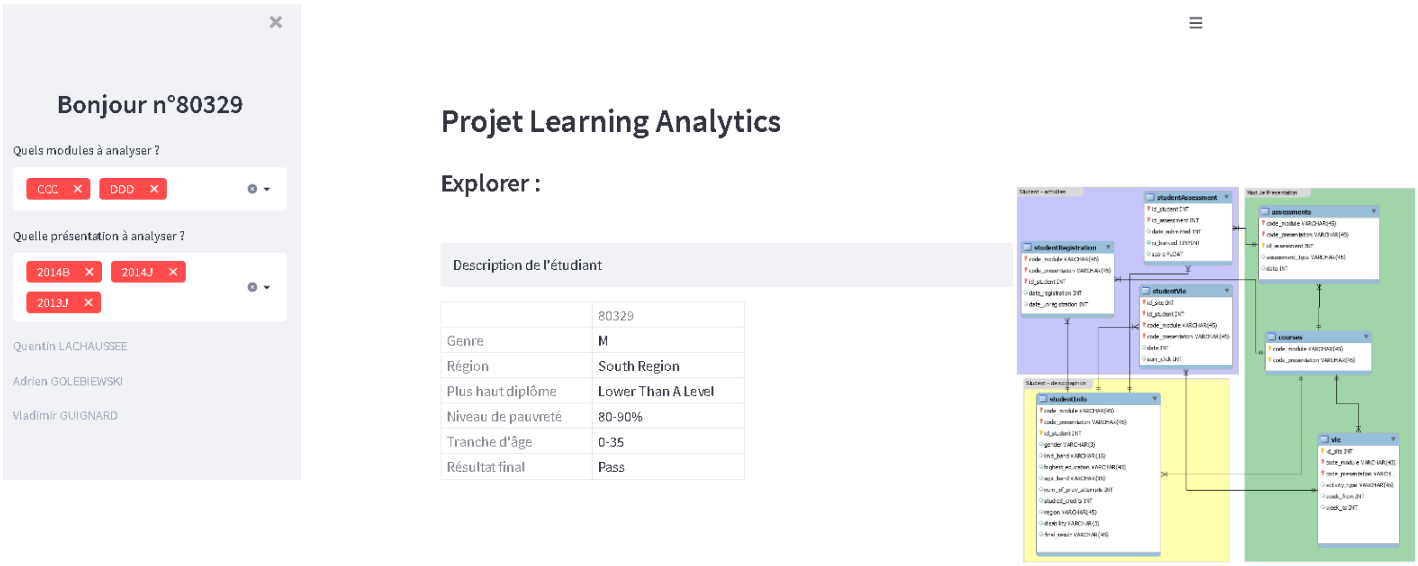
UE Machine Learning

–

Projet Learning Analytics

GOLEBIEWSKI ADRIEN – GUIGNARD VLADIMIR – LACHAUSSEE QUENTIN



Conservatoire national des Arts et Métiers – 2021/202

Introduction et objectifs du sujet

Depuis plusieurs années, l'enseignement en ligne sous forme de MOOC (Massive Open Online Courses) ou autres plateformes en ligne gagne en popularité. Le concept de matériel pédagogique mis à disposition gratuitement a séduit des millions de personnes partout dans le monde. Ces plateformes ont le potentiel d'améliorer la façon dont nous apprenons en ligne, de réduire le coût de l'éducation, de rendre l'éducation accessible et bien plus encore.

L'un des principaux avantages des plateformes en ligne par rapport aux cours traditionnels des institutions physiques est la quantité et la granularité des données qu'ils génèrent et enregistrent. Ces données sont sources d'information et peuvent être analysées pour apporter des informations aux étudiants et utilisées pour former des modèles capables de prédire leur échec / abandon.

C'est dans ce contexte que nous avons eu à étudier le jeu de données OULAD (Open University Learning Analytics Dataset). Il contient des données sur les cours, les étudiants et leurs interactions avec l'environnement d'apprentissage virtuel (VLE) de l'Open University pour sept cours différents.

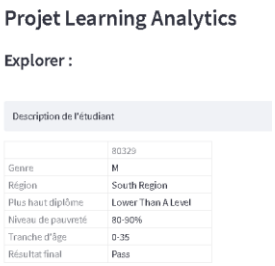
Dans le cadre de ce projet, nous visons à explorer les outils et les approches d'analyse descriptive, de clustering et de prédiction sur l'ensemble de données OULAD afin d'aborder le problème de suivi des activités et de prédiction d'échec / retrait des étudiants de la plateforme. Cet outil est destiné à l'élève lui permettant de donner une vision d'ensemble de ses activités et de son statut au sein du Virtual Learning Environment (VLE).

