

STATLOG (LANDSAT SATELLITE) DATA SET

Rapport final
ESILV – Python for Data Analysis

By SCHAUB Yanniss & EL-HADDAD Imrane

SOMMAIRE

- I) EXPLICATION DU DATASET
- II) REFLEXION SUR LE SUJET
- III) VISUALISATION DES DONNÉES
- IV) PRÉDICTION
- V) API FLASK

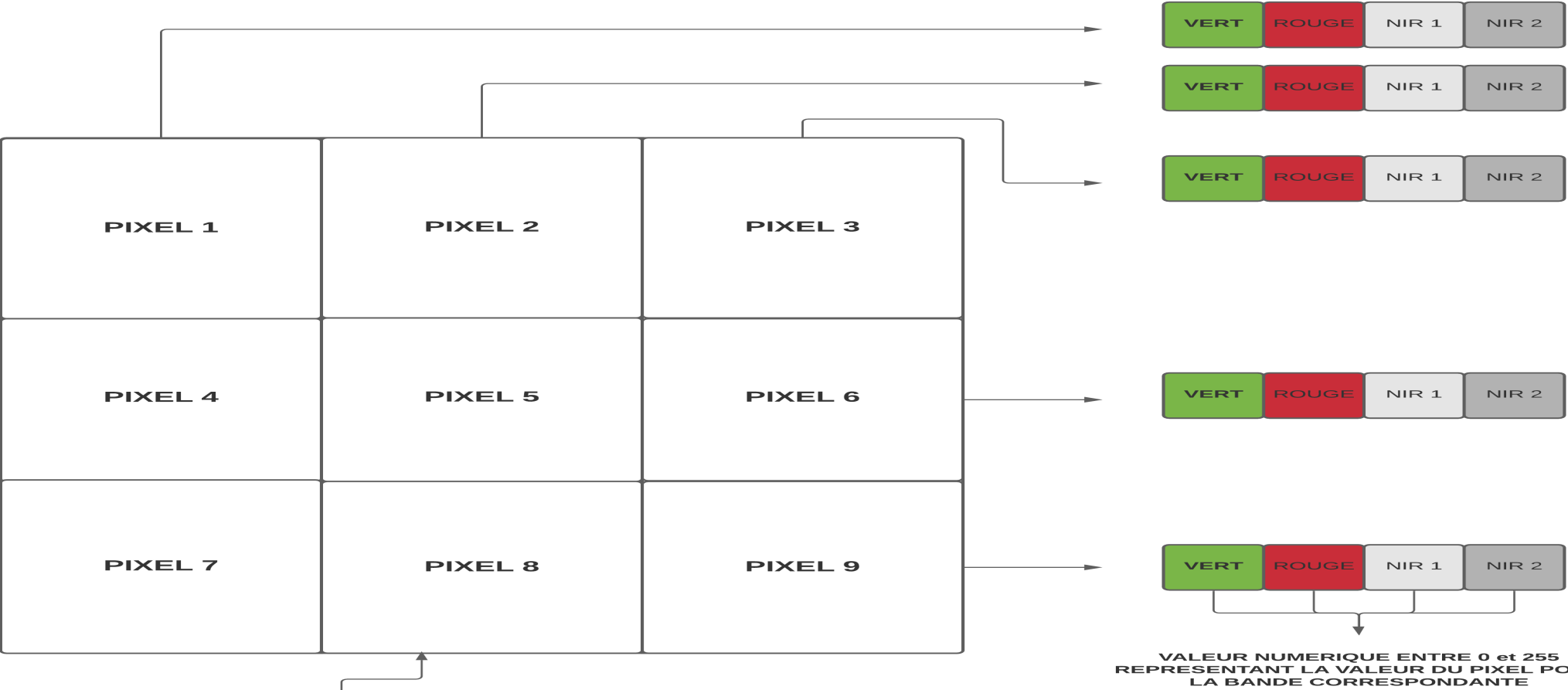
I) EXPLICATION DU DATASET

- Ces données correspondent au "Statlog (Landsat Satellite) Data Set".
- Les données Landsat originales pour cette base de données ont été générées à partir de données achetées à la NASA
- La base de données comprend les valeurs multispectrales des pixels des quartiers 3x3 d'une image satellite, et la classification associée au pixel central dans chaque quartier.
- La résolution spatiale d'un pixel est d'environ 80m x 80m. Chaque image contient 2340 x 3380 de ces pixels.
- La base de données est une (minuscule) sous-zone d'une scène, composée de 82 x 100 pixels. Chaque ligne de données correspond à un quartier de 3x3 carrés de pixels entièrement contenus dans la sous-zone de 82x10. Chaque pixel est constitué de 4 valeurs numériques comprises entre 0 et 255 correspondant à la couleur du pixel dans 4 grandes bandes spectrales :
 - *Bande Verte*
 - *Bande Rouge*
 - *Bande Near Infrarouge 1*
 - *Bande Near Infrarouge 2*
 - Ce qui nous donne $4 \times 9 (3 \times 3) = 36$ colonnes + 1 colonne de classification = 37 colonnes.

