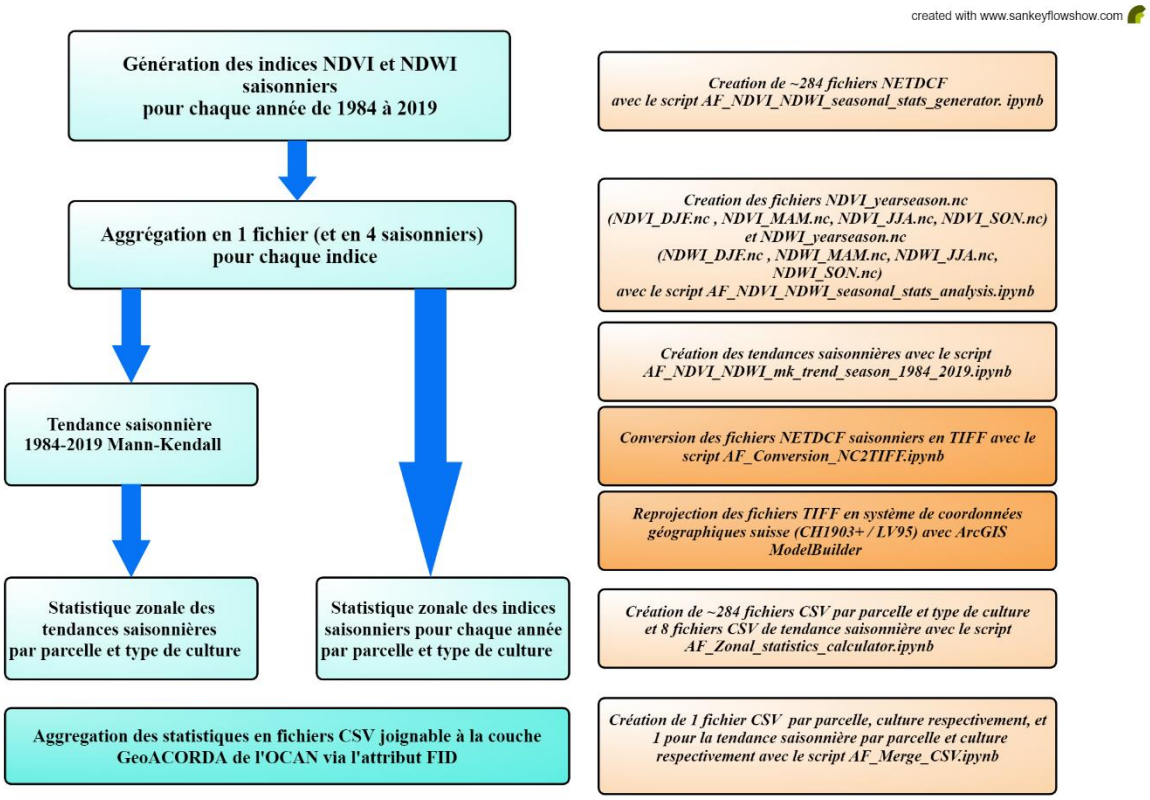


Figure 3: Méthode de génération des statistiques zonales saisonnières pour le canton de Genève

Pour garantir que chaque fichier NETCDF saisonnier et mensuel exclue les pixels couverts par les nuages ou les ombres de nuages de l'analyse, un masque basé sur la bande Landsat **pixel_qa** (Quality Assessment) a été appliqué via la fonction *load_multi_clean().* L'élimination des scènes satellites inexploitables a éliminé ~30 mois sur 36 années d'observations (le plus souvent en période hivernale). Par conséquent, les moyennes saisonnières générées pour chaque année sont basées sur jusqu'à ~8 observations en été à moins de ~1 observations en hiver (Annexe 1, Annexe 2). De ce fait, les moyennes mensuelles et saisonnières annuelles peuvent être jaugées statistiquement insuffisantes. Néanmoins, étant donné la nature précise des observations satellitaires et que ce stage vise à évaluer la tendance de ces observations, ce nombre d'observations est jugé suffisant pour la conduite de cette analyse. Les fonctions np.nanmean(), np.nanstd() et np.nancount ont permis de calculer les moyennes et déviations standards des indices sans prendre en compte les pixels sans valeurs (assignés aux nuages et aux ombres des nuages).

Le script *AF_NDVI_NDWI_seasonal_stats_analysis.ipynb* – et son équivalent mensuel *AF_NDVI_NDWI_monthly_stats_analysis.ipynb* – a permis de lister différents fichiers NETCDF crées et les a combiné pour obtenir un jeu de données saisonnier – mensuel respectivement- de NDVI et NDWI en format NETCDF de 1984 à 2019 (Figure 3, milieu). Consécutivement, un masque assignant une valeur nulle aux pixels couvrant les surfaces d'eau (Lac Léman, Rhône, Arve) a été appliqué au jeu de données saisonnier afin d'éliminer tout biais pouvant résulter de l'interprétation des indices de végétation. Il est important de mentionner qu'après vérification des données mensuelles, étant donné le nombre important de données manquantes, la création de statistiques zonales mensuelles pour chaque année, ainsi que les tendances, par parcelle et type de culture, n'ont pas été entreprises. Ensuite, le script Jupyter Notebook *AF_NDVI_NDWI_mk_trend_season_1984_2019.ipynb* a calculé la tendance statistique non-paramétrique des indices environnementaux pour chaque saison durant la période 1984-2019 (Figure 3, milieu gauche). Le test de tendance Mann-Kendall consiste à réfuter l'hypothèse H_0 que les observations de la série de données – ici les indices de suivi de la végétation – sont indépendantes les unes des autres et sont distribuées de manière identique. L'hypothèse alternative H_1 est qu'il y a une tendance temporelle monotonique dans la série de données (Hipel, 1994). Si la statistique de test Mann-Kendall S est différente de 0, alors il existe une tendance monotonique dans la série de données :

$$S = \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} sgn(x_j - x_k)$$

$$\text{Où } sgn(x_j - x_k) = \begin{cases} 1, sgn(x_j - x_k) > 0 \\ 0, sgn(x_j - x_k) = 0 \\ -1, sgn(x_j - x_k) < 0 \end{cases}$$

Cette valeur S est la somme des n * (n - 1) différences entre les valeurs d'observations x_j et les valeurs des observations antérieures x_k. Dans notre analyse, le nombre d'observation est donc de 35 * 36 = 1260 observations. Dans le but de déterminer le signe de la tendance, le script calcule le coefficient de corrélation pour chaque pixel de la série de données. Par ailleurs, celui-ci assigne à la tendance monotonique de chaque pixel les valeurs 1, 2 et 3 en fonction de la probabilité d'occurrence de cette tendance (90, 95 et 99 % respectivement).

Afin de créer les statistiques zonales de ces indices, il a été nécessaire de procéder en deux étapes : convertir les fichiers NETCDF en TIFFs via le script *AF_Conversion_NC2TIFF.ipynb* , puis reprojeter ces fichiers TIFFs - à l'aide de l'outil d'automatisation ModelBuilder d'ArcGIS - en système de coordonnées projetées suisse, soit dans le même système de coordonnées géographiques que la couche GEOACCORDA des parcelles culturales (Figure 3, milieu droit). Ensuite, l'exécution du script *AF_Zonal_statistics_calculator.ipynb* a permis de générer des fichiers CSV – moyennes, déviations standards - des indices NDVI et NDWI saisonniers par parcelle et type de culture pour chaque année de 1984 à 2019 (Figure 3, milieu bas). Ces fichiers ont ensuite été agrégés en fichiers CSV contenant ces statistiques à l'aide du script *AF_Merge_CSV.ipynb* (Figure 3, bas). Cette même procédure a été utilisée pour générer les statistiques zonales des tendances de NDWI et NDVI par parcelle et culture.

Dans le but de déterminer les indicateurs utiles et vérifier la qualité des données, et ainsi, juger de la pertinence d'une mise à disposition de ces données aux exploitants agricoles genevois, des réunions régulières à l'OCAN et un entretien avec un viticulteur de l'OCAN ont été menées : ces entretiens ont notamment permis de déterminer les causes des variations temporelles observées dans les indices NDVI et NDWI pour une parcelle de vigne, confirmant l'utilité potentielle de ces indicateurs.

IV. Résultats

La Figure 4 représente le ratio NDVI /NDWI de chaque culture en fonction de l'indice NDWI pendant la saison de croissance végétale maximale. On constate que les cultures affichant un ratio NDVI /NDWI élevé (faible) couplé à un indice NDWI faible (élevé), témoignent d' une faible (forte) susceptibilité à un stress hydrique.

Les prairies, ainsi que le colza d'automne et les cultures de blé d'automne affichent des ratios NDVI/NDWI modérés à faibles montrant que ces cultures sont sensibles à un déficit hydrique (Figure 4, Figure 6). La vigne, en revanche, a un ratio NDVI/NDWI plus élevé (Figure 4, Figure 6). En effet, l'évolution temporelle du NDVI et du NDWI de ces cultures confirment cette typologie (Figure 5). On remarque que pour un indice NDVI de la vigne similaire à celui du blé d'automne, son indice NDWI est beaucoup plus faible, indiquant que la vigne est résistante à un déficit hydrique (Figure 5).

Cependant, le stress hydrique n'est qu'un seul facteur influençant la croissance de la photosynthèse, et par conséquent le ratio NDVI/NDWI. En effet, la température, la disponibilité en CO_2, ou celle d'autres nutriments tel que l'azote et le phosphore sont déterminantes. Ces ratios doivent donc être interprétés en considérant quel facteur est l'élément limitant de chaque culture.

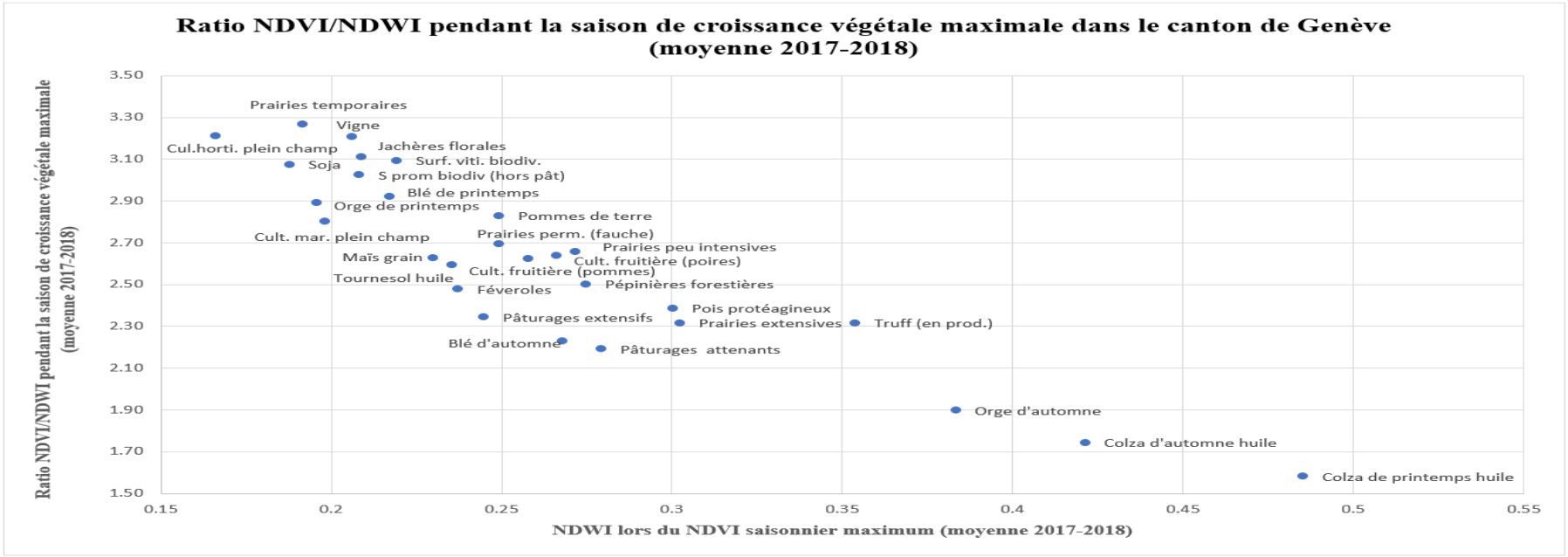


Figure 4: Ratio NDVI/NDWI en fonction du NDWI par culture pendant la saison de croissance végétale maximale dans le canton de Genève (moyenne 2017-2018)

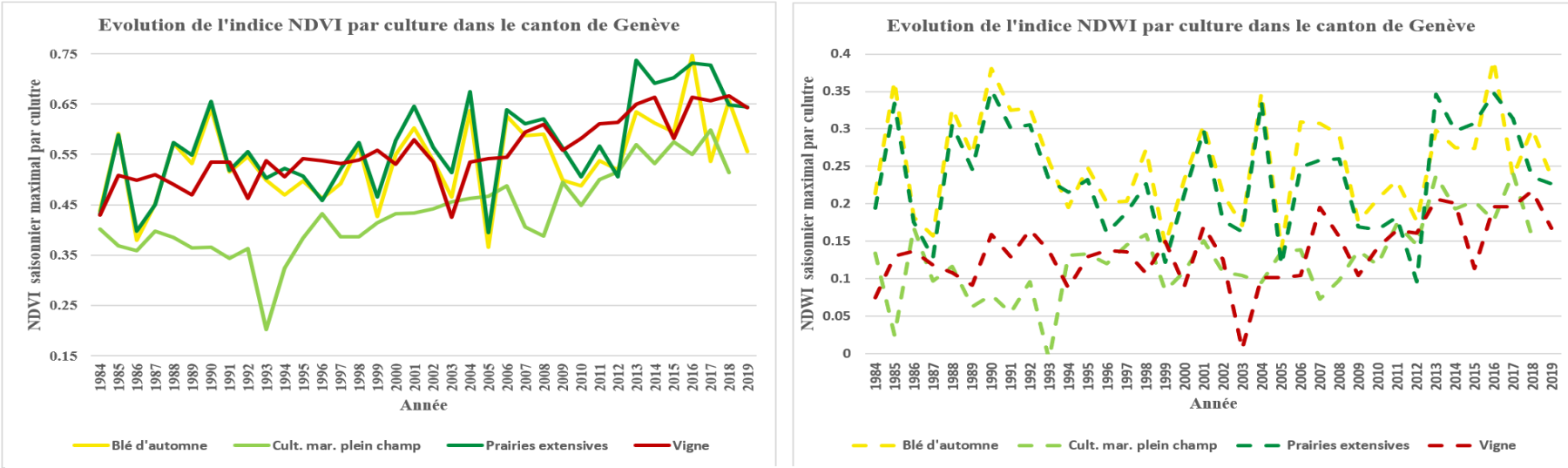


Figure 5: Evolution de l'indice NDVI (gauche) et NDWI (droite) par culture pendant la période de croissance végétale maximum dans le canton de Genève

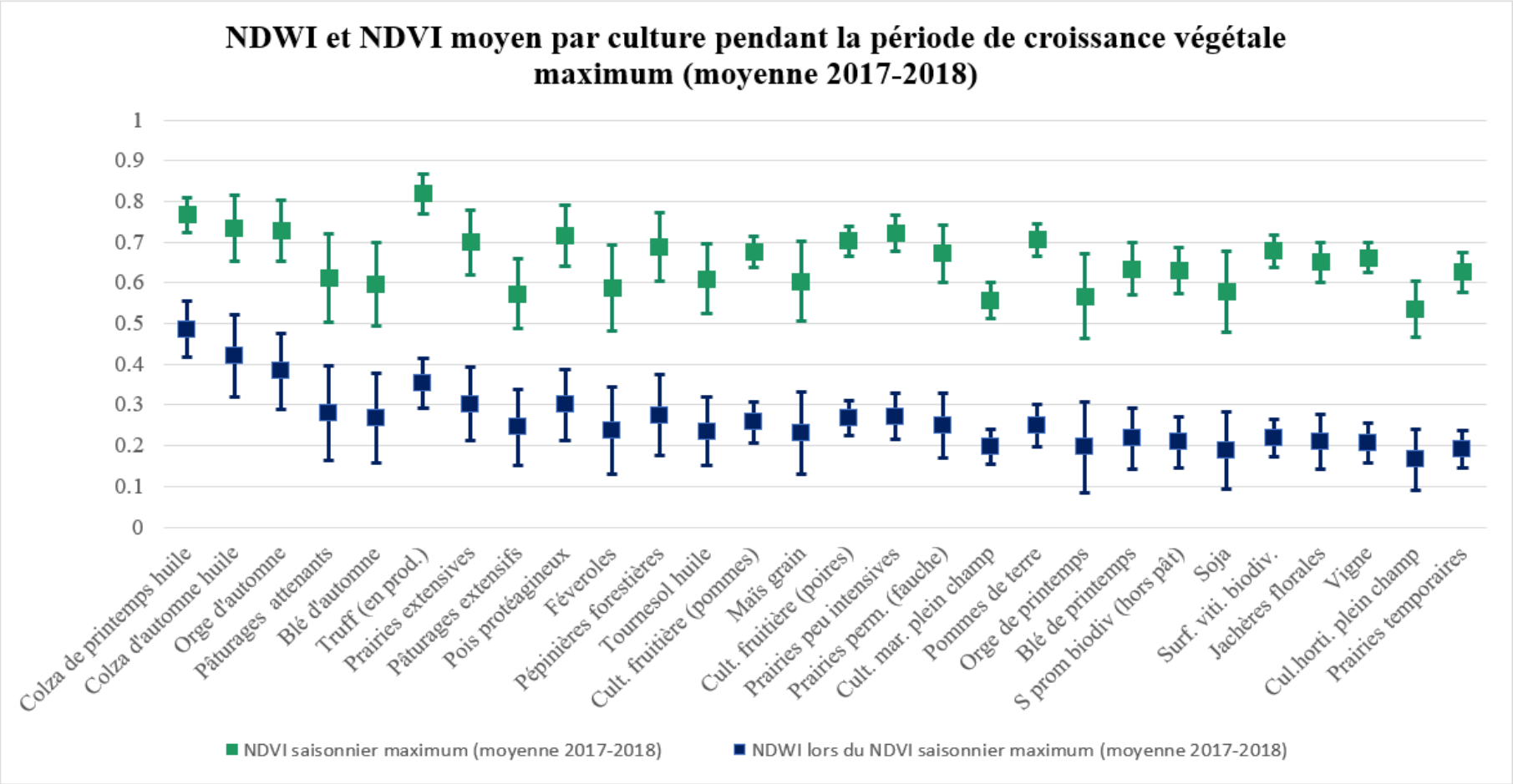


Figure 6: NDWI et NDVI pendant la saison de croissance végétale maximale dans le canton de Genève (moyenne 2017-2018).
Les extrémités des barres d'erreurs réfèrent à la première déviation standard (moyenne 2017-2018) du NDWI et NDVI de chaque culture respectivement.