Notre objectif est ainsi triple :

- Définir une interface permettant à l'étudiant de se connecter à notre dashboard (avec son numéro id et son mot de passe)
- Définir pour l'étudiant :
    - Sa fiche descriptive
    - En fonction du module et d'une « presentation » donnée :
        - Son nombre de clics moyens
        - Sa note moyenne par module
        - Sa note moyenne par module selon la date, le type d'évaluation
    - Et en fonction d'une présentation et d'un module donné :
        - Une prédiction de son résultat final
    - Enfin, de manière globale, permettre à l'élève de connaitre
        - La répartition de succès et d'échec de ses collègues (autres étudiants) en fonction du niveau de pauvreté.

Ces informations fournies sous différentes formes à l'étudiant, seront disponibles sous formes de tableau de bord à l'aide de la technologie **StreamLit** de l'outil Python.

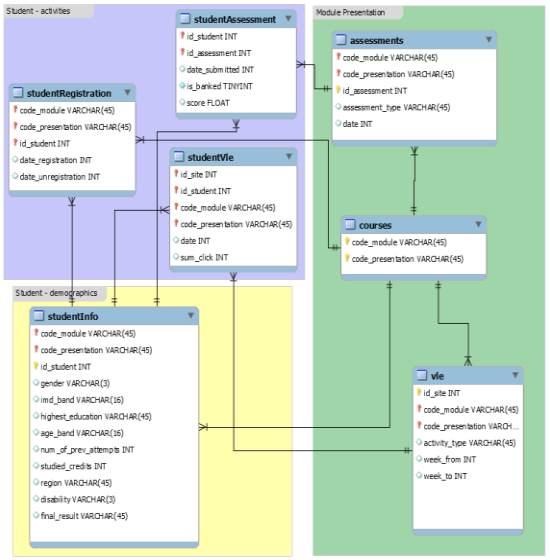
StreamLit est une application open-source permettant de créer des dashboards qui peuvent intégrer aisément des modèles de machine learning et des outils de visualisation de données. Codée avec le langage Python, le développement et l'administration de l'application se voit être à la fois souple et robuste.



Data Preparation – Création de nos bases de données

L'ensemble de nos données OULAD, réparti dans 7 fichiers Excel, se décline en trois catégories :

- les informations sur les étudiants,
- les données d'évaluation « assessments »
- les données VLE (environnement d'apprentissage virtuel).



La figure de droite illustre l**a structure globale de l'ensemble de données fourni**. Ce jeu de données est orienté sur les étudiants plutôt que sur le cours dans son ensemble. Par conséquent, la table/fichier centrale est 'studentInfo', qui est liée aux cours (un étudiant peut avoir plus d'un cours enregistré), 'student_registration' contient des informations sur les dates d'enregistrement et de désenregistrement. 'studentVle' contient les enregistrements des interactions des étudiants avec le système VLE. Chaque cours contient plusieurs évaluations (fichier/table assessment), qui sont reliées à l'étudiant par la table student_assessment qui contient les résultats des évaluations des étudiants

Dans notre interface de développement Python, nous avons construit ainsi **un dictionnaire** regroupant et appelant l'ensemble des fichiers Excel préalablement définis.

```
# on stocke les références
dataset_dict = {
    'assessments': assessments_df,
    'courses': courses_df,
    'studentAssessment': studentAssessment_df,
    'studentInfo': studentInfo_df,
    'studentRegistration': studentRegistration_df,
    'studentVle': studentVle_df,
    'vle': vle_df
}
pickle.dump(dataset_dict, open("dataset_dict.p", "wb"))
```

Grâce aux fonctions « load » et « dump » de la librairie « **Pickle** » de Python, nous pouvons, à notre guise, enregistrer et appeler ce dictionnaire dans n'importe quel script Python.