I) EXPLICATION DU DATASET

- Le but de cette analyse est de prédire cette classification, en utilisant des valeurs multispectrales.
- Dans la base de données échantillon, la classe d'un pixel est codée sous forme de nombre. La classe à prédire correspond à des valeurs entre 1 et 7. Nous avons 4435 valeurs dans le training set et 2000 dans le testing set. C'est un dataset associé à une multi classification, le but étant de connaître quel est la classe associée au pixel.
- Numéro de la classe :
 - 1 : terre rouge
 - 2 : la culture du coton
 - 3 : terre grise
 - 4 : sol gris humide
 - 5 : sol avec chaumes de végétation
 - 6 : classe de mélange (tous les types présents)
 - 7 : sol gris très humide
 - NB. Il n'y a pas d'exemples avec la classe 6 dans cet ensemble de données.
- Les données sont données dans un ordre aléatoire et certaines lignes de données ont été supprimées pour ne pas reconstruire l'image originale à partir de cet ensemble de données. Dans chaque ligne de données, les quatre valeurs spectrales pour le pixel supérieur gauche sont données en premier, suivies des quatre valeurs spectrales pour le pixel moyen supérieur, puis de celles pour le pixel supérieur droit, et ainsi de suite avec les pixels lus en séquence de gauche à droite et de haut en bas.
- Ainsi, les quatre valeurs spectrales pour le pixel central sont données par les attributs 17, 18, 19 et 20. La classification est associée uniquement au pixel central
- Nous avons réalisé un schéma pour expliquer le dataset (slide suivante)

EXPLICATION DU DATASET - SATELLITE LANDSTAT



II) REFLEXION SUR LE SUJET

- Le but étant de prédire la classification d'un pixel, il nous a fallu en amont bien comprendre le dataset afin de faire des analyses pertinentes et utiles à la résolution de la prédiction finale
- On a choisi d'analyser d'une part les valeurs de tous les pixels et dans un second temps uniquement les valeurs des pixels centraux afin d'avoir des analyses claires
- On s'est posé la question de si une bande spectrale en particulier avait un impact plus fort sur la classification qu'une autre, si la moyenne des valeurs de ces pixels pouvait nous donner des indications sur la classification, si la distribution des valeurs des pixels pourrait nous renseigner sur la classification...

II) REFLEXION SUR LE SUJET

■ Problèmes rencontrés :

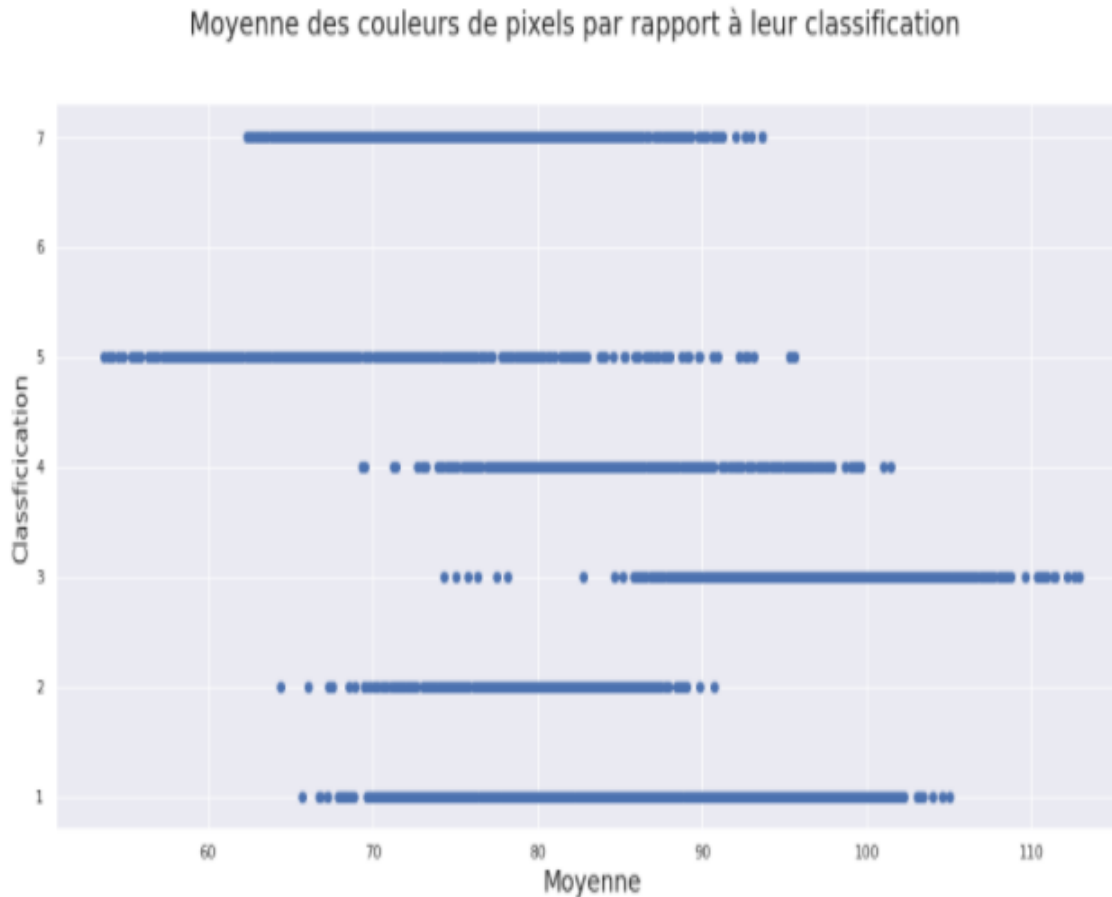
- *Dataset difficile à comprendre : beaucoup d'informations, sujet que l'on ne maîtrise pas et qui ne fait pas partie de notre quotidien.*
- *Nous n'avons qu'un seul dataset qui fonctionnait pour notre groupe de 2 élèves (celui de Imrane EL-HADDAD ne contient aucune donnée)*
- *Difficulté dans la visualisation des données car nous avons des données mélangées et non structurées*

III) VISUALISATION DES DONNEES

- Les données que nous avons sont réparties en 2 fichiers :
 - *1 trainset de 4435 valeurs (1 valeur = 1 bloc de 3 X 3 pixel)*
 - *1 testset de 2000 valeurs*