En analysant une parcelle viticole aléatoire du Canton de Genève, l'évolution temporelle 1984-2019 des indices saisonniers NDVI et NDWI démontre leurs potentiels – en conjonction avec des relevés météorologiques - pour informer sur l'état de stress de la vigne lors de conditions météorologiques extrêmes : La Figure 7 illustre l'évolution temporelle de ces indices pendant le printemps. On remarque que l'indice NDVI baisse en 2010, qui était très changeant, avec une alternance entre temps chaud et frais pluvieux[3]. Cette chute du NDVI au printemps s'est renforcée en été (Figure 8), ce qui est expliqué par les épisodes de mildiou – confirmés par l'entretien avec le viticulteur de l'OCAN Florian Favre (Favre, 2019) - dont a souffert la vigne en 2010, favorisés par les conditions variables du printemps et de début Juin cette année-là. En effet, le mildiou prolifère en Juin dans des températures optimales comprises entre 20 et 25°C par temps humide[4].

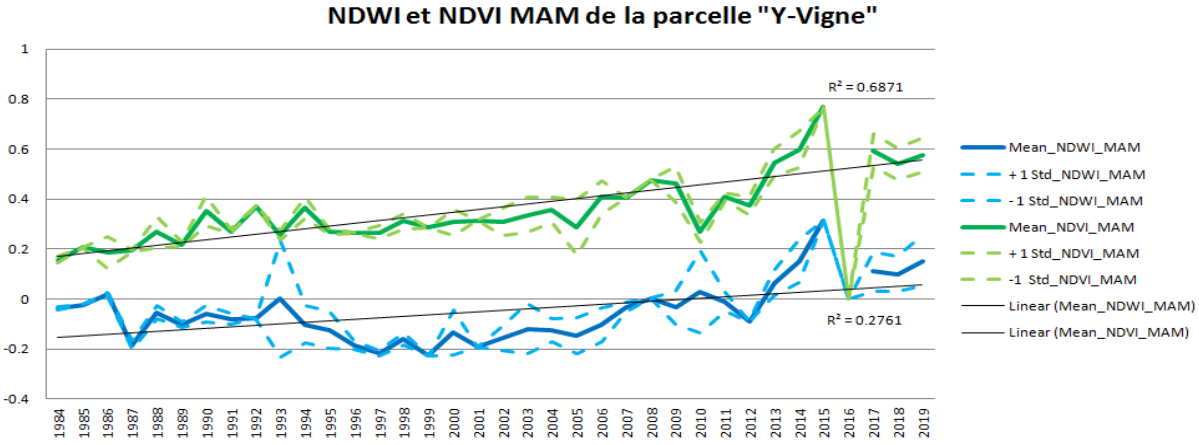


Figure 7: Evolution temporelle de l'indice d'humidité foliaire (NDWI) et de végétation (NDVI) d'une parcelle viticole genevoise au printemps

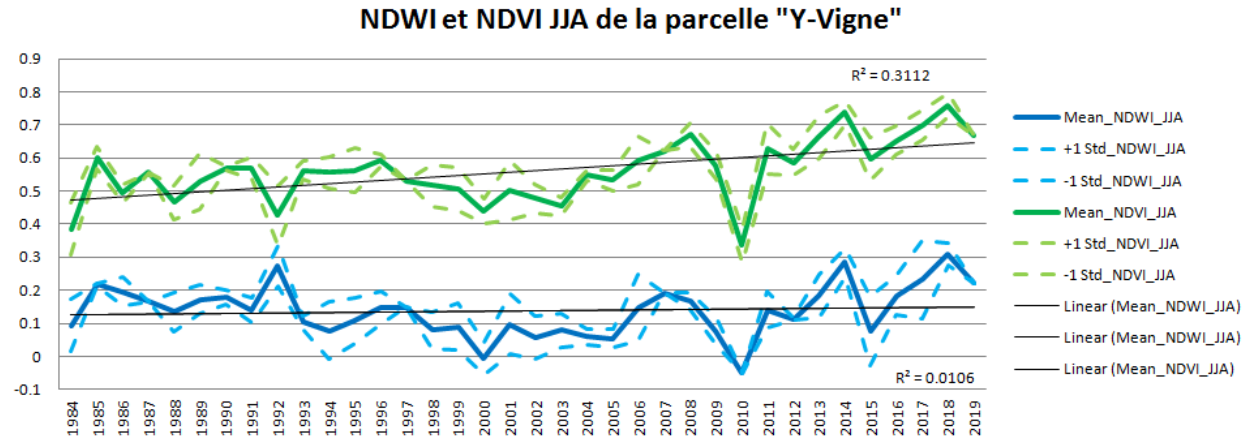


Figure 8: Evolution temporelle de l'indice d'humidité foliaire (NDWI) et de végétation (NDVI) d'une parcelle viticole genevoise en été

[3] RTS, 2010. Le temps a été très changeant en 2010 en Suisse. Disponible sur : https://www.rts.ch/info/sciences-tech/2829335-le-temps-a-ete-tres-changeant-en-2010-en-suisse.html

[4] BASF SE, 2019. Le mildiou : l'ennemi numéro 1 de la vigne. Disponible sur : https://www.agro.basf.fr/fr/cultures/vigne/maladies_de_la_vigne/mildiou_de_la_vigne/

A l'échelle du canton de Genève, l'activité photosynthétique et le contenu en eau des feuilles des parcelles culturales genevoises a augmenté pendant la période 1984-2019. Cependant, l'évolution de ces deux indices de santé de la végétation varie de manière substantielle en fonction de la saison, et en fonction du type de culture.

Comme l'illustre la Figure 9, l'activité photosynthétique au printemps a augmenté sur 40% des parcelles (fort probable à très probable), sur 21% (probable à fort probable) et sur 10% (incertain à probable)[5]. Au total, l'activité photosynthétique s'est intensifiée sur 70% des parcelles agricoles genevoises. A l'inverse, 22% des parcelles ne dénotent pas d'évolution significative et seuls 1.6% des parcelles ont enregistré une baisse d'activité photosynthétique[6]. L'analyse du contenu en eau des feuilles au printemps dépeint une évolution plus nuancée (Figure 10): sur 16% des parcelles le contenu en eau des feuilles a diminué, tandis que 51% des parcelles n'enregistrent pas d'évolution significative. La proportion des parcelles affichant un indice d'humidité foliaire plus élevé atteint 27%. Autrement dit, deux parcelles sur trois ont enregistré une stagnation ou une baisse de leur indice d'humidité foliaire au printemps, contrastant avec une tendance positive de la végétation pour la majorité des cultures à cette même période.

En été, la tendance positive de l'activité photosynthétique est plus prononcée (Figure 11) : les cultures de 55% des parcelles ont expérimenté une augmentation d'activité photosynthétique (fort probable à très probable), 16% (probable à fort probable) et 8.1% (incertain à probable), totalisant 79.1% de parcelles témoignant d'une végétation plus vigoureuse en 36 ans. A rebours, les cultures dont l'intensité de la végétation a diminué ou stagné ne représentent que 14% des parcelles. Contrairement à la tendance printanière, en été l'indice d'humidité foliaire a augmenté sur une partie conséquente des parcelles culturales, soit sur 41%, et baissé sur 8.3% (Figure 12).

La comparaison des tendances saisonnières des deux indices permet d'isoler une évolution distincte au printemps et en été. En effet, l'indice d'humidité foliaire a décru de manière plus forte au printemps qu'en été, bien que l'activité photosynthétique ait augmentée de manière similaire durant les deux saisons.

[5] Echelle : Probabilité [entre 95 et 99% - fort probable à très probable ; entre 90 et 95% - probable à fort probable ; inférieur à 90% - incertain à probable]

[6] La somme des % ne totalise pas 100% car certaines parcelles étaient plus petites que 0.1 ha et n'étaient donc pas incluses dans un pixel.

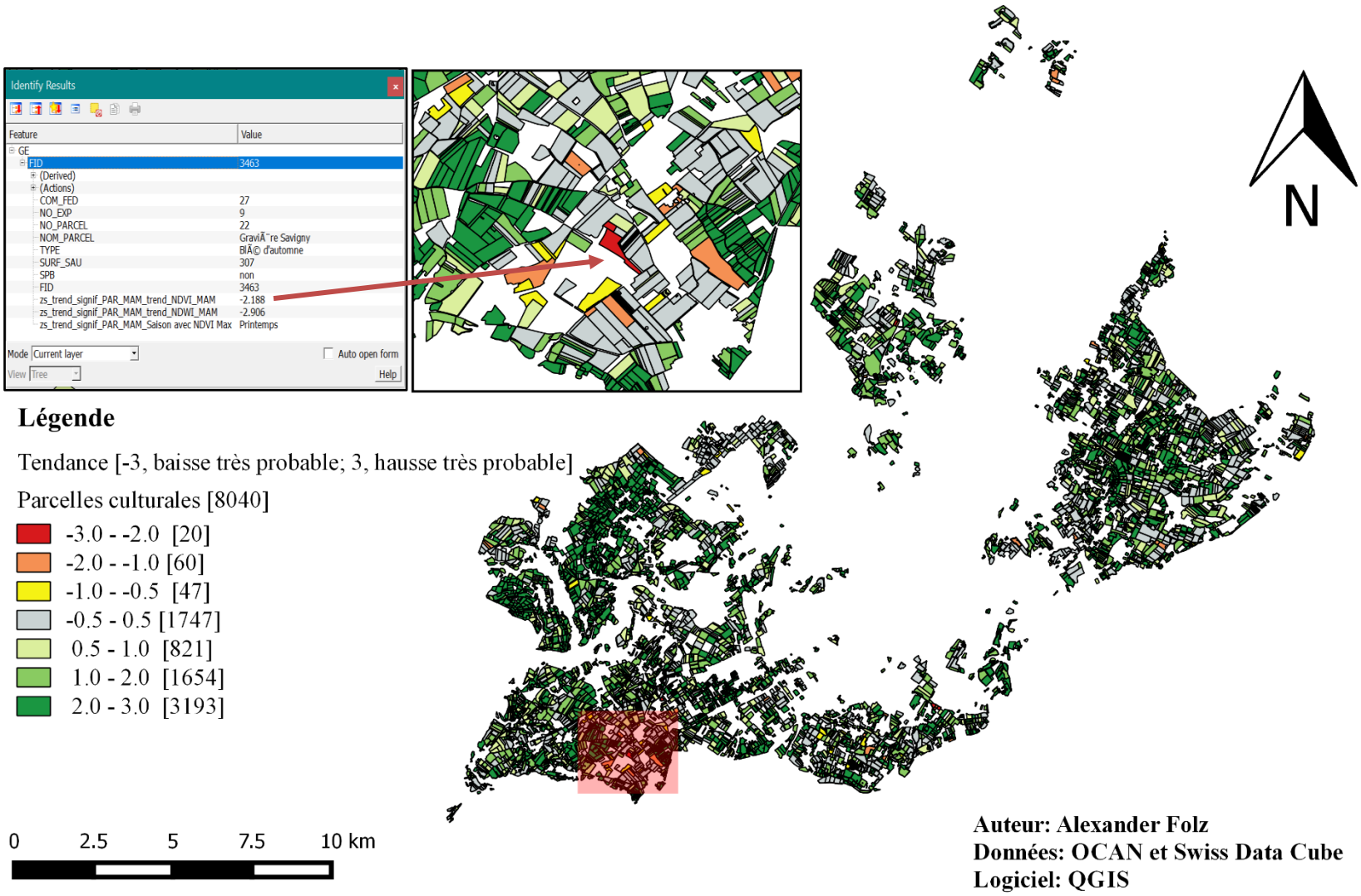


Figure 9: Tendance 1984-2019 de l'indice de végétation normalisé (NDVI) des parcelles culturales genevoises au printemps

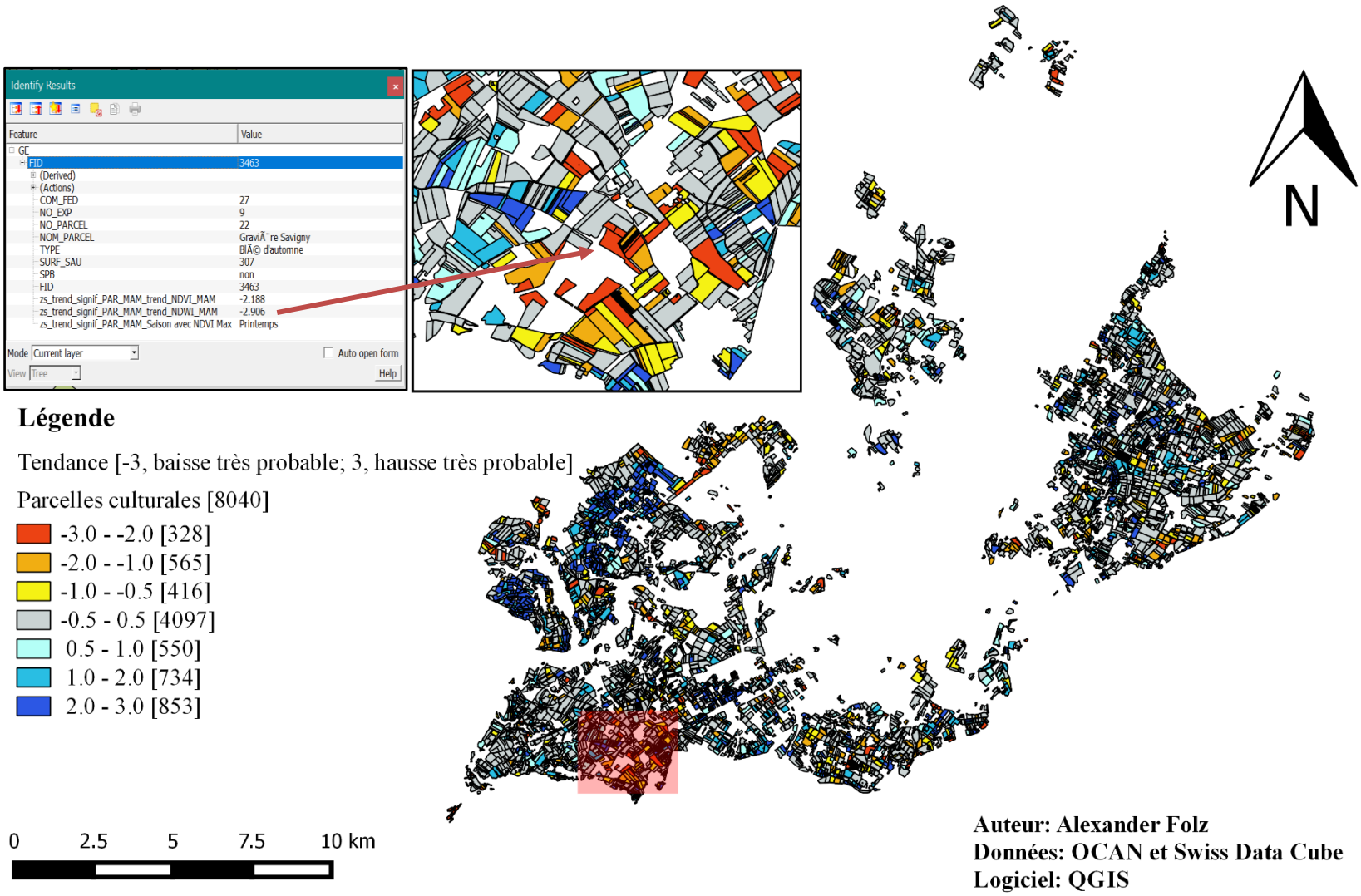


Figure 10: Tendance 1984-2019 de l'indice d'humidité foliaire normalisé (NDWI) des parcelles culturales genevoises au printemps

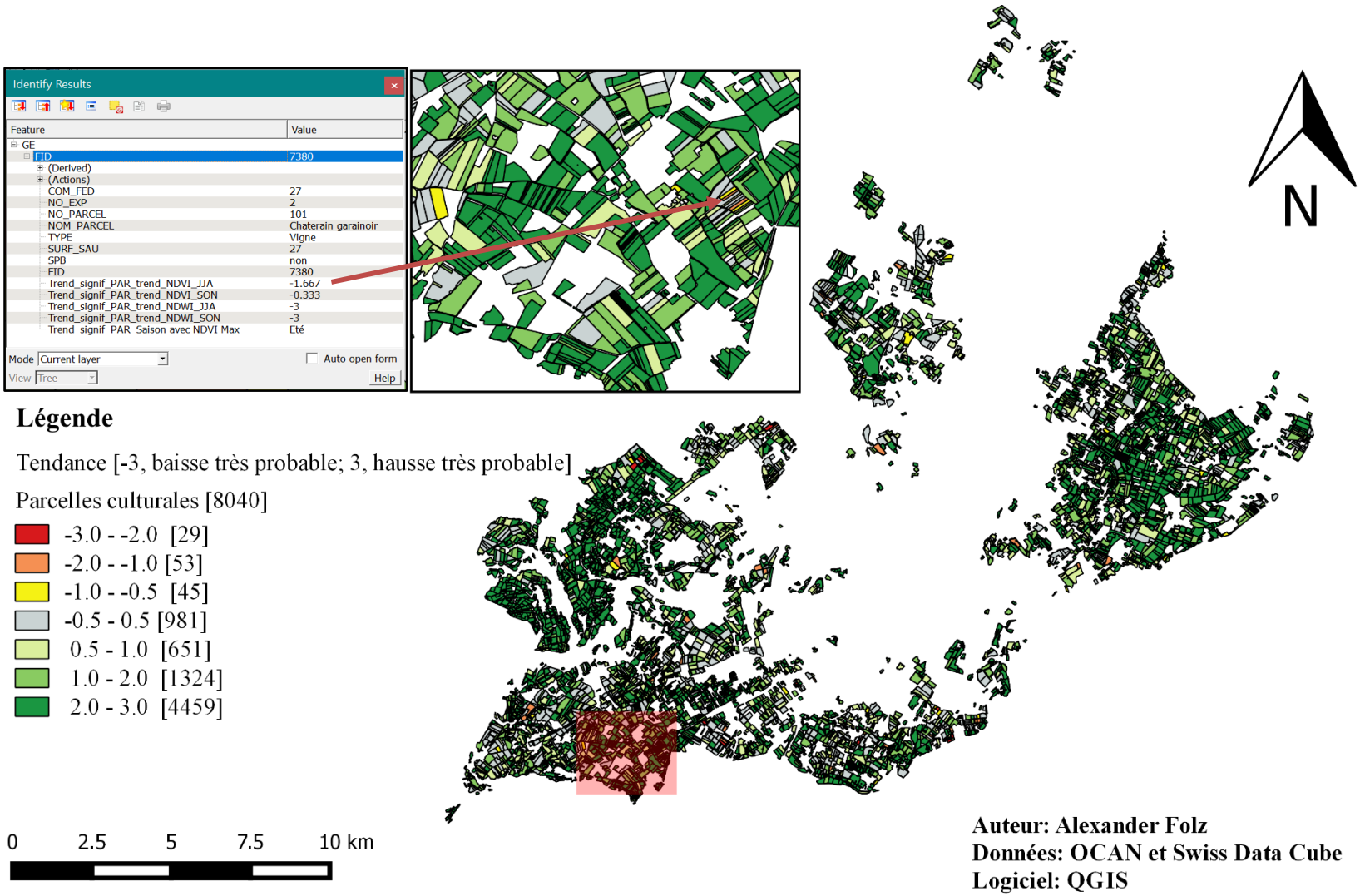


Figure 11: Tendance 1984-2019 de l'indice de végétation normalisé (NDVI) des parcelles culturales genevoises en été

Tendance 1984-2019 de l'indice d'humidité foliaire normalisé (NDWI) des parcelles culturales genevoises en été