Pour répondre à nos objectifs nous nous sommes fixé deux stratégies « data » :

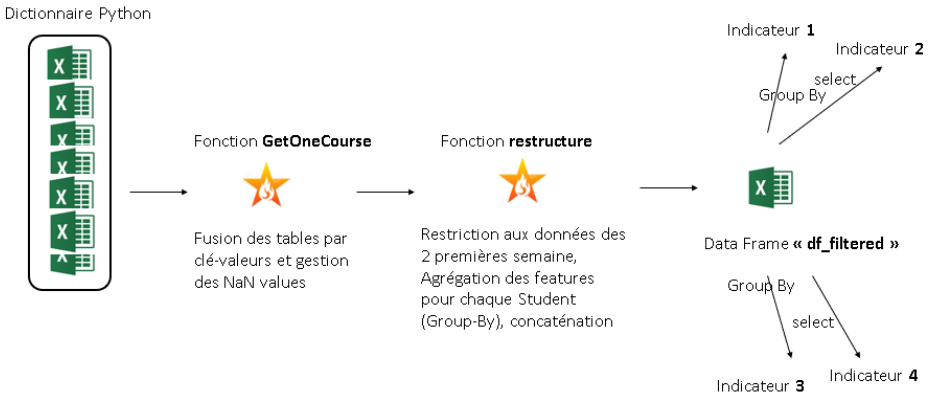
- 1ère stratégie pour définir un data frame comme base de calcul des indicateurs

- 2ème stratégie pour définir un dataframe et des variables explicatives pour notre prédiction

1ère Stratégie – Indicateurs

La 1ère stratégie nous mène à concevoir notre 1er data frame exclusivement dédié à concevoir nos indicateurs de visualisation. Les étapes menant à ce 1er data frame sont résumées ci-dessous :

Notre dictionnaire de fichiers excel constitue notre base de départ à laquelle on applique plusieurs fonctions pour mettre en forme notre premier data frame :



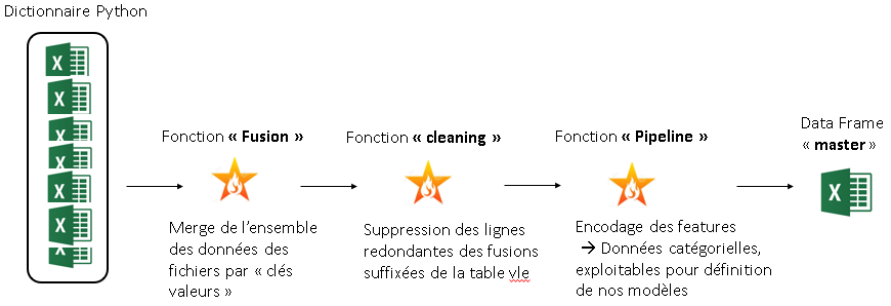
La fonction « **GetOneCourse** » fusionne toutes les tables par leurs clés primaires pour un seul cours (**critères d'unicité**) et remplace les NaNs valeurs des dates Exam/unregistration manquantes avec module_presentation_length

La fonction « **restructure** » ne conserve que les données des deux premières semaines. En effet, la prédiction et les indicateurs ne sont intéressants que lorsqu'il s'agit du début du cours, et non de la fin. Les 14 premiers jours ont donc été ciblés. De plus, nous avons supprimé ceux qui se sont désinscrits avant le début car nous ne pouvons rien faire pour eux. Cette première étape laisse ensuite place à un processus « group by » pour chaque étudiant, afin d'agréger les features. Enfin nous correspondons une ligne de notre data frame a qu'un seul élève.

Le **data Frame final** constitue ainsi une base solide pour calculer nos indicateurs souhaités à l'aide de fonction python spécifiques.

2ème Stratégie – Prédictions

La 2ème stratégie nous mène à concevoir notre 2ème data frame exclusivement dédié à concevoir nos prédictions par machine Learning. Les étapes menant à ce 2er data frame sont résumées ci-dessous :

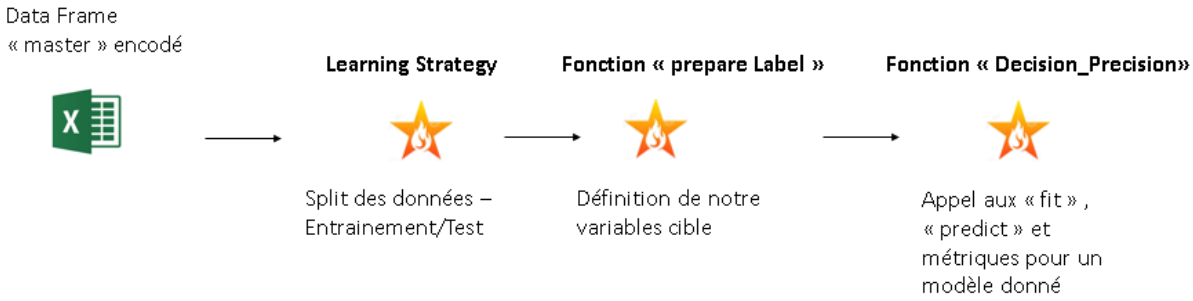


Les phases classiques de cleaning (suppression des données redondantes) et de pipeline (encodage des données catégorielles) sont réalisées.