- Ces deux documents contiennent 37 colonnes :
 - *36 colonnes correspondants au 4 bandes spectrales de chacun des 9 pixels (3X3)*
 - *1 colonne de classification*

III) VISUALISATION DES DONNEES



Après une première analyse, cela nous a permis d'identifier quelques constats :

- Principalement, ces valeurs numériques sont à peu près regroupées :

- Classe 7** : Entre 63 et 89 (environ)

- Classe 6** : Aucune Data

- Classe 5** : Entre 55 et 85 (environ)

- Classe 4** : Entre 75 et 96 (environ)

- Classe 3** : Entre 85 et 109 (environ)

- Classe 2** : Entre 70 et 86 (environ)

- Classe 1** : Entre 70 et 103 (environ)

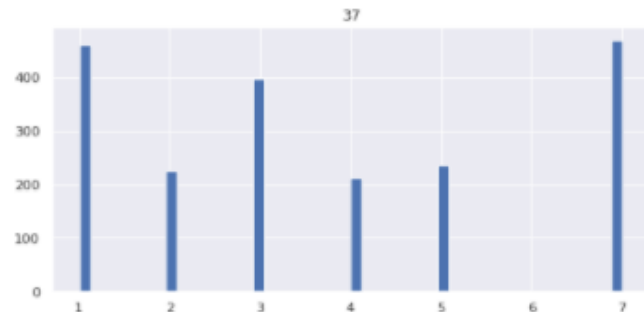
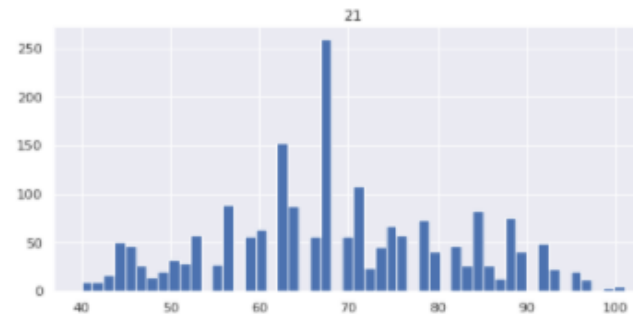
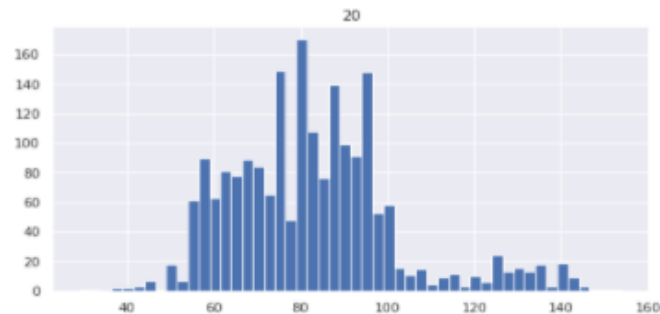
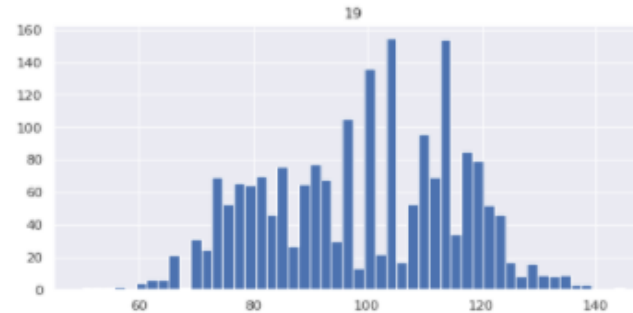
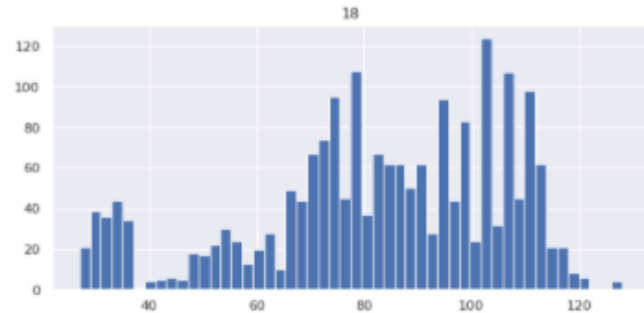
- La moyenne ne peut pas suffire pour déterminer la classification mais on a pu en déduire que :

- Si la moyenne du 3X3 pixel est entre (environ) 103 et + --> **Sa classification sera 1 ou 3**

- Si la moyenne du 3X3 pixel est entre 0 et 70 (environ) --> **Sa classification sera 5 ou 7**

III) VISUALISATION DES DONNEES

```
#Nous allons tenter d'observer la distribution des données pour notre partie de dataset concernant le pixel centrale  
statlog_tst_only_main_five_columns.hist(bins=50, figsize=(20,15))  
plt.show()
```



Nous avons la distribution des valeurs numériques des pixels centraux de nos 3X3 pixels qui nous montrent que pour chaque bande de chaque pixel central possède un intervalle de valeur plus rencontrées dans le dataset :