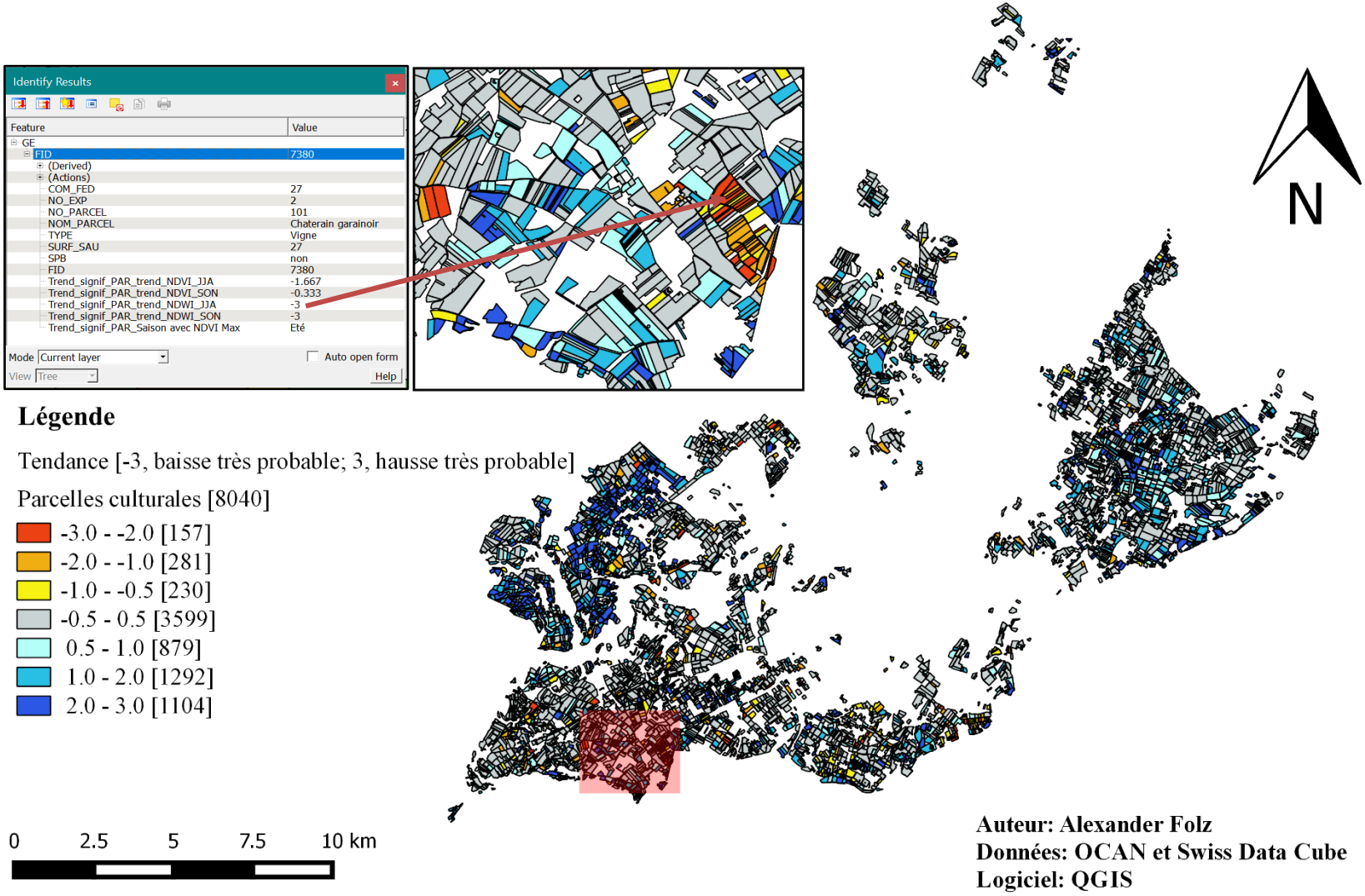


Figure 12: Tendance 1984-2019 de l'indice d'humidité foliaire normalisé (NDWI) des parcelles culturales genevoises en été

Afin d'isoler les cultures en fonction de la tendance de leurs indices, les parcelles culturales ont été séparées en cinq groupes, regroupant 90.7% des parcelles.

1. Les parcelles culturales affichant une tendance de l'indice NDWI négative et une tendance de l'indice NDVI positive [$Tendance_{NDWI} < -0.5$ et $Tendance_{NDVI} > 0.5$]

2. Les parcelles culturales affichant une tendance de l'indice NDWI neutre et une tendance de l'indice NDVI positive [$-0.5 < Tendance_{NDWI} < 0.5$ et $Tendance_{NDVI} > 0.5$]

3. Les parcelles culturales affichant une tendance de l'indice NDWI positive et une tendance de l'indice NDVI positive [$Tendance_{NDWI} > 0.5$ et $Tendance_{NDVI} > 0.5$]

4. Les parcelles culturales affichant une tendance de l'indice NDWI négative et une tendance de l'indice NDVI neutre [$Tendance_{NDWI} < -0.5$ et $-0.5 < Tendance_{NDVI} < 0.5$]

5. Les parcelles culturales affichant une tendance de l'indice NDWI neutre et une tendance de l'indice NDVI neutre [$-0.5 < Tendance_{NDWI} < 0.5$ et $-0.5 < Tendance_{NDVI} < 0.5$]

Comme illustré par la Figure 13**,** malgré une augmentation de l'activité photosynthétique, l'indice d'humidité foliaire au printemps a baissé sur 4.2% des parcelles culturales. La majorité des parcelles témoignant de cette baisse sont des prairies extensives. Par ailleurs, l'indice d'humidité foliaire est resté constant sur 38% des parcelles en dépit d'une augmentation du NDVI (Figure 14**)**: ces parcelles sont en majorité des prairies, cultures de blé d'automne et de vigne.

En somme, 42% des cultures dont l'activité photosynthétique a cru, en majorité les prairies et dans une moindre mesure, les pâturages, la vigne et le blé d'automne, subissent une stagnation ou diminution du contenu en eau de leur feuilles. En revanche, sur 27% des parcelles, l'indice NDWI et NDVI a augmenté : un grand nombre de parcelles bénéficiant de cette hausse sont celles cultivant la vigne. Les prairies et cultures telles que le blé d'automne, le colza à huile et l'orge d'automne témoignent également de cette hausse (Figure 15).

Enfin, l'influence limitante d'une baisse ou stagnation de l'humidité foliaire sur la croissance de l'activité photosynthétique se matérialise pour 22% des parcelles culturales : 9% dénotent une stagnation du NDVI conjugué à diminution du NDWI (Figure 16) - la plupart sont des cultures de blé d'automne, des prairies extensives et des cultures maraîchères - ;11% témoignent d'une stagnation des deux indicateurs (Figure 17) - représentées par des cultures de blé d'automne, des prairies extensives et des cultures de colza d'automne à huile.

Cette différence de tendance entre le NDVI et le NDWI devient plus marginale pendant l'été . A cette période, les indices NDVI et NDWI augmentent ensemble sur 41% des parcelles, soit sur un grand nombre de prairies et vignes (Figure 18).

Distribution et nombre de parcelles culturales affichant une tendance négative de l'indice d'humidité foliaire (< - 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) au printemps

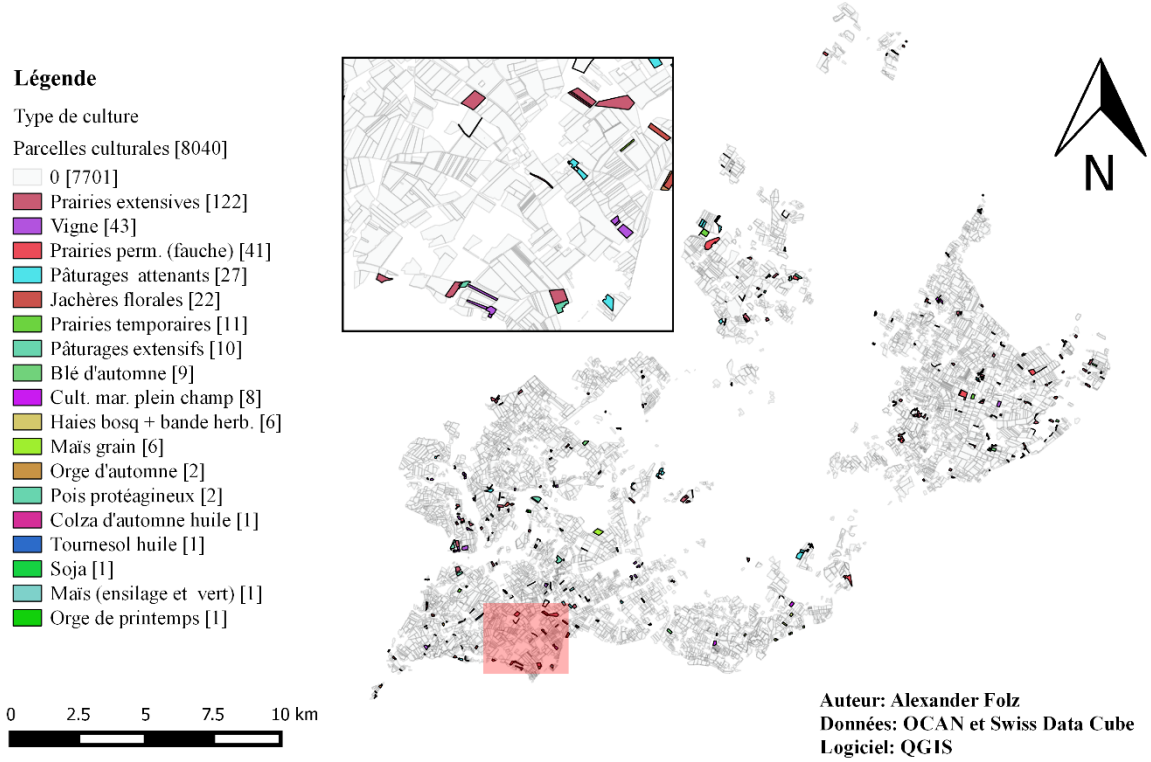


Figure 13: Distribution et nombre de parcelles culturales affichant une tendance négative de l'indice d'humidité foliaire (< -0.5) et une tendance positive de l'activité photosynthétique (> 0.5) au printemps

Distribution et nombre de parcelles culturales affichant une tendance neutre de l'indice d'humidité foliaire (-0.5 < x < 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) au printemps

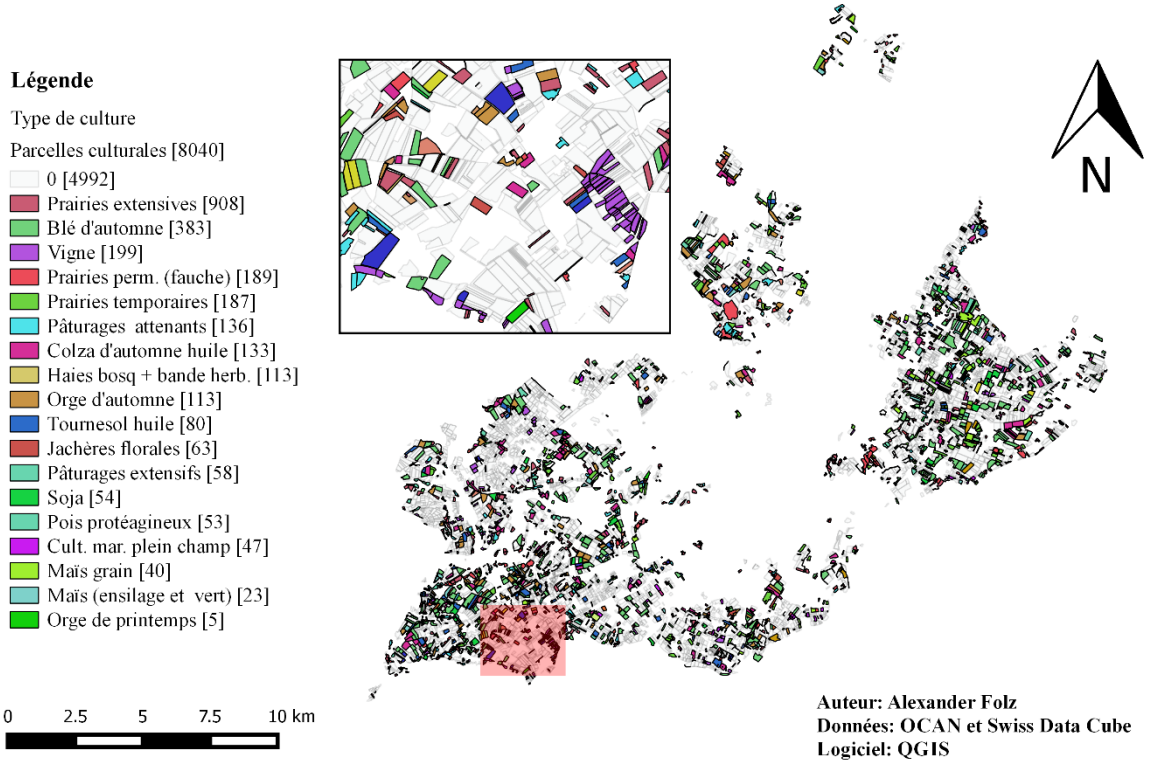


Figure 14: Distribution et nombre de parcelles culturales affichant une tendance neutre de l'indice d'humidité foliaire (-0.5 < x < 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) au printemps

Distribution et nombre de parcelles culturales affichant une tendance positive de l'indice d'humidité foliaire (> 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) au printemps

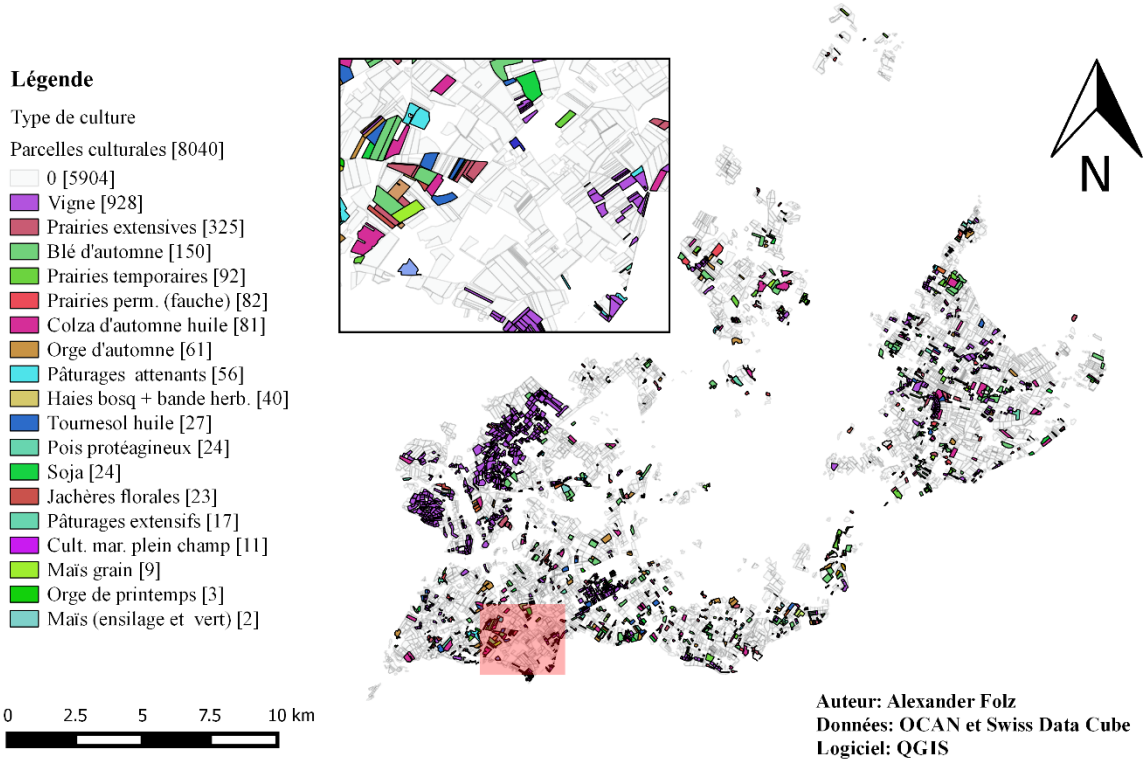


Figure 15: Distribution et nombre de parcelles culturales affichant une tendance positive de l'indice d'humidité foliaire (> 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) au printemps

Distribution et nombre de parcelles culturales affichant une tendance négative de l'indice d'humidité foliaire (x < -0.5) et une tendance neutre de l'activité photosynthétique (-0.5 < x < 0.5) au printemps

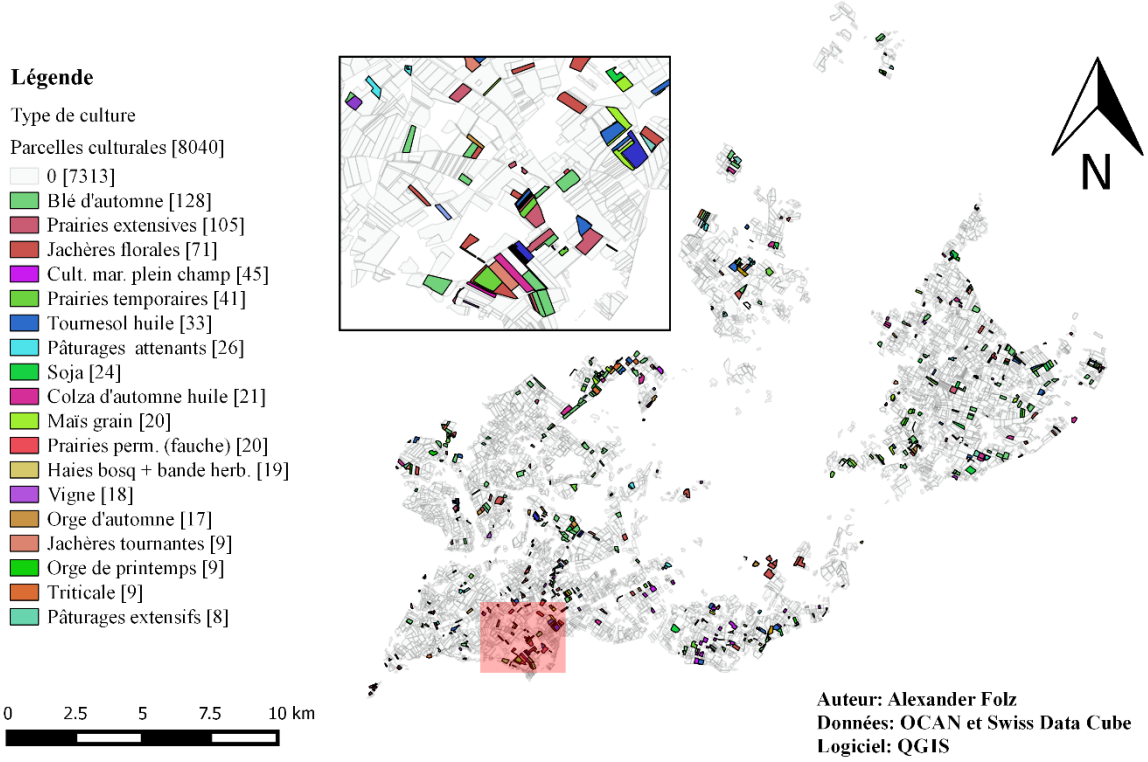


Figure 16: Distribution et nombre de parcelles culturales affichant une tendance négative de l'indice d'humidité foliaire (x < -0.5) et une tendance neutre de l'activité photosynthétique (-0.5 < x < 0.5) au printemps