Le **data Frame final** constitue ainsi une base solide pour réaliser nos étapes de préparation et de construction de nos modèles d'apprentissage.

Phase Prédictive – machine Learning

Cette phase de prédiction s'est appuyée tout d'abord sur le data frame « master », un des deux data frame issu de la phase de « data preparation » définie précédemment.



La première étape est la séparation de notre ensemble de données/labels en : données/labels d'apprentissage « train » (80%) et données/labels « test » (20%).

```
# Crer un  train test split
train_set, test_set = train_test_split(master, test_size=0.2, random_state=20)

# Séparer les étiquettes
train_values, train_labels = split_labels(train_set)
test_values, test_labels = split_labels(test_set)

all_values, all_labels = split_labels(master)
student_set = all_values.reset_index(drop=True)[all_values.reset_index(drop=True).id_student==id_student]
student_set = student_set[(student_set.code_module.isin(code_module)) & (student_set.code_presentation.isin(code_
student_index = student_set.index
```

La deuxième étape est d'appliquer notre fonction « prepare_label » aux labels de nos variables. Notre variable cible est la réussite ou non de l'étudiant. Comme nous ne prévoyons que des succès et des échecs, nous réétiquetons la mention 'distinction ' en tant que réussite et retrait 'withdrawn' comme échec. Nous utilisons **1** pour représenter **la réussite** et **0** pour **l'échec** pour les fonctions de la métrique de notation

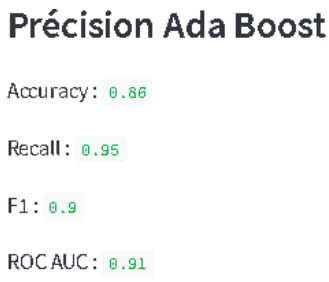
```
def prepare_labels(labels):
    # Comme nous ne prévoyons que des succès et des échecs, nous réétique
    # réussite et retrait comme échec
    # Nous utilisons 1 pour représenter la réussite et 0 pour l'échec pou
    lab_dict = {'Pass': 1, 'Fail': 0, 'Withdrawn': 0, 'Distinction': 1}
    return labels.replace(lab_dict)
```

Enfin **la 3ème étape** est l'application de notre fonction « decision_prediction » sur un modèle de Machine Learning donné afin de retourner :

- Score d'accuracy (métrique d'évaluation du modèle)
- Matrice de confusion
- Courbe Roc

```
if model_to_show == "Arbre":
    model = load_model("Best DecisionTreeClassifier")
    decision_precision(model_to_show, model, student_se
elif model_to_show == "Forêt aléatoire":
    model = load_model("Best RandomForestClassifier")
    decision_precision(model_to_show, model, student_se
elif model_to_show == "K Voisins":
    model = load_model("Best KNeighborsClassifier")
    decision_precision(model_to_show, model, student_se
elif model_to_show == "Ada Boost":
    model = load_model("Best AdaBoostClassifier")
    decision_precision(model_to_show, model, student_se
```

Nous avons sélectionné **4 modèles de Machine Learning** à tester sur nos ensembles d'apprentissage et de test :



Ces 4 modèles ont été choisis car ils nous sont familiers et ont été testés au cours de nos 3 ans études au CNAM au travers d'autres projets.

Dans notre dashboard, il sera indiqué s'il y a réussite ou échec selon les modules auxquels l'étudiant est inscrit et a déjà passé au moins 1 évaluation dans ce module. **La prédiction évolue donc en fonction du module choisi pour un étudiant donné.**

```
try:
    load_model("Best RandomForestClassifier")
    modeles = ("Forêt aléatoire", "Ada Boost", "K Voisins", "Arbre")
except:
    modeles = ("Ada Boost", "K Voisins", "Arbre")

st.subheader("Choisir un modèle :")
st.markdown(f"<h6 style='color:yellow; margin-bottom:-25px;'>Conseil : '{modeles[0]}' présente le meilleur taux de réussite.
model_to_show = st.selectbox("", modeles)
```

D'un point de vue algorithmique, on précise que tel ou tel modèle (selon le module et l'étudiant) est le plus pertinent, c'est-à-dire celui qui a une meilleure métrique de précision.