Pour la colonne 18 (Bande Verte) :
Entre 70 et 110 principalement

Pour la colonne 19 (Bande Rouge) :
Entre 70 et 125 principalement

Pour la colonne 20 (Bande NIR 1) :
Entre 55 et 100 principalement

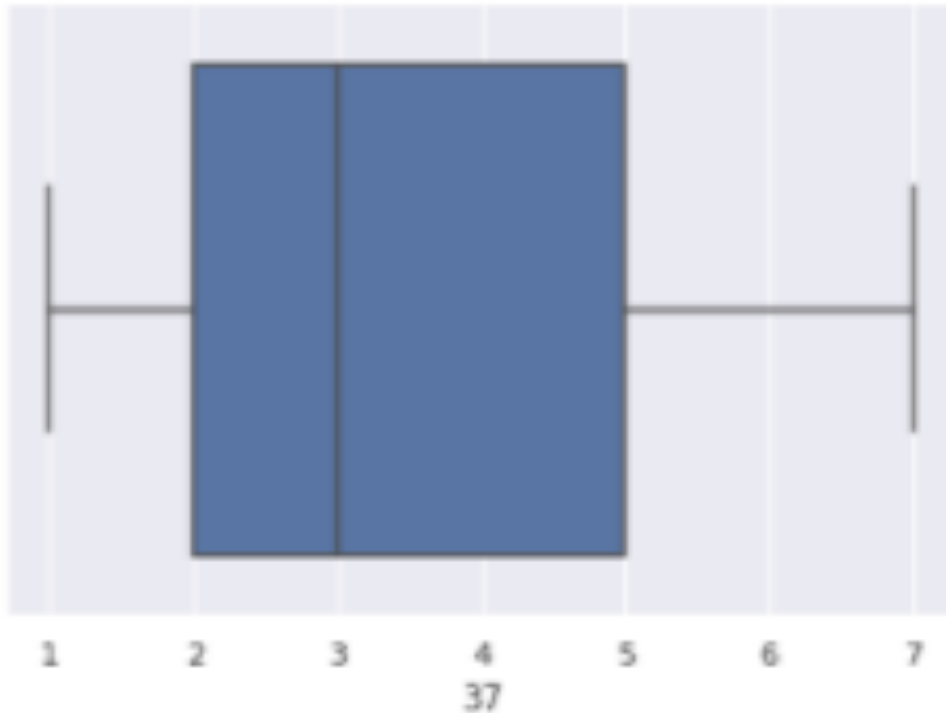
Pour la colonne 21 (Bande NIR 2) : Il y a un pic qui se démarque des autres data --> valeur 67 - 69 (environ)

III) VISUALISATION DES DONNEES



```
sns.boxplot(x=statlog_tst_only_main_five_colomns['37'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa5d4ad0b70>
```



Un box plot montre la distribution des données quantitatives de manière à faciliter les comparaisons entre les variables. La boîte montre les quartiles de l'ensemble de données tandis que les moustaches s'étendent pour montrer le reste de la distribution, à l'exception des points qui sont déterminés comme étant "aberrants" en utilisant une méthode qui est fonction de l'intervalle interquartile.

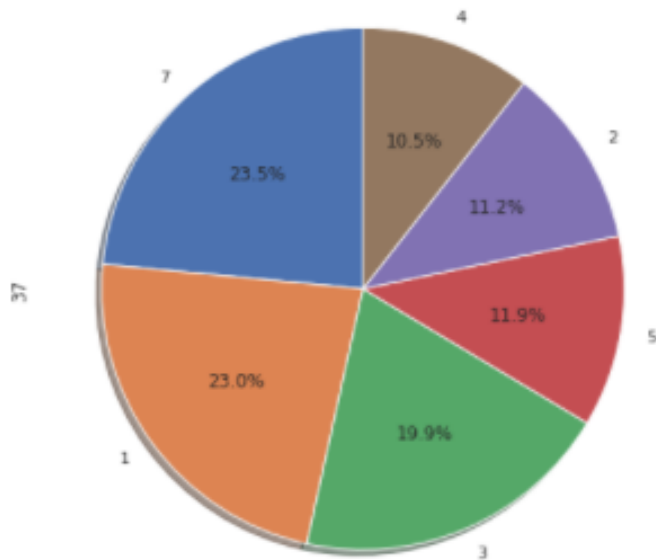
Dans notre cas, on observe que la plus pars des pixels de cette image se situe **entre la classe 2 et la classe 5.**

III) VISUALISATION DES DONNEES

```
[30] statlog_tst_only_main_five_colomns['37'].value_counts().plot(kind='pie', subplots=True, shadow = True, startangle=90,
      figsize=(13,8), autopct='%1.1f%%', title="Classification")
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fa5d223ae10>],
      dtype=object)
```