Distribution et nombre de parcelles culturales affichant une tendance neutre de l'indice d'humidité foliaire (-0.5 < x < 0.5) et une tendance neutre de l'activité photosynthétique (-0.5 < x < 0.5) au printemps

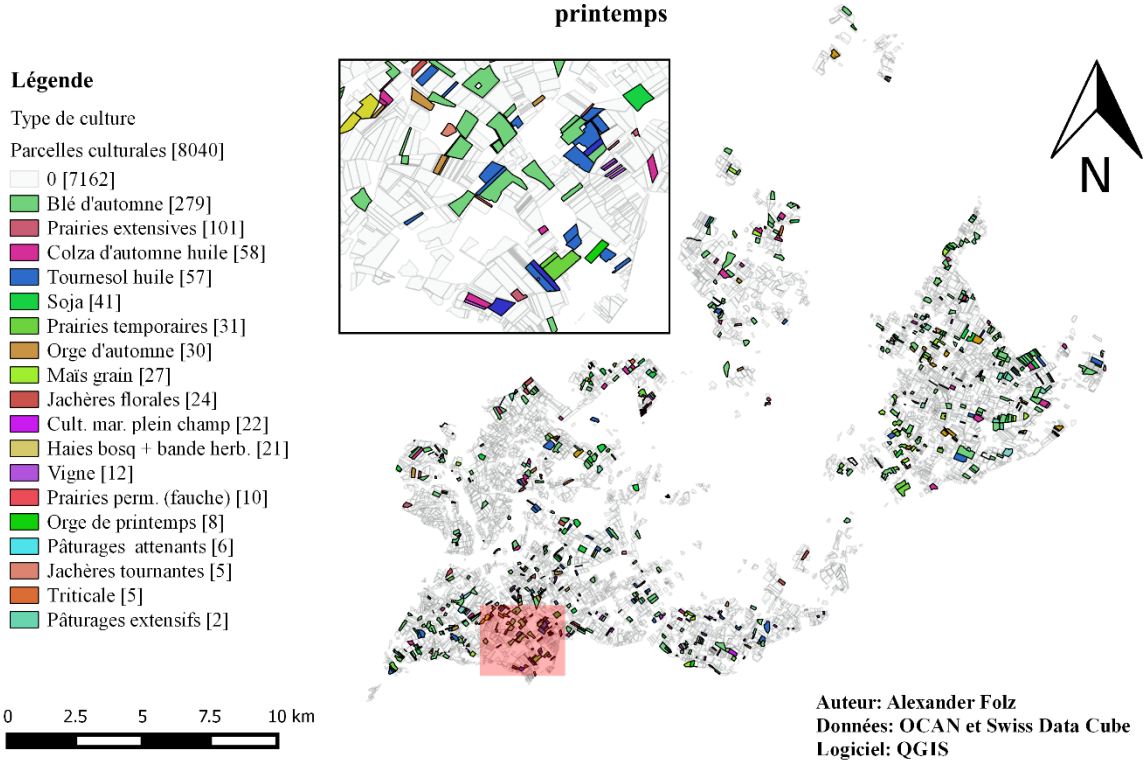


Figure 17: Distribution et nombre de parcelles culturales affichant une tendance neutre de l'indice d'humidité foliaire (-0.5 < x < 0.5) et une tendance neutre de l'activité photosynthétique (-0.5 < x < 0.5) au printemps

Distribution et nombre de parcelles culturales affichant une tendance positive de l'indice d'humidité foliaire (> 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) en été

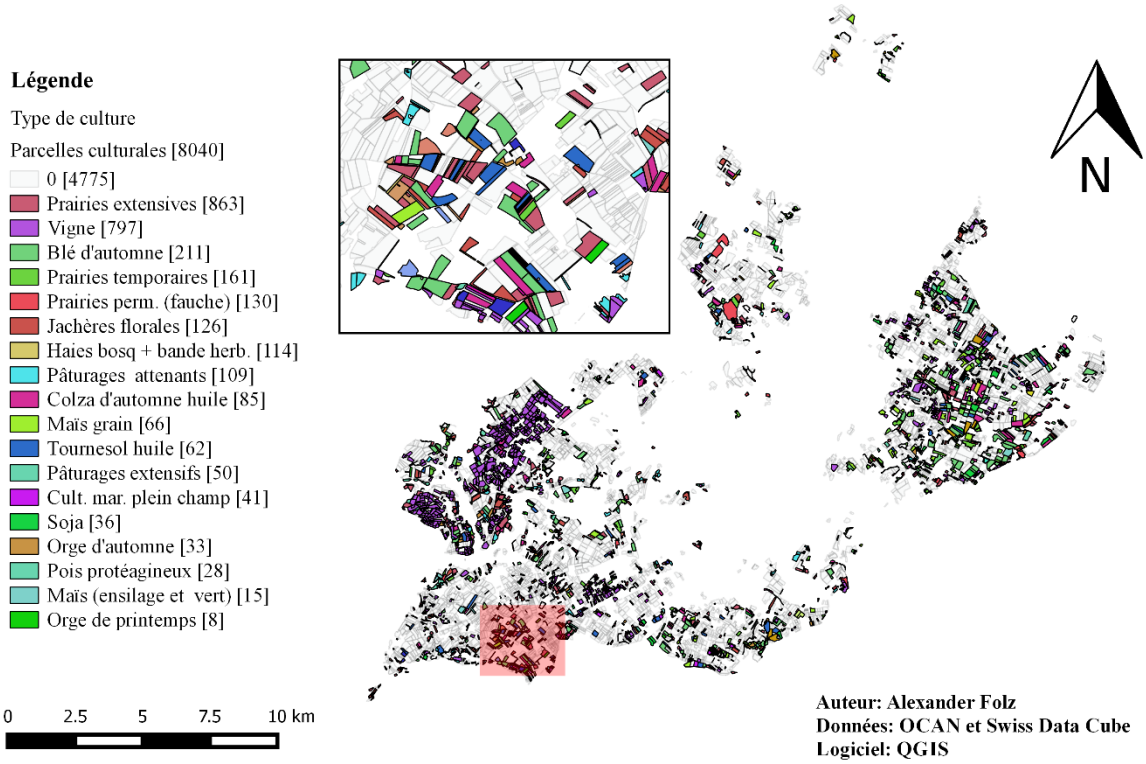


Figure 18: Distribution et nombre de parcelles culturales affichant une tendance positive de l'indice d'humidité foliaire (> 0.5) et une tendance positive de l'activité photosynthétique (> 0.5) en été

Corrigeant le biais induit par le fait que les cultures soient à nu hors de leur période de croissance, la Figure 19 détermine l'évolution temporelle de l'activité photosynthétique et du contenu hydrique de la plante lors de son maximum saisonnier de végétation.

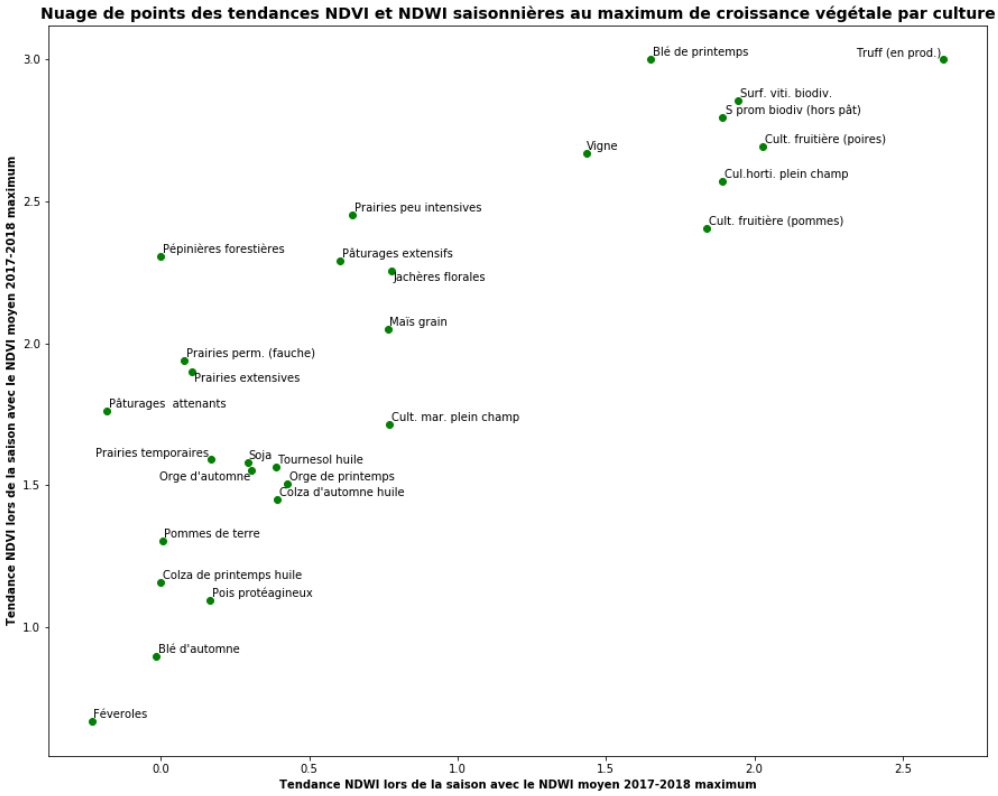


Figure 19: Nuage de points des tendances NDVI et NDWI saisonnières au maximum de croissance végétale par culture

On constate qu'à l'échelle des cultures, les tendances négatives et neutres de certaines parcelles sont compensées par les tendances positives d'autres parcelles, illustrant ainsi la tendance générale d'une culture. Il est ainsi possible de différencier 3 groupes de cultures selon leurs trajectoires de tendance:

- Les cultures arboricoles, horticoles, la vigne et le blé de printemps, dont le NDVI et le NDWI ont augmenté de manière significative en 36 ans (Figure 18**)**
- Les prairies, pâturages, autres espaces peu intensifs et cultures maraîchères témoignant d'une importante augmentation du NDVI et d'une hausse du NDWI plus modeste, voire nulle (Figure 13, Figure 14, Figure 15, Figure 16, Figure 17)
- Les grandes cultures, dont l'activité photosynthétique n'a que modestement augmenté et l'humidité foliaire est resté proche de constant (Figure 14, Figure 16, Figure 17).

Cette variation de tendance entre les cultures lors de leur croissance végétale maximum permet de confirmer les tendances observées à l'échelle des parcelles culturales.

Cependant, il est important de considérer que ces tendances en période de croissance végétale maximum présentent certains biais:

- Les cultures occupant une faible surface agricole utile présentent potentiellement des valeurs plus extrêmes que les cultures dont la tendance a été agrégée sur un plus grand nombre de parcelles. C'est le cas notamment du blé de printemps, qui n'est représenté que par 4 parcelles sur 8040, les cultures de poires (11 parcelles sur 8040) ou les pommes de terre (21 parcelles sur 8040).

- Ces tendances illustrent l'évolution différenciée des indices NDVI et NDWI au maximum de croissance végétale et négligent la variation de tendance durant les autres saisons.

V. Discussion

Cette étude a montré le potentiel du Swiss Data Cube pour créer des indicateurs de suivi environnemental des parcelles culturales genevoises. La comparaison des indices NDVI et NDWI apparaît utile pour déterminer la sensibilité des cultures à un déficit hydrique.

En particulier, le ratio entre le NDVI et le NDWI (moyenne 2017-2018) de la vigne en été a montré que la croissance de celle-ci dépend peu de sa disponibilité en eau. Cependant, ce ratio capture aussi l'effet d'autres facteurs sur la croissance de l'activité photosynthétique tels que la température, le CO_2, l'azote ou le phosphore. En conséquence, des études ultérieures seraient nécessaires afin d'expliquer la susceptibilité de chaque culture aux différents facteurs limitants.

En outre, l'analyse temporelle du NDVI et du NDWI saisonnier semble capturer les influences de facteurs externes sur les cultures (ex. situations météorologiques exceptionnelles ou épisodes de maladie foliaire etc.). Par conséquent, ces indicateurs ont montré leur utilité - en combinaison avec des relevés météorologiques - pour aider les exploitants agricoles à prévenir des potentielles pertes de récolte.

L'analyse de la tendance saisonnière 1984-2019 a montré qu'il existait des groupes de culture avec une tendance caractéristique :

- Les cultures arboricoles, horticoles, la vigne et le blé de printemps qui dénotent une tendance positive du NDVI et du NDWI en 36 ans.
- Les prairies, pâturages, autres espaces peu intensifs et cultures maraîchères témoignant d'une importante augmentation du NDVI et d'une hausse du NDWI plus modeste, voire nulle.
- Les grandes cultures, dont l'activité photosynthétique n'a que modestement augmenté et l'humidité foliaire est resté proche de constant.

L'augmentation des indices NDVI pour ces trois groupes de cultures est liée à une augmentation de la température et à une fertilisation du CO_2 (Zhu et al., 2016). Selon (Myneni et al., 1997), ces augmentations sont liées à un allongement de la saison de croissance débutant jusqu'à 7 jours plus tôt dans l'année (Keeling et al., 1996 cited: in Myneni et al., 1997).

Les tendances modestes, voir nulles, de l'indice d'humidité foliaire au printemps, résultent potentiellement d'une baisse de la surface neigeuse au printemps ces dernières décennies (Groisman et al., 1994) . Cette baisse a contribué à l'augmentation de la température de surface terrestre dans l'hémisphère nord par l'effet d'une diminution de l'albédo (ibid., 1994). Résultant d'une augmentation de la température de surface, les tendances NDVI et NDWI reflètent des traits physiologiques et phénologiques des cultures distincts : la vigne témoigne d'une résistance à la sécheresse plus élevée, tandis que les prairies et les grandes cultures sont plus sensibles à un déficit hydrique (Figure 6, Figure 19). Ces différences de tendances sont notamment expliquées par la période de croissance distincte des cultures : une culture d'automne est atteinte par une augmentation de la température plus importante qu'une culture d'été.

Malgré le potentiel du Swiss Data Cube pour informer sur le statut environnemental des parcelles culturales genevoises, cette étude présente certaines limitations :

- Les données de surface agricole utile par type de culture utilisées dans le cadre de l'analyse temporelle réfèrent à l'année 2017. Par conséquent, si une parcelle a été allouée à la production d'une autre culture principale pendant l'intervalle 1984-2019, les résultats ne sont pas applicables pour cette parcelle.

- L'indice NDVI est robuste pour une analyse avec un fort contraste entre sol nu, végétation et surfaces d'eau, tel que le canton de Genève. Cependant, cet indice est brouillé par la haute réflexion du sol nu dans le spectre infrarouge (Tucker, 1979), ainsi que via la diffusion atmosphérique du rayonnement solaire par les aérosols dans le spectre rouge (Kaufman and Tanré, 1992 cited in Gao, 1997).

  Ainsi, dans le cas de sols nus très secs, comme dans les zones arides, l'indice NDVI ne serait pas recommandé car la différence de contraste du NDVI entre le sol et la végétation est très faible. Par conséquent, pour l'analyse de ce type de géographie, l'utilisation d'un indice de végétation nuançant la haute réflexion du sol dans l'infrarouge proche (ex. MSAVI2) est recommandée. (Tucker, 1979). Pour supprimer l'influence de la diffusion atmosphérique – ex. nuages - dans le NDVI, d'autre indicateurs tel que l'indice ARVI sont recommandés (Kaufman and Tanré, 1992 cited in Gao, 1997). Par ailleurs, dans le cas de forêts denses, l'indice NDVI sature car la végétation dense réfléchit de manière indifférenciée dans le spectre infrarouge. De ce fait, l'indice EVI est employé pour éliminer tant l'influence de la diffusion atmosphérique que l'effet de saturation du NDVI.

  Dans le cadre de futures recherches visant à produire d'autres indicateurs utiles à l'exploitant agricole, en plus de NDVI et NDWI améliorés, le calcul d'un autre indicateur - NPCRI (Normalized Pigment Chlorophyll Ratio Index) - permettant d'approximer le ratio caroténoïde-chlorophylle des plantes, et ainsi, l'efficacité de leur système photosynthétique, serait recommandé.

- Selon (Gao, 1997), la bande MODIS 5 centrée autour de 1.24 µm serait plus adaptée pour isoler le contenu en eau liquide des feuilles que la bande SWIR1 présente dans les capteurs Landsat et Sentinel 2 car l'utilisation de la première permettrait de supprimer la variation de réflectance liée au mésophylle spongieux. Néanmoins, en raison du ratio signal-bruit faible de la bande MODIS 5, l'indice NDWI est calculé en remplaçant la bande MODIS 5 par la bande MODIS 6, comme c'est le cas pour le produit NDWI du centre européen de recherche (Joint Research Center – JRC).

- Les moyennes saisonnières de NDVI et NDWI pour chaque année ont été calculées en se basant sur ~6 observations par saison. Par conséquent, l'exploitation statistique de ces données est limitée. Pour remédier à cette limitation, il serait utile de générer les

moyennes mensuelles et saisonnières des indices NDVI et NDWI à l'aide des capteurs des satellites Sentinel 2. Les capteurs de Sentinel 2 génèrent ~36 images satellites par saison, ce qui augmenterait la robustesse statistique des moyennes générées.

- L'utilisation des données temporelles à des fins d'observation en temps réel n'est actuellement pas possible car il existe un délai d'~1 mois entre la prise de vue par le capteur satellite Landsat et la disponibilité de l'image pour analyse au format (ARD).

- Les résultats issus de cette étude sont exploratoires et nécessitent la validation d' études statistiques ultérieures.

VI. Conclusion

Dans le contexte d'une agriculture moderne vulnérable à diverses instabilités naturelles et sociales, cette étude a montré le potentiel du Swiss Data Cube pour créer des indicateurs saisonniers et mensuels de suivi environnemental des parcelles culturales genevoises.

Le ratio de l'indice NDVI et NDWI d'une culture permet d'informer sur sa susceptibilité à un stress hydrique. L'analyse de la tendance saisonnière 1984-2019 a montré qu'il existait des groupes de culture avec une tendance caractéristique : les cultures arboricoles, horticoles, la vigne et le blé de printemps qui dénotent une tendance positive du NDVI et du NDWI en 36 ans ; les prairies, pâturages, autres espaces peu intensifs et cultures maraîchères témoignant d'une importante augmentation du NDVI et d'une hausse du NDWI plus modeste, voire nulle ; les grandes cultures, dont l'activité photosynthétique n'a que modestement augmenté et l'humidité foliaire est resté proche de constant.

Résultant d'une augmentation de la température de surface, les tendances NDVI et NDWI reflètent des traits physiologiques et phénologiques des cultures distincts: la vigne témoigne d'une résistance à la sécheresse plus élevée, tandis que les prairies et les grandes cultures sont plus sensibles à un déficit hydrique. Ces différences de tendances sont notamment expliquées par la période de croissance distincte des cultures : une culture d'automne est atteinte par une augmentation de la température plus importante qu'une culture d'été.