Présentation et organisation du tableau de bord

A) Frontend

Pour rappel, nous avons fait le choix d'utiliser l'application « **StreamLit** », cette solution nous permet de proposer :

- Un moyen efficace de créer et partager gratuitement une application,
  ➔ Pour **accompagner** au plus près ces utilisateurs.

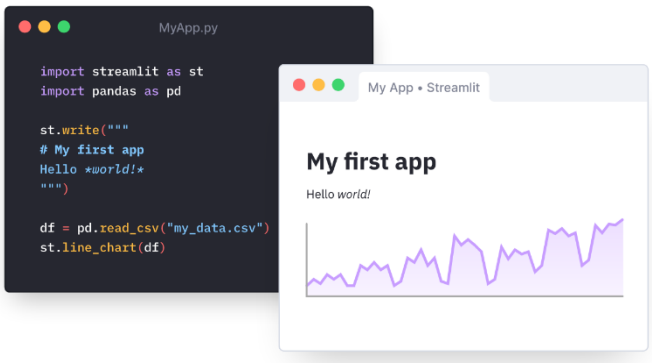
- Un accès par identifiant et mot de passe,
  ➔ Pour une **expérience unique** selon l'individu.

- Un outil 100% en Python,
  ➔ Pour faciliter et **approfondir l'intégration** de modèles de Machine Learning.

- Une approche différente de la plupart des outils de visualisation,
  ➔ Pour **se distinguer** des autres livrables avec des visualisations entièrement personnalisables.

L'application s'implémente en python en important la librairie du même nom « streamlit ».

En l'appelant, on peut y instancier des variables de type « Widgets » (traductible par « éléments graphiques » comme des vignettes ou des courbes…) et les afficher simplement comme-ci dessous :



A l'aide des nombreux widgets que propose la librairie « streamlit » et la direction de notre projet, nous avons opté pour une application construite autour de 4 grandes parties :

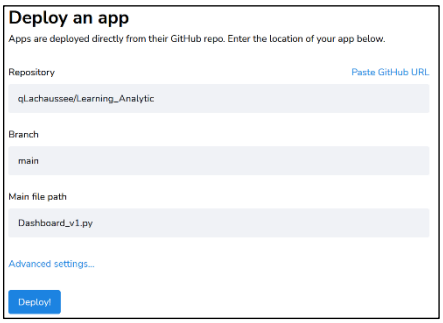
1. <u>La connexion</u>

Pour accéder à l'application, il faut un accès internet, et spécifier cet URL dans la barre de recherche :

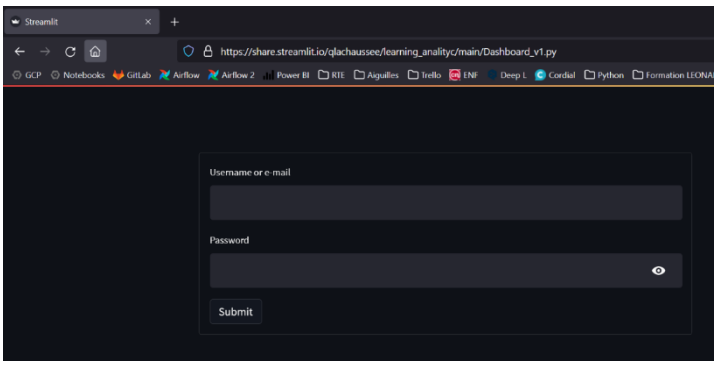
https://share.streamlit.io/qlachaussee/learning_analityc/main/Dashboard_v1.py

Cette URL a été créé à l'issue d'une demande de déploiement adressé directement au support technique de l'application StreamLit.
Il aura fallu remplir un formulaire en expliquant les raisons de ce déploiement, et finir par spécifier le répertoire GitHub où est entreposé le projet (codes et fichiers de données) :

https://github.com/qLachaussee/Learning_Analytic



Apparait alors cette page d'accueil (en mode clair ou sombre selon la préférence de l'étudiant) :