Classification



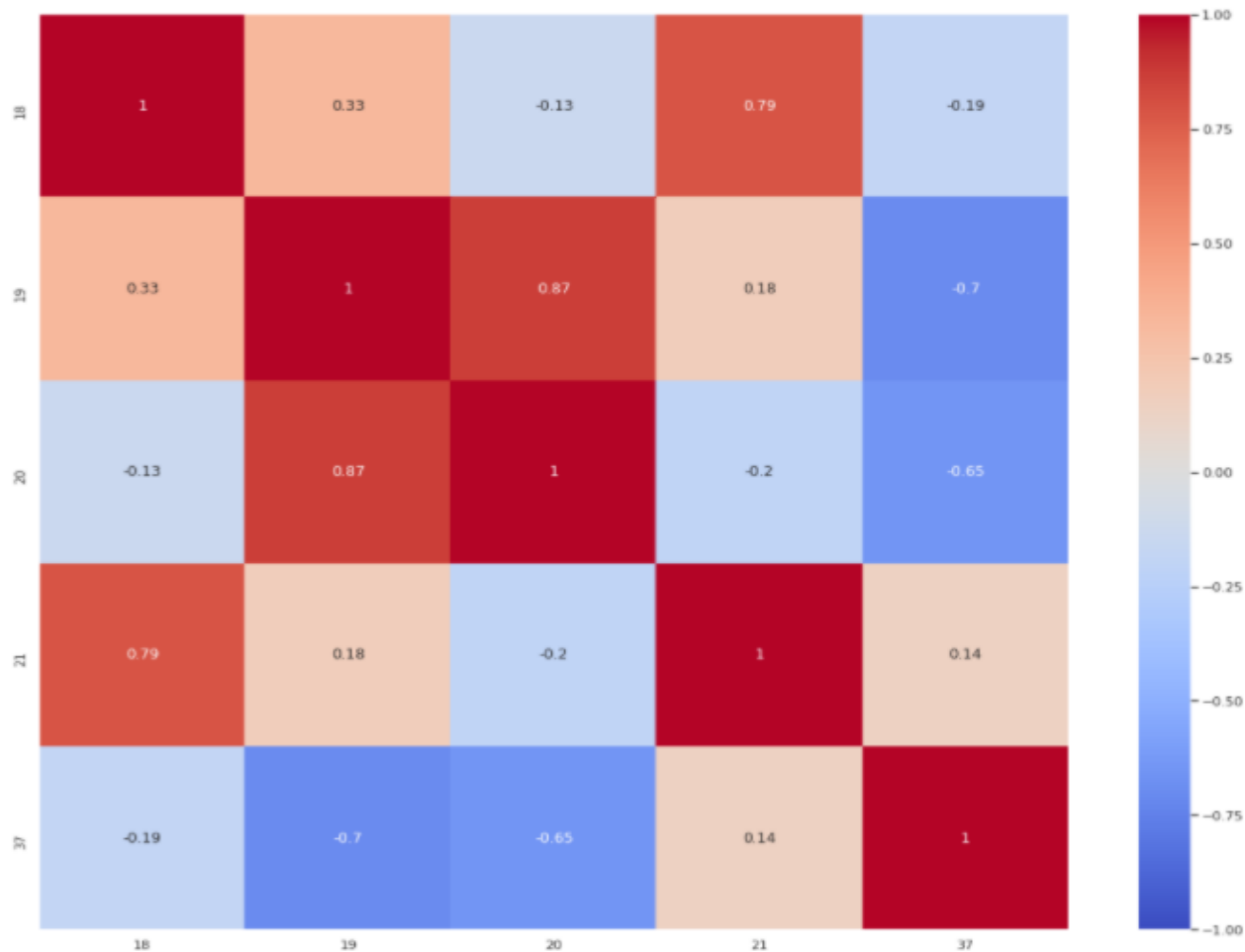
En moyenne, la classification de nos pixels est à 3.70, ce qui semblerait peut être que la surface globale analysée par le satellite est entre du "sol gris" (classe 3) et du "sol gris humide" (classe 4)

Avec une observation des pourcentages deligne appartenant à chaque classification, il semblerait que la classification majoritaire de cette surface serait :

La classe 7 : **“sol gris très humide”**
mais avec un pourcentage très proche de la classe :1 **“Sol Rouge”**

III) VISUALISATION DES DONNEES

```
[32] plt.figure(figsize=(20,15))
sns.heatmap(statlog_tst_only_main_five_columns.corr("pearson"),
            vmin=-1, vmax=1,
            cmap='coolwarm',
            annot=True,
            square=True);
```



La corrélation positive entre "37" et "21" (0,1) nous laisse penser que la valeur numérique de la bande NIR 2 du pixel central à un impact positif sur la classification (plus la valeur numérique de la bande NIR2 est élevée = plus la classification est élevée)

Mais la corrélation négative entre "37" et "19" (-0,70) et "20" (-0,65) nous laisse penser que la valeur numérique de la bande Rouge et la bande NIR 1 du pixel central à un impact négatif sur la classification (plus la valeur numérique de la bande NIR2 est élevée = plus la classification est faible)

On peut en déduire que les bandes Rouge et NIR1 ont un plus grand impact sur la classification que les deux autres bandes.

IV) PREDICTION

PREDICTION

```
[33] from sklearn.linear_model import LogisticRegression  
     from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, f1_score
```

```
[34] #Separation des données de notre trainset et de notre dataset dans 4 variables  
     X_train = statlog_trn.iloc[:,0:36]  
     Y_train = statlog_trn.iloc[:,-1]  
     X_test = statlog_tst.iloc[:,0:36]  
     Y_test = statlog_tst.iloc[:,-1]
```

En analysant les labels et l'évènement de multiclassification (1 à 7), on peut apercevoir que ses données ne sont pas bien réparties dans notre dataset (mélangé aléatoirement par les auteurs)

La métrique "f1_score" semble être la métrique parfaite pour analyser la prédiction de nos modèles, afin de savoir quel modèle nous modifierons dans le but d'avoir le meilleur modèle possible pour notre dataset

Le score F1 peut être interprété comme une moyenne pondérée de la précision et du rappel, où un score F1 atteint sa meilleure valeur à 1 et sa pire valeur à 0. La contribution relative de la précision et du rappel au score F1 est égale.

IV) PREDICTION

Ada Boost

```
[35] #Modèle Ada Boost
      from sklearn.ensemble import AdaBoostClassifier

      ada_model = AdaBoostClassifier()

      ada_model.fit(X_train, Y_train)

      ada_prediction = ada_model.predict(X_test)

      ada_f1_score = f1_score(Y_test, ada_prediction, average='weighted')
      print('f1_score : ' + str(ada_f1_score))

      f1_score : 0.6913806422929347
```

La valeur du métrique f1_score n'est pas optimale (environ 70 %). Nous allons essayer avec d'autres modèles

IV) PREDICTION

Logistic Regression

```
[36] #Modèle LogisticRegression model
      model = LogisticRegression()

      model.fit(X_train, Y_train)
      model_prediction = model.predict(X_test)

      model_f1_score = f1_score(Y_test, model_prediction, average='weighted')
      print('f1_score : ' + str(model_f1_score))
```

```
f1_score : 0.7583027139712215
```