VII. Références

Articles scientifiques

Gao, B. C., 1996. NDWI—A normalized difference water index for remote sensing of vegetation liquid water from space. *Remote sensing of environment, Vol. 58(3) : 257-266.*

Groisman, P. Y., 1994. Observed impact of snow cover on the heat balance and the rise of continental spring temperatures. *Science, Vol. 263(5144) :198-200.*

Gu, Y. et al., 2007. A five-year analysis of MODIS NDVI and NDWI for grassland drought assessment over the central Great Plains of the United States. *Geophysical Research Letters, Vol. 34(6).*

Hoekstra, A. Y., & Mekonnen, M. M. 2012. The water footprint of humanity. *Proceedings of the national academy of sciences, Vol. 109(9) : 3232-3237.*

Lillesaeter O., 1982. Spectral reflectance of partly transmitting leaves: Laboratory measurements and mathematical modeling. *Remote Sensing of Environment, Vol.12(3): 247-254.* doi:10.1016/0034-4257(82)90057-8

Mekonnen, M. M., & Hoekstra, A. Y., 2011. The green, blue and grey water footprint of crops and derived crop products. *Hydrology and Earth System Sciences, Vol. 15(5): 1577-1600.* doi:10.5194/hess-15-1577-2011

Mekonnen, M. M. & Hoekstra, Arjen Y, 2012. A global assessment of the water footprint of farm animal products. *Ecosystems, Vol. 15(3): 401-415.* doi: 10.1007/s10021-011-9517-8

Myneni R.B., et al., 1997. Increased plant growth in the northern high latitudes from 1981 to 1991. *Nature, Vol. 386(6626): 698.*

Tucker, C. J., 1979. Red and Photographic Infrared linear Combinations for Monitoring Vegetation. *Remote Sensing Of Environment, Vol. 8(2): 127-150.*

Xue J. & Su B., 2017. Significant remote sensing vegetation indices: A review of developments and applications. *Journal of Sensors, Vol. 2017.*

Yuan W., et al., 2019. Increased atmospheric vapor pressure deficit reduces global vegetation growth. *Science advances, Vol. 5(8): Eaax1396.*

Zhu, Z. et al., 2016. Greening of the Earth and its drivers. *Nature climate change, Vol. 6(8): 791.*

Entretiens

Favre, F., 2019. Entretien avec le viticulteur de l'OCAN Florian Favre.

Fiches techniques

JRC, 2011. NDWI: Normalized Difference Water Index. Version 1. Disponible sur: http://edo.jrc.ec.europa.eu/documents/factsheets/factsheet_ndwi.pdf

Revues scientifiques

Bretscher et al., 2014. Emissions de gaz à effet de serre dans l'agriculture et la filière alimentaire en Suisse. *Recherche Agronomique Suisse, Vol. 5 (11–12): 458–465*.

Rapports

Arneth A., et al. , 2019. IPCC Special Report on Climate Change, Desertification, Land Degradation, Sustainable Land Management, Food Security, and Greenhouse gas fluxes in Terrestrial Ecosystems. Summary for Policymakers. Approved Draft.

FAO, 2009. Women and Rural Employment: Fighting Poverty by Redefining Gender Roles. Disponible sur : http://www.fao.org/tempref/docrep/fao/012/ak485e/ak485e00.pdf

Grain, 2014. Hungry for Land. Small farmers feed the world with less than a quarter of all farmland. Disponible sur : https://www.grain.org/article/entries/4929-hungry-for-land-small-farmers-feed-the-world-with-less-than-a-quarter-of-all-farmland#sdfootnote37sym

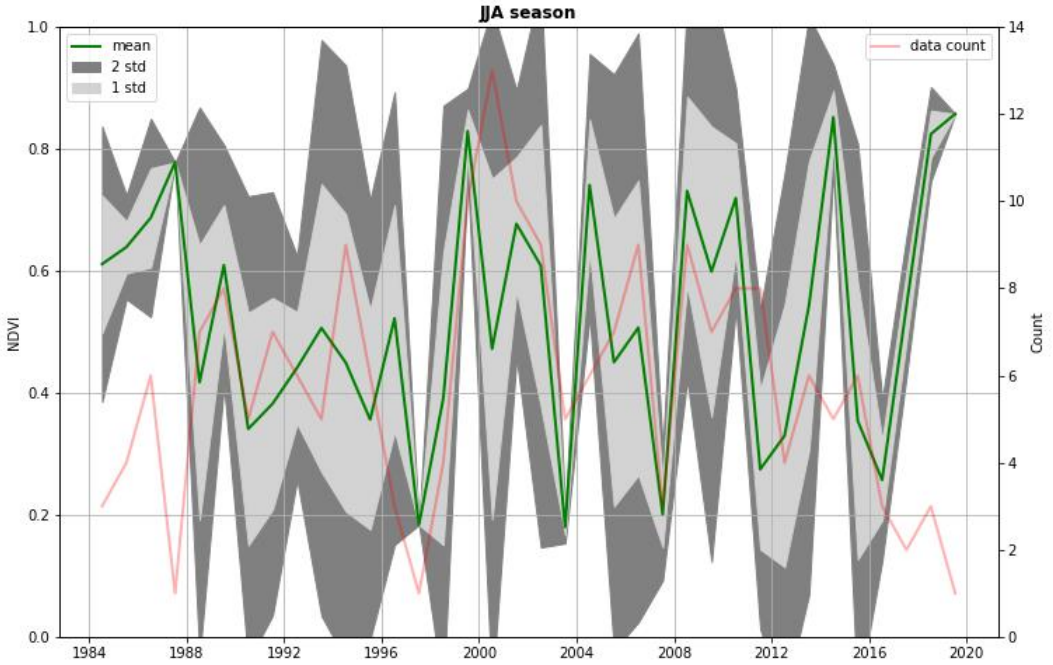
Oxfam, 2018. Ripe for change. Ending human suffering in supermarket supply chains. Disponible sur : https://www.oxfam.de/system/files/1._studie_ripe_for_change_englische_originalfassung.pdf

Livres

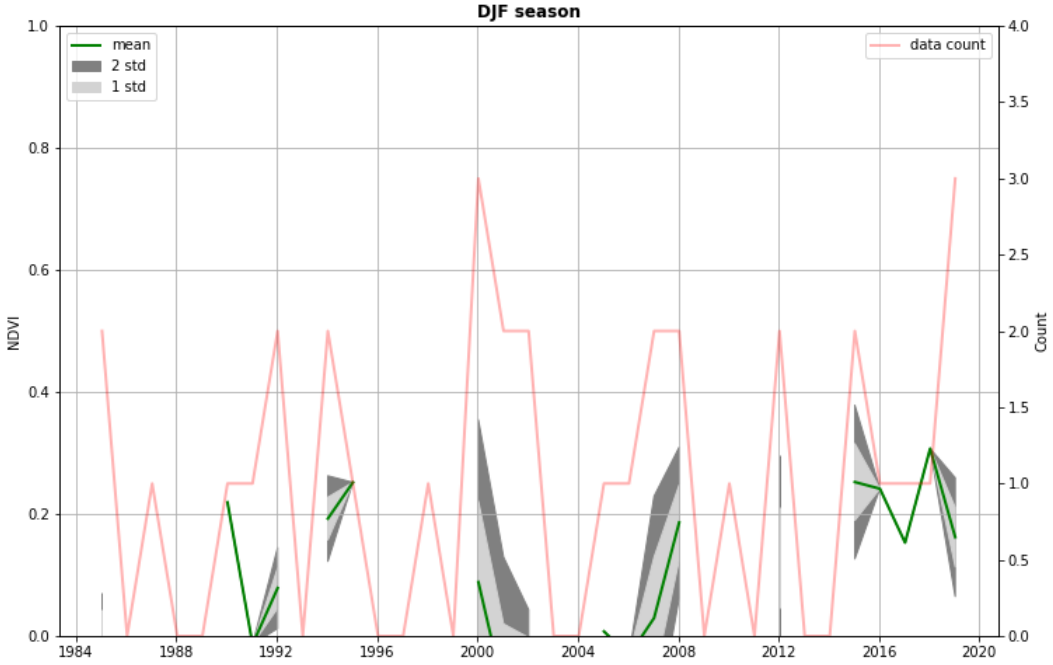
Hipel K. W. et al., 1994 . Time series modelling of water resources and environmental systems. Elsevier. Vol. 45. ISBN : 0-444-89270-2

Steinfeld H. et al., 2006. Livestock's long shadow: environmental issues and options. Food and Agricultural Organization of the United Nations. Disponible sur: http://www.fao.org/3/a0701e/a0701e00.htm

VIII. Annexes



Annexe 1: Moyenne, première et seconde déviation standard du NDVI et nombre d'images satellites utilisées dans le calcul du NDVI d'un pixel (culture) pour l'été



Annexe 2 : Moyenne, première et seconde déviation standard du NDVI et nombre d'images satellites utilisées dans le calcul du NDVI d'un pixel (zone urbaine) pour l'hiver

Annexe 3: Scripts Jupyter Notebook

Seasonal statistics (mean, standard deviation) generator

In [ ]:
```
# Import dependencies

import os
import sys
import shutil
import calendar

import xarray as xr
import numpy as np

from datetime import date, datetime
from multiprocessing import Pool, Lock, Manager
from IPython.display import display, Javascript

import datacube
dc = datacube.Datacube()

from utils.data_cube_utilities.dc_chunker import create_geographic_chunks, combine_geographic_chunks

from swiss_utils.data_cube_utilities.sdc_utilities import load_multi_clean, printandlog

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
# Configuration section

# Make sure to change the working directories ('/.../')
# of ndvi_season and ndwi_season as they will respectively host the created resulting seasonal netdcf files

# Change the logname ('') to be created and the user mail to which to logfile will be delivered

products = ["ls8_lasrc_swiss", "ls7_ledaps_swiss", "ls5_ledaps_swiss"]

measurements=['red', 'nir', 'swir1','pixel_qa']

start_year = 1984
end_year = 1984

start_month = 1
end_month = 12

ind_season = ['DJF','DJF',
              'MAM','MAM','MAM',
              'JJA','JJA','JJA',
              'SON','SON','SON',
              'DJF']

# Canton de Genève exact
min_lon = 5.96
max_lon = 6.32
min_lat = 46.12
max_lat = 46.37

ndvi_season = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Seasonal_NDVI_test'

ndwi_season = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Seasonal_NDWI_test'

log_name = 'AF_GE_mean_seasonal_indexes.log'
user_mail = 'alexander.folz@unepgrid.ch'
```

In [ ]:
```
# Check ind_season variable and inform in case of problem
if len(ind_season) != 12:
    display(Javascript("""
    require(
        ["base/js/dialog"],
        function(dialog) {
            dialog.modal({
                title: 'Script interrupted',
                body: 'ind_season variable should contain 12 entries (one per month)',
                buttons: {
                    'Close': {}
                }
            });
        }
    );
    """))
    sys.exit('ind_season variable should contain 12 entries (one per month)')
```

In [ ]:
```
# Detect index of starting year season
start_season = ind_season[0]
for m in range(len(ind_season), 0, -1):
    if ind_season[m - 1] != start_season:
        print(m)
        switch_ind = m
        break
```

In [ ]:
```
def multi_indexes(i):

    with dc_lock:
        dataset_clean, clean_mask = load_multi_clean(dc = dc, products = products , time = [start_date, end_date],
                                                     lon = geographic_chunks[i]['longitude'],
                                                     lat = geographic_chunks[i]['latitude'],
                                                     measurements = measurements)

        if dataset_clean != 0:

# Define NDVI variables to compute

            ndvi = (dataset_clean.nir - dataset_clean.red) / (dataset_clean.nir + dataset_clean.red)
            ndvi_count = ndvi.count(dim=['time'])
            ndvi_qual = np.rint((ndvi_count / len(ndvi['time']) * 100)).astype(np.uint8)

            ndvi_mean = np.nanmean(ndvi.values, axis=0)
            ndvi_mean = xr.DataArray(ndvi_mean, dims=['latitude', 'longitude']).astype(np.float64)
            ndvi_mean = ndvi_mean.assign_coords(latitude=ndvi.latitude, longitude=ndvi.longitude)
            ndvi_mean.name = 'ndvi_mean'

            ndvi_std = np.nanstd(ndvi.values, axis=0)
            ndvi_std = xr.DataArray(ndvi_std, dims=['latitude', 'longitude']).astype(np.float64)
            ndvi_std = ndvi_std.assign_coords(latitude=ndvi.latitude, longitude=ndvi.longitude)
            ndvi_std.name = 'ndvi_std'

# Define NDWI variables to compute

            ndwi = (dataset_clean.nir - dataset_clean.swir1) / (dataset_clean.nir + dataset_clean.swir1)
            ndwi_count = ndwi.count(dim=['time'])
            ndwi_qual = np.rint((ndwi_count / len(ndwi['time']) * 100)).astype(np.uint8)
            del dataset_clean
            del clean_mask
            ndwi_mean = np.nanmean(ndwi.values, axis=0)
            ndwi_mean = xr.DataArray(ndwi_mean, dims=['latitude', 'longitude']).astype(np.float64)
            ndwi_mean = ndwi_mean.assign_coords(latitude=ndwi.latitude, longitude=ndwi.longitude)
            ndwi_mean.name = 'ndwi_mean'

            ndwi_std = np.nanstd(ndwi.values, axis=0)
            ndwi_std = xr.DataArray(ndwi_std, dims=['latitude', 'longitude']).astype(np.float64)
            ndwi_std = ndwi_std.assign_coords(latitude=ndwi.latitude, longitude=ndwi.longitude)
```

```
            ndwi_std.name = 'ndwi_std'

# Append NDVI result

            with append_mean_lock:
                ndvi_mean_chunks.append(ndvi_mean.to_dataset('mean'))
                del ndvi_mean
            with append_std_lock:
                ndvi_std_chunks.append(ndvi_std.to_dataset('std'))
                del ndvi_std
            with append_count_lock:
                ndvi_count_chunks.append(ndvi_count.to_dataset('count'))
                del ndvi_count
            with append_qual_lock:
                ndvi_qual_chunks.append(ndvi_qual.to_dataset('qual'))
                del ndvi_qual

# Append NDWI result

            with append_mean_lock:
                ndwi_mean_chunks.append(ndwi_mean.to_dataset('mean'))
                del ndwi_mean
            with append_std_lock:
                ndwi_std_chunks.append(ndwi_std.to_dataset('std'))
                del ndwi_std
            with append_count_lock:
                ndwi_count_chunks.append(ndwi_count.to_dataset('count'))
                del ndwi_count
            with append_qual_lock:
                ndwi_qual_chunks.append(ndwi_qual.to_dataset('qual'))
                del ndwi_qual
```

In [ ]:
```
# Create/Empty working directory
if os.path.exists(ndvi_season):
    shutil.rmtree(ndvi_season)
os.makedirs(ndvi_season)

if os.path.exists(ndwi_season):
    shutil.rmtree(ndwi_season)
os.makedirs(ndwi_season)

# Cut the geographic extents into chunks

# Note that in the case of the Canton of Geneva,
# the area covered is only about = 0.25° of latitude * 0.36° of longitude, so a chunksize of 0.09 (in squared degrees)
# Hence as the chunksize of 0.09 is smaller than that used below of the calculation, the computation of the indexes
# is completed for one chunk.

chunk_size = 0.5
pool_nb = 4
geographic_chunks = create_geographic_chunks(latitude=(min_lat, max_lat),
                                             longitude=(min_lon, max_lon),
                                             geographic_chunk_size=chunk_size)
tot = len(geographic_chunks)

printandlog('Indexes seasonal mean calculation started',log_name, reset = True)
start_time = datetime.now()

years = range(start_year, end_year + 1)
# List seasons keeping original order
seen = set()
seasons = [x for x in ind_season if not (x in seen or seen.add(x))]

for year in years:
    for season in seasons:
        starty_time = datetime.now()
        print(' Preparing NDVI_%s_%s.nc & NDWI_%s_%s.nc' % (year, season, year, season))
        indxs = [i for i, x in enumerate(ind_season) if x == season]

# A l'intérieur d'un index, le chiffre réfère à la place du nombre indexé, en commençant par 0.
        if max(indxs) < switch_ind:
            start_date = datetime.strptime("%i-%i-1" % (year, min(indxs) + 1), '%Y-%m-%d')
            end_date = datetime.strptime("%i-%i-%i" % (year, max(indxs) + 1, calendar.monthrange(year, max(indxs) + 1)
```

```
[1]), '%Y-%m-%d')
        else:

            start_date = datetime.strptime("%i-%i-1" % (year - 1, min(indxs[indxs.index(switch_ind):]) + 1), '%Y-%m-%d
')
            end_date = datetime.strptime("%i-%i-%i" % (year, max(indxs[:indxs.index(switch_ind)]) + 1, calendar.monthr
ange(year, max(indxs[:indxs.index(switch_ind)]) + 1)[1]), '%Y-%m-%d')

        manager = Manager()
        ndvi_mean_chunks = manager.list([])
        ndvi_std_chunks = manager.list([])
        ndvi_count_chunks = manager.list([])
        ndvi_qual_chunks = manager.list([])

        ndwi_mean_chunks = manager.list([])
        ndwi_std_chunks = manager.list([])
        ndwi_count_chunks = manager.list([])
        ndwi_qual_chunks = manager.list([])

        dc_lock = Lock() # as dc cannot be accessed concurently
        append_mean_lock = Lock()
        append_std_lock = Lock()
        append_count_lock = Lock()
        append_qual_lock = Lock()

        bar_len = 20
        p = Pool(pool_nb)

        for i, _ in enumerate(p.imap(multi_indexes, range(tot))):
            filled_len = round(bar_len * (i + 1) / tot)
            sys.stdout.write('\r[{0}{1}] in {2}'
                             .format('#' * filled_len, '-' * (bar_len - filled_len), datetime.now() - starty_time
))
            sys.stdout.write('\r[{0}] in {1} - Seasonal indexes for all chunks completed\n'.format('#' * bar_len, date
time.now() - starty_time))
            p.close()

            if len(ndvi_mean_chunks) > 0:

# Combine chunks

                with np.errstate(divide='ignore'):
```

```
                    NDVI = combine_geographic_chunks(ndvi_mean_chunks)
                    NDVI = NDVI.merge(combine_geographic_chunks(ndvi_std_chunks))
                    NDVI = NDVI.merge(combine_geographic_chunks(ndvi_count_chunks))
                    NDVI = NDVI.merge(combine_geographic_chunks(ndvi_qual_chunks))
                    del ndvi_mean_chunks
                    del ndvi_count_chunks
                    del ndvi_qual_chunks

                    NDWI = combine_geographic_chunks(ndwi_mean_chunks)
                    NDWI = NDWI.merge(combine_geographic_chunks(ndwi_std_chunks))
                    NDWI = NDWI.merge(combine_geographic_chunks(ndwi_count_chunks))
                    NDWI = NDWI.merge(combine_geographic_chunks(ndwi_qual_chunks))
                    del ndwi_mean_chunks
                    del ndwi_count_chunks
                    del ndwi_qual_chunks

                NDVI.to_netcdf('%s/NDVI_%s_%s_stats.nc' % (ndvi_season, year, season))
                NDWI.to_netcdf('%s/NDWI_%s_%s_stats.nc' % (ndwi_season, year, season))

                printandlog('NDVI & NDWI mean of %s_%s calculated' % (year, season), log_name, started = start_time)

printandlog('ALL DONE\n', log_name)

if user_mail != '':
    os.system("mail -s 'AF_NDVI_NDWI_seasonal_stats_generator completed' %s < %s" % (user_mail, log_name))
```