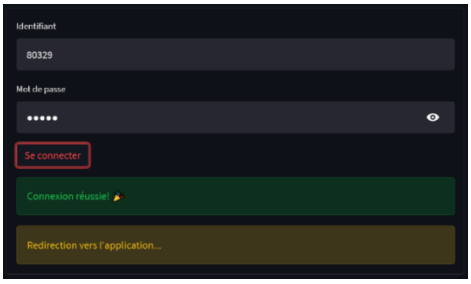


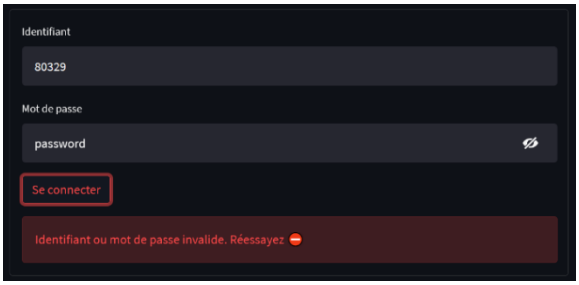
Pour se connecter à l'application, il faut spécifier :

- ✓ Un identifiant (qui correspond à l'ID d'un étudiant de la base de données OULAD),
  ➔ La liste des ID est disponible simplement dans le csv « list_student.csv »

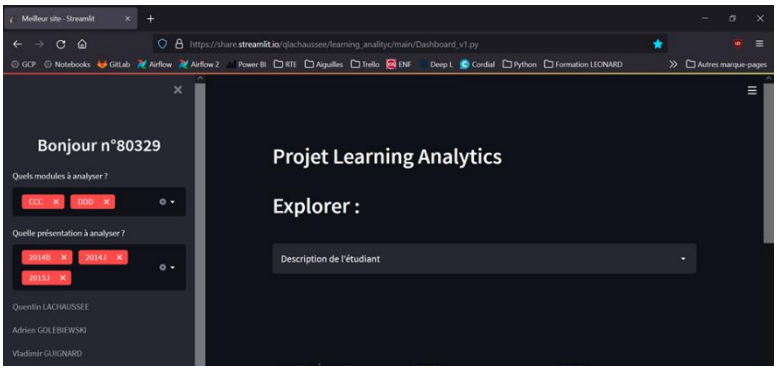
- ✓ Du mot de passe « 20/20 » (qui correspond à la note que ce projet mérite),

➔ En phase de bêta-test, le mot de passe est le même pour tous (« mdp »), mais pourra être redéfini simplement à l'avenir.

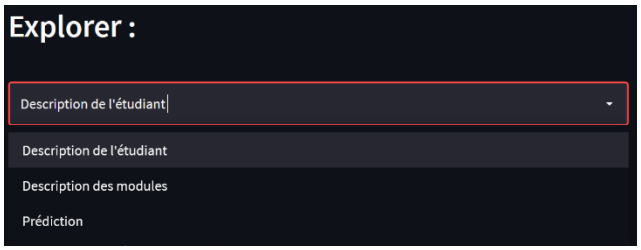




Une fois connecté, vous serez redirigé vers cette page :