La valeur du métrique f1_score n'est pas optimale (environ 75 %) mais est un peu meilleur que le modèle précédent. Nous allons essayer avec d'autres modèles

IV) PREDICTION

Decision Tree

```
[37] #Modèle Decision Tree model
      from sklearn.tree import DecisionTreeClassifier, plot_tree

      tree_model = DecisionTreeClassifier()

      tree_model.fit(X_train, Y_train)

      tree_prediction = tree_model.predict(X_test)

      tree_f1_score = f1_score(Y_test, tree_prediction, average='weighted')
      print('f1_score : ' + str(tree_f1_score))

      f1_score : 0.8522594014077233
```

La valeur du métrique f1_score n'est pas optimale (environ 85 %) mais est un peu meilleur que le modèle précédent. On se rapproche de quelque chose de correct mais nous allons essayer avec d'autres modèles

IV) PREDICTION

Boosting

```
[38] #Modèle Boosting model
      from sklearn.ensemble import GradientBoostingClassifier

      boosting_model = GradientBoostingClassifier()

      boosting_model.fit(X_train, Y_train)

      boosting_prediction = boosting_model.predict(X_test)
      boosting_prediction_proba = boosting_model.predict_proba(X_test)

      boosting_f1_score = f1_score(Y_test, boosting_prediction, average='weighted')
      print('f1_score : ' + str(boosting_f1_score))

      f1_score : 0.8909649730985422
```

La valeur du métrique f1_score a augmenté (environ 89 %) mais nous allons essayer avec d'autres modèles

IV) PREDICTION

K-Neighbors

```
[39] #Modèle KNeighbors model
      from sklearn.neighbors import KNeighborsClassifier

      knn_model = KNeighborsClassifier()

      knn_model.fit(X_train, Y_train)

      knn_prediction = knn_model.predict(X_test)

      knn_f1_score = f1_score(Y_test, knn_prediction, average='weighted')
      print('f1_score : ' + str(knn_f1_score))
```

```
○ f1_score : 0.903709462983651
```

La valeur du métrique f1_score a augmenté (environ 90 %) mais nous allons essayer avec un dernier modèle

IV) PREDICTION

Random Forest

```
▶ #Modèle SRandom Forest model
from sklearn.ensemble import RandomForestClassifier

forest_model = RandomForestClassifier(random_state=42)

forest_model.fit(X_train, Y_train)

forest_prediction = forest_model.predict(X_test)

forest_f1_score = f1_score(Y_test, forest_prediction, average='weighted')
print('f1_score : ' + str(forest_f1_score))
```

```
↳ f1_score : 0.9085232613433516
```

La valeur du métrique f1_score a augmenté (environ 91 %) et cette valeur nous semble acceptable. Nous allons modifier les hyperparamètres de notre Random Forest afin d'avoir un modèle sur mesure pour nos datasets.

III) VISUALISATION DES DONNEES

```
▶ tab_result_comparison=pd.DataFrame(tab )  
tab_result_comparison
```

| | Modelling Algo | f1_score |
|---|----------------------------|----------|
| 0 | AdaBoostClassifier | 0.691381 |
| 1 | LogisticRegression | 0.758303 |
| 2 | DecisionTreeClassifier | 0.852331 |
| 3 | GradientBoostingClassifier | 0.890532 |
| 4 | KNeighborsClassifier | 0.903709 |
| 5 | RandomForestClassifier | 0.908523 |

C'est le **Random Forest** qui possède le meilleur f1_score

C'est donc ce modèle que nous allons modifier

III) VISUALISATION DES DONNEES

```
▶ rfc=RandomForestClassifier(random_state=42)

#Nous avons choisis de modifier les hyperparamètres : n_estimators, max_depth et min_impurity_split
param_grid = {
    'n_estimators':range(50,500,50), #[50,100,150,200,250,300,350,400,500],
    'max_depth' : [6,8,10,20,30], #[50,100,150,200,250,300,350,400,500],
    'min_impurity_split' : [0.01,0.001,0.0005,0.0001]
}

CV_rfc = GridSearchCV(estimator=rfc,param_grid=param_grid,cv=2, n_jobs=2, scoring='accuracy')
CV_rfc.fit(X_train, Y_train)

GridSearchCV(cv=2, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False, random_state=42,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=2,
             param_grid={'max_depth': [6, 8, 10, 20, 30],
                         'min_impurity_split': [0.01, 0.001, 0.0005, 0.0001],
                         'n_estimators': range(50, 500, 50)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

Nous avons modifié les hyperparamètres de notre Random Forest en choisissant les bonnes valeurs de ces paramètres grâce au module GridSearchCV

III) VISUALISATION DES DONNEES