Creating a Netcdf dataset containing seasonal statistics of NDVI and NDWI in the Canton of Geneva

In [3]:
```
# Import dependencies

%matplotlib inline

import os
import sys
import math

import xarray as xr
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from datetime import datetime

# For resampling a dataarray to desired extent extent and projection using GDALWARP
import rasterio

import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
# Configuration section

start_year = 1984
end_year = 2019

# Should be as in the script who generated the -nc files
ind_season = ['DJF','DJF',
              'MAM','MAM','MAM',
              'JJA','JJA','JJA',
              'SON','SON','SON',
              'DJF']

ndvi_season = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Seasonal_NDVI_test'

ndwi_season = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Seasonal_NDWI_test'

inputs = '/datacube/ui_results/notebook/AF_GE_inputs'

outputs = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary'
```

In [5]:
```
# Check ind_season variable and inform in case of problem
if len(ind_season) != 12:
    display(Javascript("""
    require(
        ["base/js/dialog"],
        function(dialog) {
            dialog.modal({
                title: 'Script interrupted',
                body: 'ind_season variable should contain 12 entries (one per month)',
                buttons: {
                    'Close': {}
                }
            });
        }
    );
    """))
    sys.exit('ind_season variable should contain 12 entries (one per month)')
```

In [6]:
```
# Detect index of starting season year
start_season = ind_season[0]
for m in range(len(ind_season), 0, -1):
    if ind_season[m - 1] != start_season:
        switch_ind = m
        break
```

In [7]:
```
# Set up season - month_day dictionnary
seen = set()
seasons =[x for x in ind_season if not (x in seen or seen.add(x))]

md_dict = {}
md_list = []
for season in seasons:
    indxs = [i for i, x in enumerate(ind_season) if x == season]


    if max(indxs) < switch_ind:
        month = indxs[math.floor(len(indxs) / 2)] + 1

    else:
        month = indxs[0] + 1

    if len(indxs) % 2 == 0:
        day = 1
    else:
        day = 15

    md_dict[season] = '%s_%s' % (month, day)
    md_list.append('%s_%s' % (month, day))

# Inversed md_dict
season_dict = {v: k for k, v in md_dict.items()}
```

In [45]:
```
## Load NDVI seasonal mean netcdfs and complete with nan

# list files in work_path (will not work with subfolders)
for dirpath, dirnames, filenames in os.walk(ndvi_season):
    break

# Compare requested and available
yearseasons_vi = []
for year in range(start_year, end_year + 1):
    for season in seasons:
        if 'NDVI_%s_%s_stats.nc' % (year, season) in filenames:
            yearseasons_vi.append('%s_%s' % (year, season))
len_yearseasons_vi = len(yearseasons_vi)

# Load and merge NDVI .nc files
NDVI = [None]*len_yearseasons_vi
times = []
for ys_ind, yearseason in enumerate(yearseasons_vi):
    date_str = '%s_%s' % (yearseason.split('_')[0], md_dict[yearseason.split('_')[1]])
    times.append(datetime.strptime(date_str, '%Y_%m_%d'))
    da = xr.open_dataset('%s/NDVI_%s_stats.nc' % (ndvi_season, yearseason))
    NDVI[ys_ind] = da
NDVI = xr.concat(NDVI, dim='time')
NDVI.coords['time'] = times

## Load NDWI seasonal mean netcdfs and complete with nan

# list files in work_path (will not work with subfolders)
for dirpath, dirnames, filenames in os.walk(ndwi_season):
    break

# Compare requested and available
yearseasons_wi = []
for year in range(start_year, end_year + 1):
    for season in seasons:
        if 'NDWI_%s_%s_stats.nc' % (year, season) in filenames:
            yearseasons_wi.append('%s_%s' % (year, season))
len_yearseasons_wi = len(yearseasons_wi)

# Load and merge NDWI .nc files
```

```
NDWI = [None]*len_yearseasons_wi
times = []
for ys_ind, yearseason in enumerate(yearseasons_wi):
    date_str = '%s_%s' % (yearseason.split('_')[0], md_dict[yearseason.split('_')[1]])
    times.append(datetime.strptime(date_str, '%Y_%m_%d'))
    da = xr.open_dataset('%s/NDWI_%s_stats.nc' % (ndwi_season, yearseason))
    NDWI[ys_ind] = da
NDWI = xr.concat(NDWI, dim='time')
NDWI.coords['time'] = times

# Renaming variables and merging both datasets

NDVI = NDVI.rename({'mean': 'ndvi_mean'})
NDVI = NDVI.rename({'std': 'ndvi_std'})
NDVI = NDVI.rename({'count': 'ndvi_count'})
NDVI = NDVI.rename({'qual': 'ndvi_qual'})

NDWI = NDWI.rename({'mean': 'ndwi_mean'})
NDWI = NDWI.rename({'std': 'ndwi_std'})
NDWI = NDWI.rename({'count': 'ndwi_count'})
NDWI = NDWI.rename({'qual': 'ndwi_qual'})

NDVI = NDVI.merge(NDWI)
VWI = NDVI
print(VWI)
```

Resample a water_mask to the dataset's extent and projection

Setting Leman lake, Rhone and Arve River as Null values

In [56]:
```
# Defining the function to clip our dataarray to the extents of the mask
# allows to resample the mask to the extents of the specified xarray dataarray

def clip_resample_load(xd, tif_path):
    """
    clip, resample and load a geotiff (tif_path) to overlay a given xarray.Dataset or DataArray (xd)
    in EPSG 4326 CRS (reproject if needed).

    Parameters
    ----------
    xd: xarray.Dataset or DataArray
    tif_path: str
        The string filepath to a GeoTIFF.
    """

    # generate temporary file name
    tmp_name = 'LS78_water_mask_GE'

    # generate a file path

    # clip and resample with gdal
    cmd = 'gdalwarp -te {:.20f} {:.20f} {:.20f} {:.20f} -ts {} {} -t_srs EPSG:4326 -co COMPRESS=LZW -wm 4096 -multi {}
{}' \
      .format(xd.longitude.values.min(), xd.latitude.values.min(), xd.longitude.values.max(), xd.latitude.values.max
(),
              len(xd.longitude), len(xd.latitude),
              tif_path, tmp_name)
    os.system(cmd)

    # load
    with rasterio.open(tmp_name, driver='GTiff') as dst:
        data_np_arr = dst.read()
        dst.close()
    os.remove(tmp_name)

    da = xr.DataArray(data_np_arr, dims=['time','latitude', 'longitude'])
    da = da.assign_coords(time = range(data_np_arr.shape[0]),
                                    latitude=xd.latitude,
                                    longitude=xd.longitude)
    return da
```

Resample water mask Netdcf file geographical extent and projection to that of a dataArray of the current dataset

In [ ]:
```
open_water_mask_GE = clip_resample_load(VWI['ndvi_mean'], './LS78_water_mask.tif')
open_water_mask_GE.plot()
## Export the water_mask to a netcdf file
#open_water_mask_GE = open_water_mask_GE.to_netcdf('%s/open_water_mask_GE.nc' % inputs)
```

Alternatively load the file from your workspace

In [ ]:
```
open_water_mask_GE = xr.open_dataarray('%s/open_water_mask_GE.nc' % inputs)

# Accessing only the latitude and longitude dimensions of the dataarray
# open_water_mask_GE.shape
# open_water_mask_GE[0]
NDVI.coords['water_mask'] = (('latitude', 'longitude'), open_water_mask_GE[0].values)
# Creating a new dataset excluding openwater areas
NDVI = NDVI.where(NDVI.water_mask != 1).dropna(how='all', dim = 'time')
NDVI
```

Exporting the entire seasonal dataset plus the seasonal temporal datasets

In [75]:
```
#VWI_DJF = VWI.where(VWI.time.dt.season == 'DJF').dropna(how='all', dim = 'time')
#VWI_MAM = VWI.where(VWI.time.dt.season == 'MAM').dropna(how='all', dim = 'time')
#VWI_JJA = VWI.where(VWI.time.dt.season == 'JJA').dropna(how='all', dim = 'time')
#VWI_SON = VWI.where(VWI.time.dt.season == 'SON').dropna(how='all', dim = 'time')

VWI.to_netcdf('%s/VWI.nc' % outputs )
VWI_DJF.to_netcdf('%s/VWI_DJF.nc' % outputs)
VWI_MAM.to_netcdf('%s/VWI_MAM.nc' % outputs)
VWI_JJA.to_netcdf('%s/VWI_JJA.nc' % outputs)
VWI_SON.to_netcdf('%s/VWI_SON.nc' % outputs)
```

VVVVVVVVVVVVVV ARCHIVE VVVVVVVVVVVVVV

In [ ]:
```
def array2da(a, ref):
    da = xr.DataArray(a, dims=['latitude', 'longitude'])
    da = da.assign_coords(latitude=ref.latitude, longitude=ref.longitude)
    da.name = [name for name in globals() if globals()[name] is a][0]
    return da
```

In [ ]:
```
import uuid

from utils.data_cube_utilities.dc_display_map import _degree_to_zoom_level
from utils.data_cube_utilities.dc_utilities import write_single_band_png_from_xr

from ipyleaflet import (
    Map,
    basemaps,
    basemap_to_tiles,
    ImageOverlay,
    DrawControl,
    LayersControl
)

def display_array_and_sample_point(da, var, cmap):
    """
    Description:
      Display a colored single variable (integer format) of an xarray.Dataset on a map
      and allow the user to select a point
    -----
    Input:
      da: xarray.Dataset
      var: variable to display (must be integer)
      cmap: colormap to apply (GRASS r.colors ot ESRI HDR color table files (.clr) format)
    Output:
      m: map to interact with
      dc: draw control
    Usage:
      View, interact and point a location to be used later on
    """

    # Check inputs
    assert 'Dataset' in str(type(da)), "da must be an xarray.Dataset"
    assert var in da, "%s variable is not available in the xarray.Dataset (da)" % var
    for item in cmap:
        if len(item) != 2: sys.exit('cmap must be a list of list [<value>, <color>] and we have %s' % item)

    # Initiate a fresh temporary working directory
    os.system("rm -rf ./tmp_display")
    os.mkdir("tmp_display")
```

```
# Convert dataArray into positive integer (needed by png format) and adapt cmap
da = da[var]
min_val = da.values.min()
if min_val < 0:
    da = (da - min_val).astype(np.uint8)
    for item in cmap:
        item[0] = item[0] - min_val

with open('./tmp_display/cmap.clr', 'w') as f:
    for item in cmap:
        f.write("%s\n" % '\t'.join(str(e) for e in item))

# Convert dataArray to temporary png
# png becomes faded dur to gdaldem use (only gdal function to deal with colormap)
# and nor -exact_color_entry or -nearest_color_entry improves it !
png_name = './tmp_display/' + str(uuid.uuid4()) + '.png'
write_single_band_png_from_xr(png_name, da.to_dataset(name = var), var,
                              no_data = 0, color_scale = './tmp_display/cmap.clr')

# Display
latitude = (da.latitude.values.min(), da.latitude.values.max())
longitude = (da.longitude.values.min(), da.longitude.values.max())

margin = -0.5
zoom_bias = 0
lat_zoom_level = _degree_to_zoom_level(margin = margin, *latitude ) + zoom_bias
lon_zoom_level = _degree_to_zoom_level(margin = margin, *longitude) + zoom_bias
zoom = min(lat_zoom_level, lon_zoom_level)
center = [np.mean(latitude), np.mean(longitude)]
m = Map(center=center, zoom=zoom)

# http://leaflet-extras.github.io/leaflet-providers/preview/
esri = basemap_to_tiles(basemaps.Esri.WorldImagery)
m.add_layer(esri)

image = ImageOverlay(url=png_name,
                     bounds=((latitude[0],longitude[0]),(latitude[1], longitude[1])),
                     opacity=0.8)
m.add_layer(image)

dc = DrawControl(circlemarker={'color': 'yellow'},
```

```
                    polygon={}, polyline={})
    m.add_control(dc)

    m.add_control(LayersControl())

    return m, dc
```

Zoom and pan, add a circlemarker to select a location and run the cells beow the map.

In [ ]:
```
# # Display sum of count (number of values used to compute statistics)
# NDVI_cnt = np.sum(VWI['ndvi_count'].values, axis=0)
# NDVI_cnt = array2da(NDVI_cnt, VWI)
# m, pos = display_array_and_sample_point(NDVI_cnt.to_dataset(name='NDVI_cnt').astype(np.uint16), 'NDVI_cnt',
#                                         [[NDVI_cnt.values.min(), 'red'], [NDVI_cnt.values.mean(), 'grey'], [NDVI_cn
t.values.max(), 'green']])

# Display Standard deviation
NDVI_std = np.nanstd(VWI['ndvi_std'].values, axis=0)
NDVI_std = array2da(NDVI_std, VWI)
m, pos = display_array_and_sample_point(NDVI_std.to_dataset(name='NDVI_std'), 'NDVI_std',
                                        [[NDVI_std.values.min(), 'red'], [NDVI_std.values.mean(), 'yellow'], [NDVI_std
.values.max(), 'green']])
m
```

In [ ]:
```
def pix_plot(da, loc, ylab, title):
    """
    Description:
      Plot mean, standard deviation (1 and 2) and count of a given xarray.Dataset for a given pixel
    -----
    Input:
      da: xarray.Dataset
      loc: pixel location (e.g. [longitude, latitude])
      ylab: y label (e.g. 'NDVI')
      title: figure title
    Output:
      XXX
    """
    px_mean = da['ndvi_mean'].sel(latitude = loc[1], longitude = loc[0], method = "nearest")
    px_std = da['ndvi_std'].sel(latitude = loc[1], longitude = loc[0], method = "nearest")
    px_count = da['ndvi_count'].sel(latitude = loc[1], longitude = loc[0], method = "nearest")

    #px_mean = da['ndwi_mean'].sel(latitude = loc[1], longitude = loc[0], method = "nearest")
    #px_std = da['ndwi_std'].sel(latitude = loc[1], longitude = loc[0], method = "nearest")
    #px_count = da['ndwi_count'].sel(latitude = loc[1], longitude = loc[0], method = "nearest")

    x = px_mean['time'].values

    fig = plt.figure(figsize = (12, 8))
    ax1 = fig.add_subplot(111)

    ax1.fill_between(x, px_mean - 2 * px_std, px_mean + 2 * px_std, label = '2 std', color = 'grey')
    ax1.fill_between(x, px_mean - px_std, px_mean + px_std, label = '1 std', color = 'lightgrey')
    ax1.plot(x, px_mean, label = 'mean', color = 'green', linewidth = 2)

    ax1.legend(loc='upper left')
    ax1.set_ylabel(ylab)
    ax1.xaxis.grid()
    ax1.yaxis.grid()
    plt.ylim(0,1)

    ax2 = ax1.twinx()
    ax2.plot(x, px_count, label = 'data count', color = 'red', linewidth = 2, alpha=0.3)
    ax2.legend(loc='upper right')
    ax2.set_ylabel('Count')
    plt.ylim(0,max(px_count) + 1)
```

```
    plt.title(title, fontweight="bold")
```

In [ ]:
```
%matplotlib inline

for season in ['DJF', 'MAM', 'JJA', 'SON']:
    pix_plot(da = VWI.where(VWI['time'].dt.season == season).dropna('time',how='all'),
             loc = pos.last_draw['geometry']['coordinates'],
             ylab = 'NDVI', title = '%s season' % (season))

    plt.savefig('NDVI_high_qual_stats_{}.png'.format(season))
```

Monthly statistics (mean, standard deviation) generator

In [1]:
```
# Import dependencies

import os
import sys
import shutil
import calendar

import xarray as xr
import numpy as np

from datetime import date, datetime
from multiprocessing import Pool, Lock, Manager
from IPython.display import display, Javascript

import datacube
dc = datacube.Datacube()

from utils.data_cube_utilities.dc_chunker import create_geographic_chunks, combine_geographic_chunks

from swiss_utils.data_cube_utilities.sdc_utilities import load_multi_clean, printandlog

import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
# Configuration section

# Make sure to change the working directories ('/.../')
# of ndvi_month and ndwi_month as they will respectively host the created resulting monthly netdcf files

# Change the logname ('') to be created and the user mail to which to logfile will be delivered

products = ["ls8_lasrc_swiss", "ls7_ledaps_swiss", "ls5_ledaps_swiss"]

measurements=['red', 'nir', 'swir1','pixel_qa']

start_year = 1984
end_year = 1985

start_month = 1
end_month = 12

# Canton de Genève exact
min_lon = 5.96
max_lon = 6.32
min_lat = 46.12
max_lat = 46.37

ndvi_month = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Monthly_NDVI_test'

ndwi_month = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Monthly_NDWI_test'

log_name = 'AF_GE_mean_monthly_indexes.log'
user_mail = 'alexander.folz@unepgrid.ch'
```

In [3]:
```
def multi_indexes(i):

    with dc_lock:
        dataset_clean, clean_mask = load_multi_clean(dc = dc, products = products , time = [start_date, end_date],
                                                     lon = geographic_chunks[i]['longitude'],
                                                     lat = geographic_chunks[i]['latitude'],
                                                     measurements = measurements)

        if dataset_clean != 0:

# Define NDVI variables to compute

            ndvi = (dataset_clean.nir - dataset_clean.red) / (dataset_clean.nir + dataset_clean.red)
            ndvi_count = ndvi.count(dim=['time'])
            ndvi_qual = np.rint((ndvi_count / len(ndvi['time']) * 100)).astype(np.uint8)

            ndvi_mean = np.nanmean(ndvi.values, axis=0)
            ndvi_mean = xr.DataArray(ndvi_mean, dims=['latitude', 'longitude']).astype(np.float64)
            ndvi_mean = ndvi_mean.assign_coords(latitude=ndvi.latitude, longitude=ndvi.longitude)
            ndvi_mean.name = 'ndvi_mean'

            ndvi_std = np.nanstd(ndvi.values, axis=0)
            ndvi_std = xr.DataArray(ndvi_std, dims=['latitude', 'longitude']).astype(np.float64)
            ndvi_std = ndvi_std.assign_coords(latitude=ndvi.latitude, longitude=ndvi.longitude)
            ndvi_std.name = 'ndvi_std'

# Define NDWI variables to compute

            ndwi = (dataset_clean.nir - dataset_clean.swir1) / (dataset_clean.nir + dataset_clean.swir1)
            ndwi_count = ndwi.count(dim=['time'])
            ndwi_qual = np.rint((ndwi_count / len(ndwi['time']) * 100)).astype(np.uint8)
            del dataset_clean
            del clean_mask
            ndwi_mean = np.nanmean(ndwi.values, axis=0)
            ndwi_mean = xr.DataArray(ndwi_mean, dims=['latitude', 'longitude']).astype(np.float64)
            ndwi_mean = ndwi_mean.assign_coords(latitude=ndwi.latitude, longitude=ndwi.longitude)
            ndwi_mean.name = 'ndwi_mean'

            ndwi_std = np.nanstd(ndwi.values, axis=0)
            ndwi_std = xr.DataArray(ndwi_std, dims=['latitude', 'longitude']).astype(np.float64)
            ndwi_std = ndwi_std.assign_coords(latitude=ndwi.latitude, longitude=ndwi.longitude)
```

```
            ndwi_std.name = 'ndwi_std'

# Append NDVI result

            with append_mean_lock:
                ndvi_mean_chunks.append(ndvi_mean.to_dataset('mean'))
                del ndvi_mean
            with append_std_lock:
                ndvi_std_chunks.append(ndvi_std.to_dataset('std'))
                del ndvi_std
            with append_count_lock:
                ndvi_count_chunks.append(ndvi_count.to_dataset('count'))
                del ndvi_count
            with append_qual_lock:
                ndvi_qual_chunks.append(ndvi_qual.to_dataset('qual'))
                del ndvi_qual

# Append NDWI result

            with append_mean_lock:
                ndwi_mean_chunks.append(ndwi_mean.to_dataset('mean'))
                del ndwi_mean
            with append_std_lock:
                ndwi_std_chunks.append(ndwi_std.to_dataset('std'))
                del ndwi_std
            with append_count_lock:
                ndwi_count_chunks.append(ndwi_count.to_dataset('count'))
                del ndwi_count
            with append_qual_lock:
                ndwi_qual_chunks.append(ndwi_qual.to_dataset('qual'))
                del ndwi_qual
```