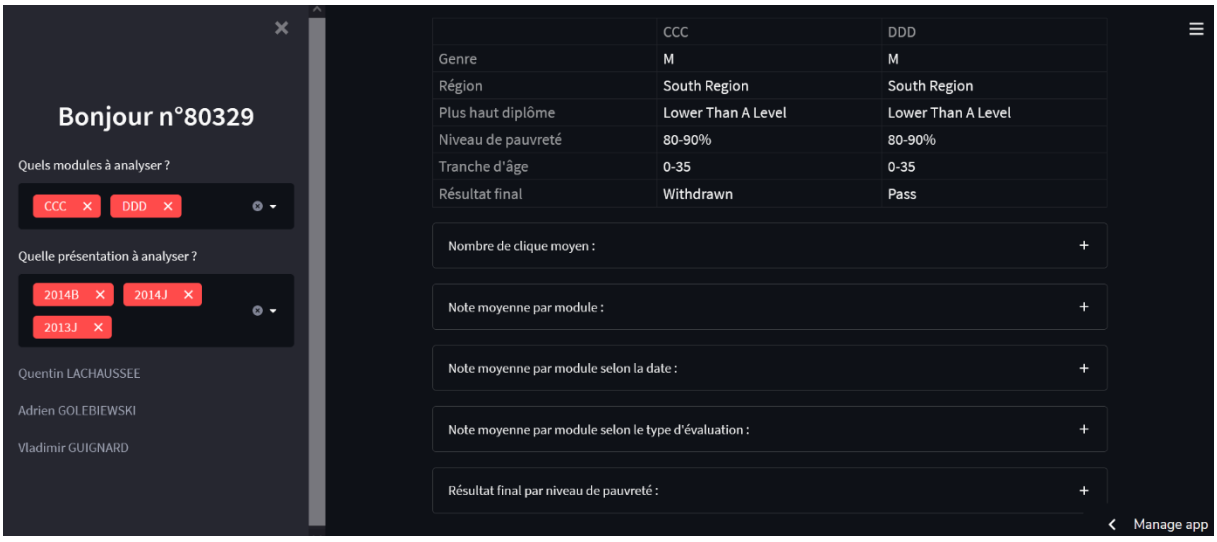


Cette page va vous proposer d'explorer les données selon 3 onglets différents. Toutes les données visualisables de cette page peuvent être filtrées selon le module et la présentation (filtres **rouges** à gauche de l'écran). A l'étudiant de choisir selon vers quelles informations il souhaite se diriger :

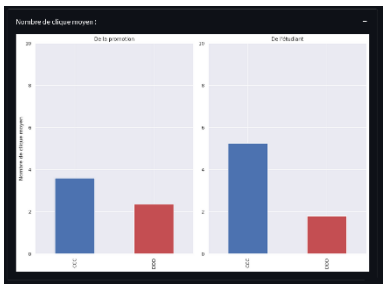
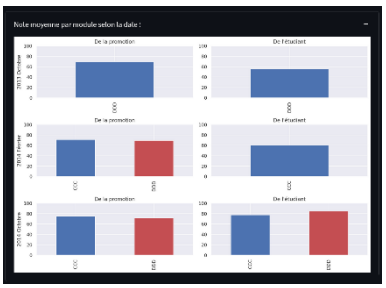
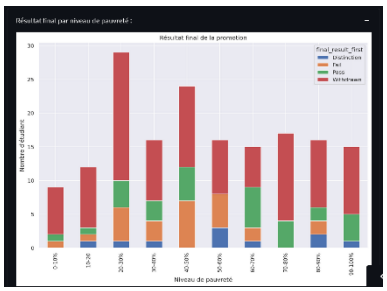


2. La description de l'étudiant

Si l'étudiant choisit « Description de l'étudiant », il obtiendra un résumé des informations différents sur ses modules et aura le choix entre 5 graphiques à afficher ou non :



Exemples de graphiques décrivant l'activité de l'étudiant : histogrammes, diagramme en barres uni/multi variables etc …

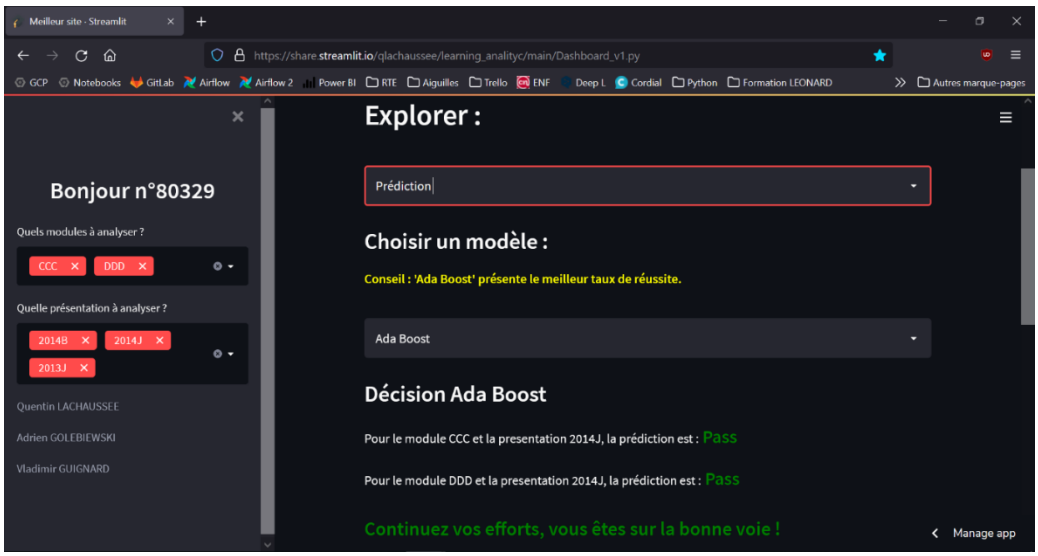
3. <u>La description des modules de l'étudiant</u>

Si l'étudiant choisit « Description des modules », il aura le choix entre 3 graphiques à afficher ou non :

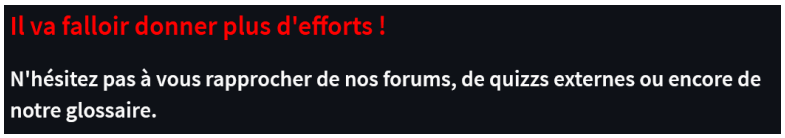
Ces graphiques lui renseignent son activité de clic, ses notes sur un module donné et des critères bien spécifiques (date, type d'évaluation).