```
[50] plot_scatter(tab_nestimator, tab_mean_score, 'Random Forest Classifier', 'tab_nestimator', 'tab_mean_score')
```



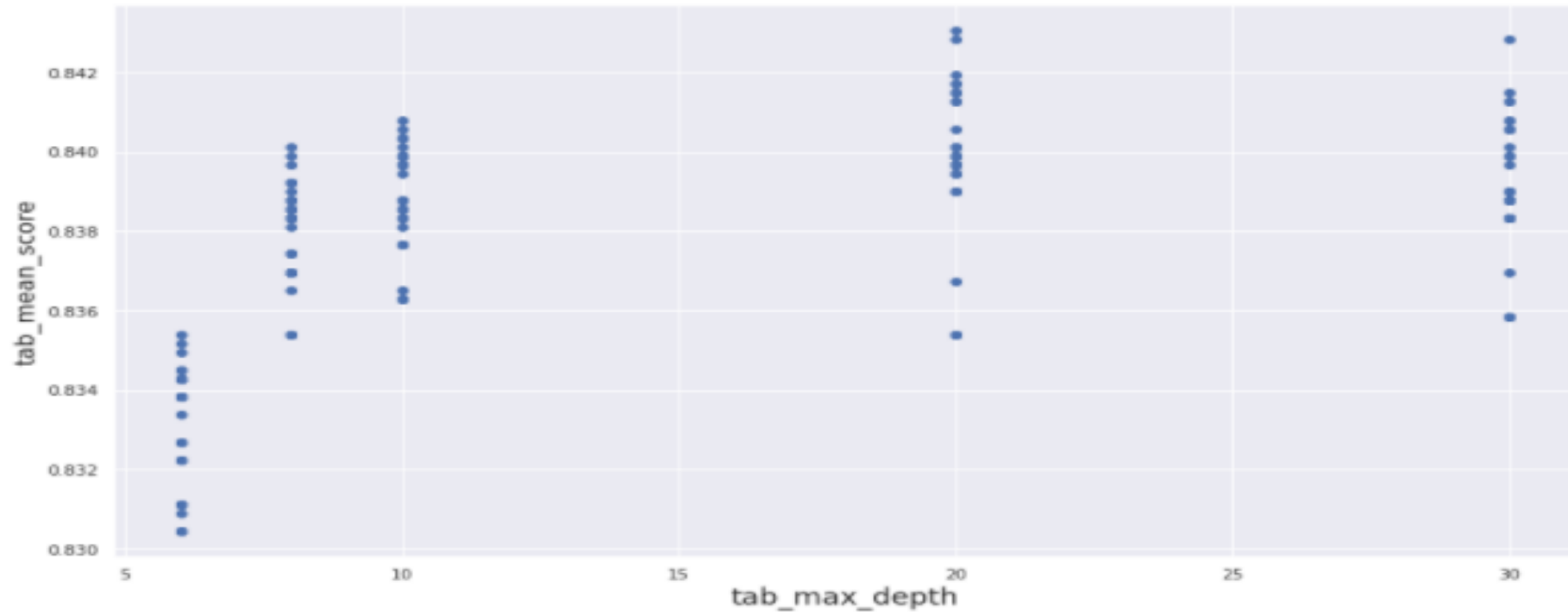
Avec ce graphique nous pouvons déduire que le meilleur `n_estimator` est 20 et qu'avec uniquement ce paramètre, notre modèle random forest modifié possède un `mean_score` de 0,843

III) VISUALISATION DES DONNEES

```
plot_scatter(tab_max_depth, tab_mean_score, 'Random Forest Classifier', 'tab_max_depth', 'tab_mean_score')
```



Random Forest Classifier



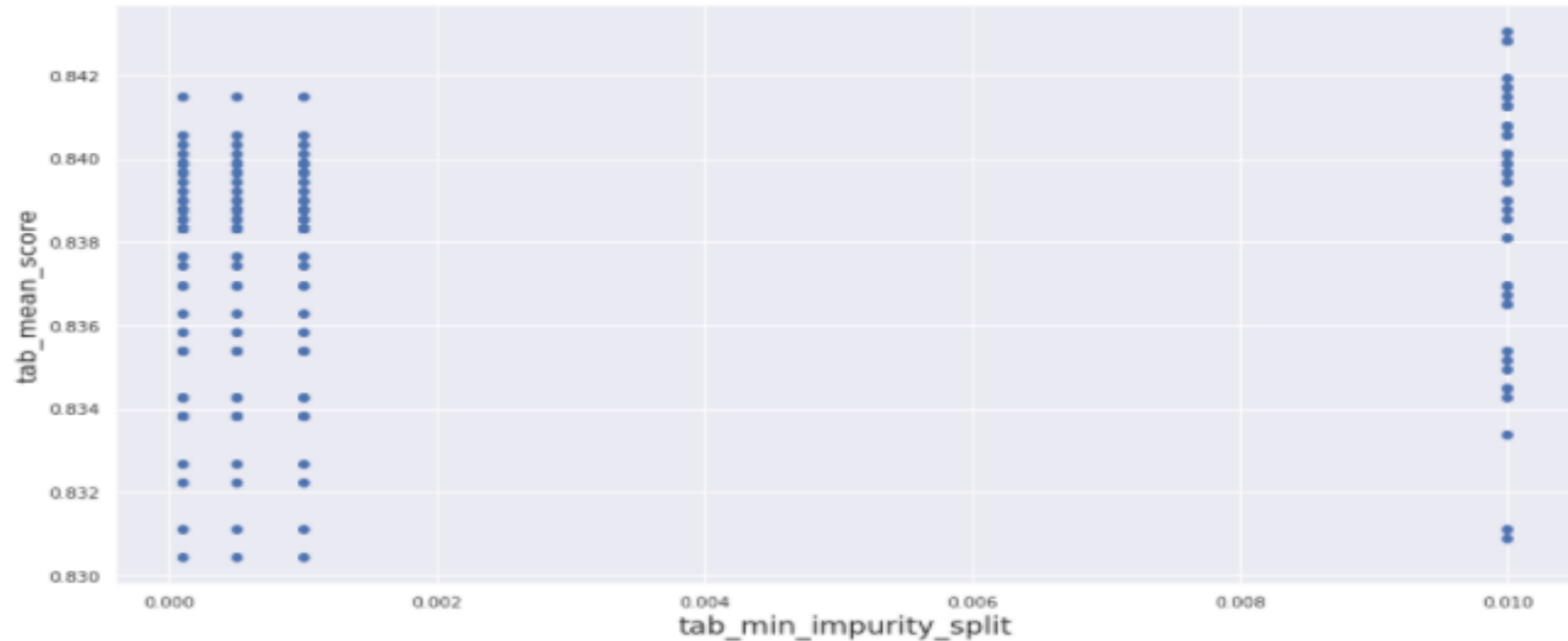
Avec ce graphique nous pouvons déduire que le meilleur max_depth est 20 et qu'avec uniquement ce paramètre, notre modèle random forest modifié possède un mean_score de 0,843

III) VISUALISATION DES DONNEES