Count, len and qual functions

ndwi_count counts, for each chunk, and for each pixel, the number of scenes (twice in a month for landsat scenes on average) which were used for calculating the ndwi function. Hence ndwi_count be either 0, 1 or 2 for landsat scenes over a month.

len(ndwi['time']) counts the amount of scenes available for each month (includes scenes covered with Nan or other Null values). As the chunk size is greater than the area of interest, effectively only one chunk is computed.

ndwi.qual refers for each chunk, and for each pixel, to the number of scenes for which ndwi was successfully calculated, as a percentage of total scenes listed by the ndwi function during a month (including Null scenes)

In [ ]:
```
# Create/Empty working directory
if os.path.exists(ndvi_month):
    shutil.rmtree(ndvi_month)
os.makedirs(ndvi_month)

if os.path.exists(ndwi_month):
    shutil.rmtree(ndwi_month)
os.makedirs(ndwi_month)

# Cut the geographic extents into chunks

# Note that in the case of the Canton of Geneva,
# the area covered is only about = 0.25° of latitude * 0.36° of longitude, so a chunksize of 0.09 (in squared degrees)
# Hence as the chunksize of 0.09 is smaller than that used below of the calculation, the computation of the indexes
# is completed for one chunk.

chunk_size = 0.5
pool_nb = 4
geographic_chunks = create_geographic_chunks(latitude=(min_lat, max_lat),
                                             longitude=(min_lon, max_lon),
                                             geographic_chunk_size=chunk_size)
tot = len(geographic_chunks)

printandlog('Indexes monthly mean calculation started',log_name, reset = True)
start_time = datetime.now()

years = range(start_year, end_year + 1)
months = range(start_month, end_month + 1)


for year in years:
    for month in months:
        starty_time = datetime.now()
        print(' Preparing NDVI_%s_%s.nc & NDWI_%s_%s.nc' % (year, month, year, month))

# A l'intérieur d'un index, le chiffre réfère à la place du nombre indexé, en commençant par 0.

        start_date = datetime.strptime("%i-%i-1" % (year, month), '%Y-%m-%d')
        end_date = datetime.strptime("%i-%i-%i" % (year, month, calendar.monthrange(year, month)[1]), '%Y-%m-%d')

        manager = Manager()
```

```
        ndvi_mean_chunks = manager.list([])
        ndvi_std_chunks = manager.list([])
        ndvi_count_chunks = manager.list([])
        ndvi_qual_chunks = manager.list([])

        ndwi_mean_chunks = manager.list([])
        ndwi_std_chunks = manager.list([])
        ndwi_count_chunks = manager.list([])
        ndwi_qual_chunks = manager.list([])

        dc_lock = Lock() # as dc cannot be accessed concurently
        append_mean_lock = Lock()
        append_std_lock = Lock()
        append_count_lock = Lock()
        append_qual_lock = Lock()

        bar_len = 20
        p = Pool(pool_nb)

        for i, _ in enumerate(p.imap(multi_indexes, range(tot))):
            filled_len = round(bar_len * (i + 1) / tot)
            sys.stdout.write('\r[{0}{1}] in {2}'
                             .format('#' * filled_len, '-' * (bar_len - filled_len), datetime.now() - starty_time
))
            sys.stdout.write('\r[{0}] in {1} - Monthly indexes for all chunks completed\n'.format('#' * bar_len, datet
ime.now() - starty_time))
            p.close()

            if len(ndvi_mean_chunks) > 0:

# Combine chunks

                with np.errstate(divide='ignore'):
                    NDVI = combine_geographic_chunks(ndvi_mean_chunks)
                    NDVI = NDVI.merge(combine_geographic_chunks(ndvi_std_chunks))
                    NDVI = NDVI.merge(combine_geographic_chunks(ndvi_count_chunks))
                    NDVI = NDVI.merge(combine_geographic_chunks(ndvi_qual_chunks))
                    del ndvi_mean_chunks
                    del ndvi_count_chunks
                    del ndvi_qual_chunks

                    NDWI = combine_geographic_chunks(ndwi_mean_chunks)
```

```
                    NDWI = NDWI.merge(combine_geographic_chunks(ndwi_std_chunks))
                    NDWI = NDWI.merge(combine_geographic_chunks(ndwi_count_chunks))
                    NDWI = NDWI.merge(combine_geographic_chunks(ndwi_qual_chunks))
                    del ndwi_mean_chunks
                    del ndwi_count_chunks
                    del ndwi_qual_chunks

                NDVI.to_netcdf('%s/NDVI_%s_%02d_stats.nc' % (ndvi_month, year, month))
                NDWI.to_netcdf('%s/NDWI_%s_%02d_stats.nc' % (ndwi_month, year, month))

                printandlog('NDVI & NDWI mean of %s_%s calculated' % (year, month), log_name, started = start_time)

printandlog('ALL DONE\n', log_name)

if user_mail != '':
    os.system("mail -s 'AF_NDVI_NDWI_monthly_stats_generator completed' %s < %s" % (user_mail, log_name))
```

Monthly statistics analysis

In [2]:
```
# Import dependencies

%matplotlib inline

import os
import sys

import xarray as xr
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime, timedelta

import warnings
warnings.filterwarnings("ignore")
```

In [132]:
```
# Configuration section

start_year = 1984
end_year = 1985

start_month = 1
end_month = 12

ind_month = ['01','02',
             '03','04','05',
             '06','07','08',
             '09','10','11',
             '12']

ndvi_month = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Monthly_NDVI_test'

ndwi_month = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary/Monthly_NDWI_test'

outputs = '/datacube/ui_results/notebook/AF_GE_outputs/Temporary'
```

In [59]:
```
# Set up month_day dictionnary
months =[x for x in ind_month]

md_dict = {}

for month in months:
    indxs = [i for i, x in enumerate(ind_month) if x == month]

    if len(indxs) % 2 == 0:
        day = 1
    else:
        day = 15

    md_dict[month] = '%s_%s' % (month, day)

# Inversed md_dict
month_dict = {v: k for k, v in md_dict.items()}
```

In [ ]:
```
# Load NDVI monthly mean netcdfs and complete with nan

# list files in work_path (will not work with subfolders)
for dirpath, dirnames, filenames in os.walk(ndvi_month):

    # Compare requested and available years
    yearmonths = []
    no_yearmonths = []
    for year in range(start_year, end_year + 1):
        for month in range(start_month, end_month + 1):
            if 'NDVI_%s_%02d_stats.nc' % (year, month) in filenames:
                yearmonths.append('%s_%02d' % (year, month))
            else:
                no_yearmonths.append('%s_%02d' % (year, month))

    len_yearmonths = len(yearmonths)

    # Load and merge .nc files
    NDVI = [None]*len_yearmonths
    times = []
    for ym_ind, yearmonth in enumerate(yearmonths):
        date_str = '%s_%s' % (yearmonth.split('_')[0], md_dict[yearmonth.split('_')[1]])
        times.append(datetime.strptime(date_str, '%Y_%m_%d'))
        da = xr.open_dataset('%s/NDVI_%s_stats.nc' % (ndvi_month, yearmonth))
        NDVI[ym_ind] = da
    NDVI = xr.concat(NDVI, dim='time')
    NDVI = NDVI.assign_coords(time = times)

    #for ym_ind, time in enumerate(NDVI.time):
       # nan_NDVI = NDVI.isel(time=0).where(not NDVI.isel(time=0), 0)
        #nan_NDVI['mean'] = nan_NDVI['mean'].where(not nan_NDVI)
        #nan_NDVI.coords['time'] = times
        #nan_NDVI = nan_NDVI.expand_dims('time')
        #NDVI = xr.merge([NDVI, nan_NDVI])

######


## Load NDWI monthly mean netcdfs and complete with nan
```

```
# list files in work_path (will not work with subfolders)
for dirpath, dirnames, filenames in os.walk(ndwi_month):

    # Compare requested and available years
    yearmonths = []
    no_yearmonths = []
    for year in range(start_year, end_year + 1):
        for month in range(start_month, end_month + 1):
            if 'NDWI_%s_%02d_stats.nc' % (year, month) in filenames:
                yearmonths.append('%s_%02d' % (year, month))
            else:
                no_yearmonths.append('%s_%02d' % (year, month))

    len_yearmonths = len(yearmonths)

    # Load and merge .nc files
    NDWI = [None]*len_yearmonths
    times = []
    for ym_ind, yearmonth in enumerate(yearmonths):
        date_str = '%s_%s' % (yearmonth.split('_')[0], md_dict[yearmonth.split('_')[1]])
        times.append(datetime.strptime(date_str, '%Y_%m_%d'))
        da = xr.open_dataset('%s/NDWI_%s_stats.nc' % (ndwi_month, yearmonth))
        NDWI[ym_ind] = da
    NDWI = xr.concat(NDWI, dim='time')
    NDWI = NDWI.assign_coords(time = times)

    #for ym_ind, time in enumerate(NDVI.time):
       # nan_NDWI = NDWI.isel(time=0).where(not NDWI.isel(time=0), 0)
        #nan_NDWI['mean'] = nan_NDWI['mean'].where(not nan_NDWI)
        #nan_NDWI.coords['time'] = times
        #nan_NDWI = nan_NDWI.expand_dims('time')
        #NDWI = xr.merge([NDWI, nan_NDWI])

    # Renaming variables and merging both datasets

NDVI = NDVI.rename({'mean': 'ndvi_mean'})
NDVI = NDVI.rename({'std': 'ndvi_std'})
NDVI = NDVI.rename({'count': 'ndvi_count'})
NDVI = NDVI.rename({'qual': 'ndvi_qual'})

NDWI = NDWI.rename({'mean': 'ndwi_mean'})
NDWI = NDWI.rename({'std': 'ndwi_std'})
```

```
NDWI = NDWI.rename({'count': 'ndwi_count'})
NDWI = NDWI.rename({'qual': 'ndwi_qual'})

NDVI = NDVI.merge(NDWI)
VWI_monthly = NDVI
VWI_monthly.to_netcdf('%s/VWI_monthly.nc' % outputs)
print(VWI_monthly)
```

Seasonal Mann_Kendall pixel temporal tendency in the Canton of Geneva

In [84]:
```
# Import dependencies

import numpy as np
import xarray as xr
import gdal
import osr
import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
```

In [7]:
```
# Configure work_path

work_path = '/datacube/ui_results/notebook/AF_GE_outputs/Final'
```

In [66]:
```
# Loading yearseason NDVI datasets

NDVI_DJF = xr.open_dataset('%s/NDVI_DJF.nc' % (work_path))
NDVI_MAM = xr.open_dataset('%s/NDVI_MAM.nc' % (work_path))
NDVI_JJA = xr.open_dataset('%s/NDVI_JJA.nc' % (work_path))
NDVI_SON = xr.open_dataset('%s/NDVI_SON.nc' % (work_path))

# Renaming NDVI variables and creating dataarrays of seasonal means

NDVI_DJF.rename({'mean': 'NDVI_DJF_mean'}, inplace = 1)
NDVI_DJF_mean = NDVI_DJF['NDVI_DJF_mean']
NDVI_MAM.rename({'mean': 'NDVI_MAM_mean'}, inplace = 1)
NDVI_MAM_mean = NDVI_MAM['NDVI_MAM_mean']
NDVI_JJA.rename({'mean': 'NDVI_JJA_mean'}, inplace = 1)
NDVI_JJA_mean = NDVI_JJA['NDVI_JJA_mean']
NDVI_SON.rename({'mean': 'NDVI_SON_mean'}, inplace = 1)
NDVI_SON_mean = NDVI_SON['NDVI_SON_mean']

# Loading yearseason NDWI datasets

NDWI_JJA = xr.open_dataset('%s/NDWI_JJA.nc' % (work_path))
NDWI_SON = xr.open_dataset('%s/NDWI_SON.nc' % (work_path))
NDWI_DJF = xr.open_dataset('%s/NDWI_DJF.nc' % (work_path))
NDWI_MAM = xr.open_dataset('%s/NDWI_MAM.nc' % (work_path))

# Renaming NDWI variables and creating dataarrays of seasonal means

NDWI_DJF.rename({'mean': 'NDWI_DJF_mean'}, inplace = 1)
NDWI_DJF_mean = NDWI_DJF['NDWI_DJF_mean']
NDWI_MAM.rename({'mean': 'NDWI_MAM_mean'}, inplace = 1)
NDWI_MAM_mean = NDWI_MAM['NDWI_MAM_mean']
NDWI_JJA.rename({'mean': 'NDWI_JJA_mean'}, inplace = 1)
NDWI_JJA_mean = NDWI_JJA['NDWI_JJA_mean']
NDWI_SON.rename({'mean': 'NDWI_SON_mean'}, inplace = 1)
NDWI_SON_mean = NDWI_SON['NDWI_SON_mean']

# Defining list containing seasonal mean dataarrays

season_ds = [NDVI_DJF_mean, NDVI_MAM_mean, NDVI_JJA_mean, NDVI_SON_mean, NDWI_DJF_mean, NDWI_MAM_mean, NDWI_JJA_mean,
NDWI_SON_mean]
```

In [83]:
```
# Function to export to geotiff

EXT2DRIV_CONVERSION = {
    "tif": "GTiff",
    "nc": "netCDF",
    "img": "HFA"
}

NP2GDAL_CONVERSION = {
      "uint8": 1,
      "int8": 1,
      "uint16": 2,
      "int16": 3,
      "uint32": 4,
      "int32": 5,
      "float32": 6,
      "uint64": 7,
      "int64": 7,
      "float64": 7,
      "complex64": 10,
      "complex128": 11,
    }

def da2raster(da, rast_name, nodata_val = None, md_rast = None, md_band = None, opt = []):
    """
    Description:
      Convert an xarray.DataArray into a proper Geotiff (no shift, CRS (epsg:4326), metadata and nodata value)
    -----
    Input:
      da: xarray.DataArray
      rast_name: string
          Output file name (could be .tif. img or .nc, but the last one will not manage the metadata)
      nodata: number (optional)
      md_rast: dictionnary (optional)
          raster metadata e.g. {'long_name':'productivity indicators', 'configuartion':'Landsat 5,7 and 8 2001-2015'}
      md_band: dictionnary (optional)
          band metadata e.g. {'long_name':'linear trend', 'units':'NDVI/year'}
      opt: list
          gdal option (depends on used raster format) e.g. ['COMPRESS=DEFLATE']
          see https://www.gdal.org/formats_list.html for options per format
    Output:
```

```
      raster file
    """

    # identify GDAL Driver using rast_name extent
    driv = EXT2DRIV_CONVERSION[rast_name.split('.')[-1]]

    # identify da GDAL data type
    # source: https://borealperspectives.wordpress.com/2014/01/16/data-type-mapping-when-using-pythongdal-to-write-numpy-arrays-to-geotiff/
    gdaltype = NP2GDAL_CONVERSION[da.dtype.name]

    # set band name
    band_name = da.name if da.name is not None else 'noname'

    # compute georeferencing parameters
    cols = da.shape[1]
    rows = da.shape[0]
    rasterOrigin = (da.longitude.values.min(), da.latitude.values.max()) # Top-Left
    pixelWidth = (da.longitude.values.max() - da.longitude.values.min()) / (cols - 1)
    pixelHeight = (da.latitude.values.max() - da.latitude.values.min()) / (rows - 1)
    origin_left = rasterOrigin[0] - pixelWidth / 2
    origin_top = rasterOrigin[1] + pixelHeight / 2

    # Create the geotiff
    driver = gdal.GetDriverByName(driv)
    outRaster = driver.Create(rast_name, cols, rows, 1, gdaltype, options=opt)
    if md_rast:
        outRaster.SetMetadata(md_rast)
    outRaster.SetGeoTransform((origin_left, pixelWidth, 0, origin_top, 0, -pixelHeight))
    outband = outRaster.GetRasterBand(1)
    outband.WriteArray(da.values)
    outband.SetDescription(band_name)
    if md_band:
        outband.SetMetadata(md_band)
    if nodata_val:
        outband.SetNoDataValue(nodata_val)
    outRasterSRS = osr.SpatialReference()
    outRasterSRS.ImportFromEPSG(4326)
    outRaster.SetProjection(outRasterSRS.ExportToWkt())
    outband.FlushCache()

    return 0
```

Mann-Kendall trajectory

In [91]:
```
# To compute Mann-Kendall trajectory of other datasets, change the dataset list accordingly

# trajectory metadata for geotiff export

for data in season_ds:
    name = data.name
    mdata = {'description':'%s trajectory using Landsat 5,7 and 8 over the period 1984-2019' % (name)}
    y = data

# Calculate mk_trend

###

    def mann_kendall(y, alpha = 0.05):
        n = len(y)

        # calculate S
        s = 0
        for k in range(n-1):
            for j in range(k+1,n):
                v = np.sign(y[j] - y[k])
                v = v.fillna(0) # replace nan with 0
                s += v
        return s.astype(np.int16)

    mk_trend = mann_kendall(y).compute()
    mk_trend.name = '%s_mk_trend' % (name)

    da2raster(mk_trend, '%s/%s_mk_trend.tif' % (work_path, name), nodata_val = 9999,
              md_rast = mdata,
              md_band = {'long_name':'%s_mk_trend_statistic' % (name), 'units':'None'},
              opt = ['COMPRESS=DEFLATE'])

    def da_linreg_params(y):
        x = y.where(np.isnan(y), y['time.year']) # attribute time to pixel with values

        mean_x = x.mean(dim='time')
        mean_y = y.mean(dim='time')
        mean_xx = (x * x).mean(dim='time')
        mean_xy = (x * y).mean(dim='time')
```

```
    s = ((mean_x * mean_y) - mean_xy) / ((mean_x * mean_x) - mean_xx)

    i = mean_y - mean_x * s

    return s, i

lin_trend, intercept = da_linreg_params(y)

lin_trend.name = '%s_lin_trend' % (name)

da2raster(lin_trend, '%s/%s_lin_trend.tif' % (work_path, name), nodata_val = 9999,
          md_rast = mdata,
          md_band = {'long_name':'%s_linear trend' % (name), 'units':'%s/season' % (name.split('_')[0])},
          opt = ['COMPRESS=DEFLATE'])

# Calulate Mann-Kendall significance

### if the number of observations is 36, the new n is going to be 32. The function get_kendall_coef()
### will loop through the list and return the value when the index reaches 32
def get_kendall_coef(n, level=95):
    # The minus 4 is because the indexing below is for a sample size of 4
    assert(n >= 4)
    n = n - 4
    coefs = {90: [4, 6, 7, 9, 10, 12, 15, 17, 18, 22, 23, 27, 28, 32, 35, 37, 40, 42,
                  45, 49, 52, 56, 59, 61, 66, 68, 73, 75, 80, 84, 87, 91, 94, 98, 103,
                  107, 110, 114, 119, 123, 128, 132, 135, 141, 144, 150, 153, 159,
                  162, 168, 173, 177, 182, 186, 191, 197, 202],
             95: [4, 6, 9, 11, 14, 16, 19, 21, 24, 26, 31, 33, 36, 40, 43, 47, 50, 54,
                  59, 63, 66, 70, 75, 79, 84, 88, 93, 97, 102, 106, 111, 115, 120,
                  126, 131, 137, 142, 146, 151, 157, 162, 168, 173, 179, 186, 190,
                  197, 203, 208, 214, 221, 227, 232, 240, 245, 251, 258],
             99: [6, 8, 11, 18, 22, 25, 29, 34, 38, 41, 47, 50, 56, 61, 65, 70, 76, 81,
                  87, 92, 98, 105, 111, 116, 124, 129, 135, 142, 150, 155, 163, 170,
                  176, 183, 191, 198, 206, 213, 221, 228, 236, 245, 253, 260, 268,
                  277, 285, 294, 302, 311, 319, 328, 336, 345, 355, 364]}
    return coefs[level][n]

kendall90 = get_kendall_coef(len(y), 90)
kendall95 = get_kendall_coef(len(y), 95)
kendall99 = get_kendall_coef(len(y), 99)

# np.logical_and used instead of np.all as it is 30 to 40% faster
```

```
signif = np.full((mk_trend.shape[0], mk_trend.shape[1]), 9999)
signif = np.where(np.logical_and(lin_trend.values < 0, abs(mk_trend) >= kendall90), -1, signif)
signif = np.where(np.logical_and(lin_trend.values < 0, abs(mk_trend) >= kendall95), -2, signif)
signif = np.where(np.logical_and(lin_trend.values < 0, abs(mk_trend) >= kendall99), -3, signif)
signif = np.where(np.logical_and(lin_trend.values > 0, abs(mk_trend) >= kendall90), 1, signif)
signif = np.where(np.logical_and(lin_trend.values > 0, abs(mk_trend) >= kendall95), 2, signif)
signif = np.where(np.logical_and(lin_trend.values > 0, abs(mk_trend) >= kendall99), 3, signif)
signif = np.where(abs(mk_trend.values) <= kendall90, 0, signif)
signif = np.where(abs(lin_trend.values) <= 0.001, 0, signif)

# Convert signif to a proper DataArray
signif = xr.DataArray(signif, dims=['latitude', 'longitude']).astype(np.int16)
signif = signif.assign_coords(latitude=mk_trend.latitude, longitude=mk_trend.longitude)
signif.name = '%s_signif' % (name)

da2raster(signif, '%s/%s_trend_signif.tif' % (work_path, name), nodata_val = 9999,
          md_rast = mdata,
          md_band = {'long_name':'%s_trend_significance' % (name), 'units':'probability_occurence'},
          opt = ['COMPRESS=DEFLATE'])

del mk_trend
del lin_trend
del signif
```

In [7]:
```
# Import dependencies

import os
import numpy as np # For plotting
import xarray as xr
import matplotlib.pyplot as plt
import osr
import gdal
```

In [9]:
```
# Configure directories

inputs = './bigdata/AF_GE_inputs'

big_data = './bigdata/AF_GE_outputs/Final'

inputs_2 = './bigdata/AF_GE_outputs/Final/AF_GE_mean_seasonal_NDVI'

inputs_3 = './bigdata/AF_GE_outputs/Final/AF_GE_mean_seasonal_NDWI'

export = './bigdata/AF_GE_outputs/Final/NC_to_TIF'
```

In [8]:
```
# Function to export a file as geotiff

EXT2DRIV_CONVERSION = {
    "tif": "GTiff",
    "nc": "netCDF",
    "img": "HFA"
}

NP2GDAL_CONVERSION = {
      "uint8": 1,
      "int8": 1,
      "uint16": 2,
      "int16": 3,
      "uint32": 4,
      "int32": 5,
      "float32": 6,
      "uint64": 7,
      "int64": 7,
      "float64": 7,
      "complex64": 10,
      "complex128": 11,
    }

def da2raster(da, rast_name, nodata_val = None, md_rast = None, md_band = None, opt = []):
    """
    Description:
      Convert an xarray.DataArray into a proper Geotiff (no shift, CRS (epsg:4326), metadata and nodata value)
    -----
    Input:
      da: xarray.DataArray
      rast_name: string
          Output file name (could be .tif. img or .nc, but the last one will not manage the metadata)
      nodata: number (optional)
      md_rast: dictionnary (optional)
          raster metadata e.g. {'long_name':'productivity indicators', 'configuartion':'Landsat 5,7 and 8 2001-2015'}
      md_band: dictionnary (optional)
          band metadata e.g. {'long_name':'linear trend', 'units':'NDVI/year'}
      opt: list
          gdal option (depends on used raster format) e.g. ['COMPRESS=DEFLATE']
          see https://www.gdal.org/formats_list.html for options per format
    Output:
```

```
      raster file
    """

    # identify GDAL Driver using rast_name extent
    driv = EXT2DRIV_CONVERSION[rast_name.split('.')[-1]]

    # identify da GDAL data type
    # source: https://borealperspectives.wordpress.com/2014/01/16/data-type-mapping-when-using-pythongdal-to-write-numpy-arrays-to-geotiff/
    gdaltype = NP2GDAL_CONVERSION[da.dtype.name]

    # set band name
    band_name = da.name if da.name is not None else 'noname'

    # compute georeferencing parameters
    cols = da.shape[1]
    rows = da.shape[0]
    rasterOrigin = (da.longitude.values.min(), da.latitude.values.max()) # Top-Left
    pixelWidth = (da.longitude.values.max() - da.longitude.values.min()) / (cols - 1)
    pixelHeight = (da.latitude.values.max() - da.latitude.values.min()) / (rows - 1)
    origin_left = rasterOrigin[0] - pixelWidth / 2
    origin_top = rasterOrigin[1] + pixelHeight / 2

    # Create the geotiff
    driver = gdal.GetDriverByName(driv)
    outRaster = driver.Create(rast_name, cols, rows, 1, gdaltype, options=opt)
    if md_rast:
        outRaster.SetMetadata(md_rast)
    outRaster.SetGeoTransform((origin_left, pixelWidth, 0, origin_top, 0, -pixelHeight))
    outband = outRaster.GetRasterBand(1)
    outband.WriteArray(da.values)
    outband.SetDescription(band_name)
    if md_band:
        outband.SetMetadata(md_band)
    if nodata_val:
        outband.SetNoDataValue(nodata_val)
    outRasterSRS = osr.SpatialReference()
    outRasterSRS.ImportFromEPSG(4326)
    outRaster.SetProjection(outRasterSRS.ExportToWkt())
    outband.FlushCache()

    return 0
```

In [ ]:
```
# Define variables for operating the loop conversion of nc files to raster

NDVI_JJA = xr.open_dataset('%s/NDVI_JJA.nc' % big_data)

years = NDVI_JJA.time.dt.year
years

# Load the water mask to apply to each nc file before creating the TIFF file

open_water_mask_GE = xr.open_dataarray('%s/open_water_mask_GE.nc' % inputs)
open_water_mask_GE[0]
```

In [ ]:
```
# Convert NDVI nc files into geotiff to operate zonal statistics


# list files in work_path (will not work with subfolders)
for dirpath, dirnames, filenames in os.walk(inputs_2):
    for filename in filenames:
        print (filename)
        da = xr.open_dataset('%s/%s' % (inputs_2, filename))
        da.coords['water_mask'] = (('latitude', 'longitude'), open_water_mask_GE[0].values)
        da = da.where(da.water_mask != 1).dropna(how='all', dim = 'latitude')
        da = da['count']
        da2raster(da,'%s/%s_count_%s_%s.tif' % (export, filename.split('_')[0], filename.split('_')[1], filename.split
('_')[2]), nodata_val = -9999,  opt = ['COMPRESS=DEFLATE'])
```

In [ ]:
```
# Convert NDWI nc files into geotiff to operate zonal statistics


# list files in work_path (will not work with subfolders)
for dirpath, dirnames, filenames in os.walk(inputs_3):
    for filename in filenames:
        print (filename)
        da = xr.open_dataset('%s/%s' % (inputs_3, filename))
        da.coords['water_mask'] = (('latitude', 'longitude'), open_water_mask_GE[0].values)
        da = da.where(da.water_mask != 1).dropna(how='all', dim = 'latitude')
        da = da['count']
        da2raster(da,'%s/%s_count_%s_%s.tif' % (export, filename.split('_')[0], filename.split('_')[1], filename.split
('_')[2]), opt = ['COMPRESS=DEFLATE'])
```

In [2]:
```
# Import dependencies necessary for doing zonal statistics

import numpy as np
import xarray as xr
import os
import time
import pandas as pd
import geopandas as gpd
from osgeo import osr

import shapely
import descartes
import rasterio
import rasterstats
import matplotlib.lines as mlines
import matplotlib.pyplot as plt
from matplotlib import colors


from rasterstats import zonal_stats
from shapely.geometry import Polygon
```

In [3]:
```
# Configure workspace

# Note that you may have to change directory locations when computing zonal statistics

inputs = './bigdata/AF_GE_inputs'

export = './bigdata/AF_GE_outputs/Final/Mean_Count_Std_indices_Yearseason_CH/Mean'

export2 ='./bigdata/AF_GE_outputs/Final/Trends_season_CH'

big_data = './bigdata/AF_GE_outputs/Temporary/Zs_Mean_TYPE_timeseries'
```

In [ ]:
```
# Yearseason zonal statistics by cultural type

GE_TYPE = gpd.read_file('%s/Dissolved_boundaries/GE_TYPE.shp' % inputs)
GE_TYPE.head()

for dirpath, dirnames, filenames in os.walk(export):
    for f_ind, filename in enumerate (filenames):
        if  '%s_mean_%s_%s' % (filename.split('_')[0],filename.split('_')[2], filename.split('_')[3]) in filename:
            print(filename)
            zs = zonal_stats('%s/Dissolved_boundaries/GE_TYPE.shp' % inputs, '%s/%s_mean_%s_%s' % (export, filename.sp
lit('_')[0],filename.split('_')[2], filename.split('_')[3]), stats= ["mean"])
            zs = pd.DataFrame(zs)
            zs = GE_TYPE.merge(zs, how='left', left_index= bool('TRUE'), right_index= bool('TRUE'))

            # exporting files as csv

            zs = zs.drop(columns = ['geometry'])
            zs
            zs.to_csv('%s/%s_mean_%s_%s.csv' % (big_data, filename.split('_')[0],filename.split('_')[2], filename.spli
t('_')[3].split('.')[0]))
```

In [ ]:
```
# Yearseason zonal statistics by parcel

GE_Par = gpd.read_file('%s/Undissolved_boundaries/GE.shp' % inputs)
GE_Par

for dirpath, dirnames, filenames in os.walk(export):
    for f_ind, filename in enumerate (filenames):
        print(filename)
        if  '%s_mean_%s_%s.tif' % (filename.split('_')[0],filename.split('_')[2], filename.split('_')[3]) in filename:

            zs = zonal_stats('%s/Undissolved_boundaries/GE.shp' % inputs, '%s/%s_mean_%s_%s.tif' % (export, filename.s
plit('_')[0],filename.split('_')[2], filename.split('_')[3]), stats= ["mean"])
            zs = pd.DataFrame(zs)
            zs = GE_Par.merge(zs, how='left', left_index= bool('TRUE'), right_index= bool('TRUE'))

            # exporting files as csv

            zs = zs.drop(columns = ['geometry'])
            zs.head()
            zs.to_csv('%s/%s_mean_%s_%s.csv' % (big_data, (filename.split('_')[0],filename.split('_')[2], filename.spl
it('_')[3]))
```

In [ ]:
```
# Zonal statistics of seasonal trends by parcels

# Step to follow if the clipping shapefile must be dissolved by its aggregation ID

GE = gpd.read_file('%s/Undissolved_boundaries/GE.shp' % inputs)
GE = GE[['TYPE', 'geometry']]
GE_TYPE = GE.dissolve(by ='TYPE')
GE_TYPE = GE_TYPE.reset_index('TYPE')

GE_TYPE.crs = {'init': 'epsg:2056'}
gdf = gpd.GeoDataFrame(GE_TYPE, geometry = GE_TYPE.geometry, crs=GE_TYPE.crs)
list(gdf)



gdf.to_file(driver = 'ESRI Shapefile', filename= '%s/Dissolved_boundaries/GE_TYPE.shp' % inputs)

GE_TYPE = gpd.read_file('%s/Dissolved_boundaries/GE_TYPE.shp' % inputs)
GE_TYPE.head()

for dirpath, dirnames, filenames in os.walk(export2):
    for f_ind, filename in enumerate (filenames):
        if '%s_%s_trend_diff_xx_xix_signif.tif' % (filename.split('_')[0],filename.split('_')[1]) in filename:

            zs = zonal_stats('%s/Dissolved_boundaries/GE_TYPE.shp' % inputs, '%s/%s_%s_trend_signif.tif' % (export2, f
ilename.split('_')[0], filename.split('_')[1]), stats= ["count", "mean", "median", "std"])
            zs = pd.DataFrame(zs)
            zs = GE_TYPE.merge(zs, how='left', left_index= bool('TRUE'), right_index= bool('TRUE'))

            # exporting files as csv

            zs = zs.drop(columns = ['geometry'])
            zs.head()
            zs.to_csv('%s/Zs_Trends_season_TYPE/%s_%s_trend_signif.csv' % (big_data, filename.split('_')[0], filename.
split('_')[1]))
```

In [ ]:
```
# Zonal statistics of seasonal trends by commune

# Step to follow if the clipping shapefile is dissolved by its aggregation ID

GE = gpd.read_file('%s/Undissolved_boundaries/GE.shp' % inputs)
GE = GE[['COM_FED', 'geometry']]
GE_COM = GE.dissolve(by ='COM_FED')
GE_COM = GE_COM.reset_index('COM_FED')

GE_COM.crs = {'init': 'epsg:2056'}
gdf = gpd.GeoDataFrame(GE_COM, geometry = GE_COM.geometry, crs=GE_COM.crs)
list(gdf)
gdf.to_file(driver = 'ESRI Shapefile', filename= '%s/Dissolved_boundaries/GE_COM.shp' % inputs)

GE_COM = gpd.read_file('%s/Dissolved_boundaries/GE_COM.shp' % inputs)
GE_COM.head()

for dirpath, dirnames, filenames in os.walk(export2):
    for f_ind, filename in enumerate (filenames):
        if '%s_%s_trend_signif.tif' % (filename.split('_')[0],filename.split('_')[1]) in filename:

            zs = zonal_stats('%s/Dissolved_boundaries/GE_COM.shp' % inputs, '%s/%s_%s_trend_signif.tif' % (export2, fi
lename.split('_')[0], filename.split('_')[1]), stats= ["count", "mean", "median", "std"])
            zs = pd.DataFrame(zs)
            zs = GE_COM.merge(zs, how='left', left_index= bool('TRUE'), right_index= bool('TRUE'))

            # exporting files as csv

            zs = zs.drop(columns = ['geometry'])
            zs.head()
            zs.to_csv('%s/Zs_Trends_season_COM/%s_%s_trend_signif_COM.csv' % (big_data, filename.split('_')[0], filena
me.split('_')[1]))
```

In [ ]:
```
# Zonal statistics of seasonal trends by cultural type
# Step to follow if the clipping shapefile is dissolved by its aggregation ID

GE = gpd.read_file('%s/Undissolved_boundaries/GE.shp' % inputs)
GE = GE[['TYPE', 'geometry']]
GE_TYPE = GE.dissolve(by ='TYPE')
GE_TYPE = GE_TYPE.reset_index('TYPE')

GE_TYPE.crs = {'init': 'epsg:2056'}
gdf = gpd.GeoDataFrame(GE_TYPE, geometry = GE_TYPE.geometry, crs=GE_TYPE.crs)
list(gdf)
gdf.to_file(driver = 'ESRI Shapefile', filename= '%s/Dissolved_boundaries/GE_TYPE.shp' % inputs)

GE_TYPE = gpd.read_file('%s/Dissolved_boundaries/GE_TYPE.shp' % inputs)
GE_TYPE.head()

for dirpath, dirnames, filenames in os.walk(export2):
    for f_ind, filename in enumerate (filenames):
        if '%s_%s_trend_signif.tif' % (filename.split('_')[0],filename.split('_')[1]) in filename:

            zs = zonal_stats('%s/Dissolved_boundaries/GE_TYPE.shp' % inputs, '%s/%s_%s_trend_signif.tif' % (export2, f
ilename.split('_')[0], filename.split('_')[1]), stats= ["count", "mean", "median", "std"])
            zs = pd.DataFrame(zs)
            zs = GE_TYPE.merge(zs, how='left', left_index= bool('TRUE'), right_index= bool('TRUE'))

            # exporting files as csv

            zs = zs.drop(columns = ['geometry'])
            zs.head()
            zs.to_csv('%s/Zs_Trends_season_TYPE/%s_%s_trend_signif_TYPE.csv' % (big_data, filename.split('_')[0], file
name.split('_')[1]))
```

In [13]:
```
# Import dependencies

import os
import xarray as xr
import pandas as pd
import numpy as np
##

import matplotlib.lines as mlines # Necessary for dissolving shapefiles
import matplotlib.pyplot as plt # For plotting
from matplotlib import colors # For plotting
```

In [14]:
```
# Configure workspace

zonal = './bigdata/AF_GE_outputs/Temporary/Zs_Mean_TYPE_timeseries'

big_data = './bigdata/AF_GE_outputs/Final'
```

In [ ]:
```
# Aggregate CSV files

da = []

seasons = {'DJF' :'02', 'MAM':'05', 'JJA':'08', 'SON':'11'}

for dirpath, dirnames, filenames in os.walk(zonal):
    for f_ind, filename in enumerate (filenames):
        df = pd.read_csv('%s/%s' % (zonal, filename))
        df = df.drop('Unnamed: 0', axis=1)
        df=df.rename(columns ={'mean': '%s_mean_%s_%s' % (filename.split('_')[0], filename.split('_')[2], filename.spl
it('_')[3].split('.')[0]), 'TYPE':'Culture_type'})
        da.append(df)

da = pd.concat(da, axis = 1)

da = da.loc[:,~da.columns.duplicated()]
da = da.reindex(sorted(da.columns), axis=1)
da = da.round(3)
da = da.transpose()
da.to_csv('%s/zs_seasonal_mean_TYPE.csv' % big_data)
da
```