```
plot_scatter(tab_min_impurity_split, tab_mean_score, 'Random Forest Classifier', 'tab_min_impurity_split', 'tab_mean_score')
```



Random Forest Classifier



Avec ce graphique nous pouvons déduire que le meilleur min_impurity_split est 0.01 et qu'avec uniquement ce paramètre, notre modèle random forest modifié possède un mean_score de 0.843

III) VISUALISATION DES DONNEES

```
[53] CV_rfc.best_params_
```

```
{'max_depth': 20, 'min_impurity_split': 0.01, 'n_estimators': 150}
```

La fonction `.best_params_` confirme les analyses que nous venons d'établir.

IV) PREDICTION

Modèle finale en utilisant les meilleurs hyperparamètres

```
[54] rfc1=RandomForestClassifier(random_state=42, n_estimators= 150, max_depth=20, min_impurity_split = 0.01)
```

```
[55] rfc1.fit(X_train, Y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,  
                        criterion='gini', max_depth=20, max_features='auto',  
                        max_leaf_nodes=None, max_samples=None,  
                        min_impurity_decrease=0.0, min_impurity_split=0.01,  
                        min_samples_leaf=1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, n_estimators=150,  
                        n_jobs=None, oob_score=False, random_state=42, verbose=0,  
                        warm_start=False)
```

```
[56] pred=rfc1.predict(X_test)
```

```
[57] print("F1_score for Random Forest on CV data: ",f1_score(Y_test,pred,average='weighted'))
```

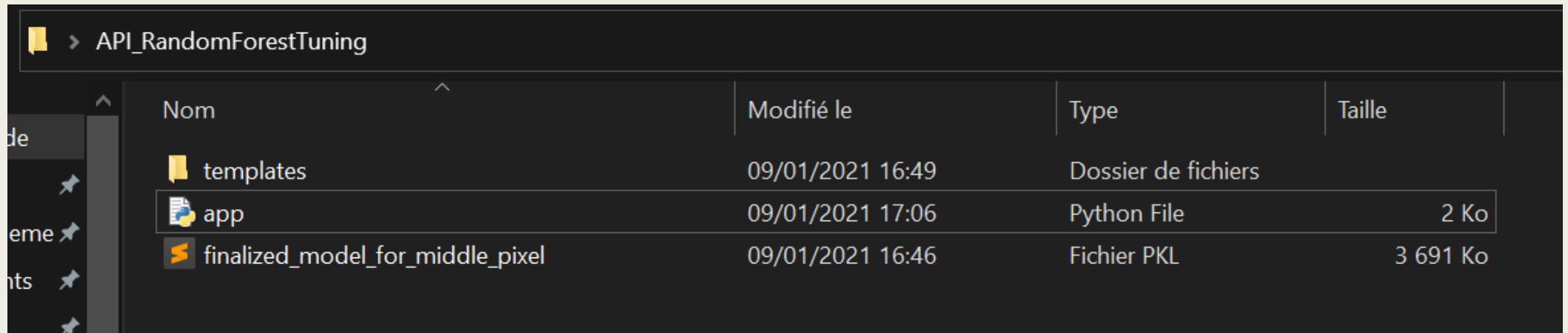
```
F1_score for Random Forest on CV data: 0.9113751380351022
```

Nous pouvons observer un léger changement de la valeur de f1_score en réglant les hyperparamètres -> f1_score de 0.9105 à 0.9114

IV) API FLASK

- Nous avons choisi de créer une API avec Flask pour ensuite l'appeler dans une simple appli web afin de permettre à l'utilisateur de rentrer les 4 valeurs du pixel sur les 4 bandes spectrales et ainsi à partir du modèle que nous avons créé pouvoir prédire la classe du pixel ainsi que la probabilité grâce au module numpy.
- Nous avons utilisé un Template HTML/CSS que nous avons déjà crée pour un ancien projet

IV) API FLASK



| API_RandomForestTuning | | | | |
|----------------------------------|------------------|---------------------|----------|--|
| Nom | Modifié le | Type | Taille | |
| templates | 09/01/2021 16:49 | Dossier de fichiers | | |
| app | 09/01/2021 17:06 | Python File | 2 Ko | |
| finalized_model_for_middle_pixel | 09/01/2021 16:46 | Fichier PKL | 3 691 Ko | |

- Le dossier « templates » contient le fichier HTML (+ du contenu CSS) pour le front de notre appli
- Le fichier app.py contient l'API Flask ainsi que la récupération de notre modèle préalablement généré par notre fichier Jupiter Notebook
- Le fichier « finalized_model_for_middle_pixel.pkl » qui correspond à notre modèle afin de prédire la classe des pixels

IV) API FLASK

Screenshot de la page de notre appli web issus de l'API Flask

The screenshot shows a web browser window with the title 'API PIXEL CLASSIFICATION'. The address bar displays '127.0.0.1:5000/predict'. Below the browser window, the page content is as follows:

PREDICTION DE LA CLASSE D'UN PIXEL

Merci de renseigner les valeurs numériques entre 0 et 255 concernant le pixel dans les champs de texte suivants :

Le pixel correspond à sol gris (classe 3) avec un probabilité de 92.28 %