4. <u>Prédiction</u>

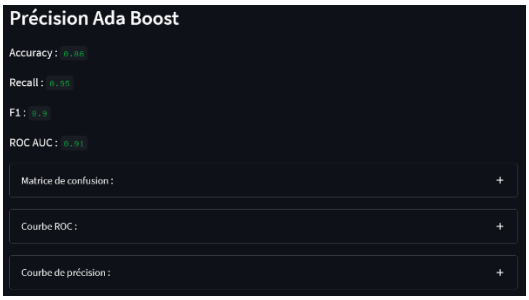
S'il choisit « Prédiction », il aura le droit à une prédiction de sa réussite ou non dans l'ensemble des modules auxquels il s'est inscrit (et déjà passé au moins 1 évaluation). Une note lui indiquera parmi les modèles de prédiction construits lequel est le plus performant. L'étudiant cliquera donc sur le modèle qui prédit avec la meilleure précision.



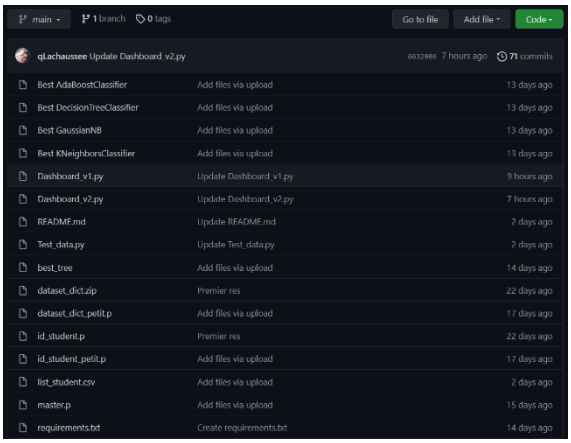
Si la prédiction est négative, des conseils peuvent vous être donnés en fonction de vos interactions avec l'environnement d'apprentissage virtuel (VLE) de l'Open University.



Vous pourrez également changer de modèle, afin d'estimer le degré de certitude des prédictions.
Des précisions sur les modèles sont également présentées :l'étudiant aura le choix entre 3 graphiques à afficher ou non pour détailler la prédiction :



B) Backend



Pour rappel, le projet est entièrement entreposé dans le répertoire GitHub :
https://github.com/qLachaussee/Learning_Analytic

L'application StreamLit cible le fichier « **Dashboard_v1.py** » qui est une version allégée du Dashboard_v2.py » avec moins d'étudiants dans la base de données à cause d'une limite de stockage sur GitHub (seuls les étudiants inscrits dans au moins 2 modules ont été conservés dans les fichiers caractérisés par le suffixe « **petit** »).
Les modèles de classification sont identifiables par le préfix « **Best** ».

Le fichier « **requirements.txt** » permet à l'application StreamLit d'importer les librairies python nécessaires au bon fonctionnement du code. Tous les fichiers de données ou modèles de classifications sont générés dans « **Test_data.py** ».

```python
if __name__ == '__main__':
    # Initialization
    if 'logged_in' not in st.session_state:
        print("noo")
        with st.form(key='login_form'):
            if "username" not in st.session_state:
                my_user = st.text_input("Identifiant")
                password = st.text_input("Mot de passe", type="password")
            submit_button = st.form_submit_button(label="Se connecter")

            if submit_button:
                if my_user.lower() in users and users[my_user.lower()] == password:
                    st.success("Connexion réussie! :tada:")
                    st.session_state.logged_in = True
                    st.session_state.key = 'OK'
                    st.session_state.id_student = my_user
                    with st.spinner("Redirection vers l'application..."):
                        st.session_state.data = pickle.load(open("dataset_dict_petit.p", "rb" ))
                        time.sleep(1)
                        print("okkkkkk")
                        st.experimental_rerun()
                        # st.experimental_rerun()
                else:
                    st.error("Identifiant ou mot de passe invalide. Réessayez :no_entry:")

    else:
        main()
```

Lors de la connexion, on va aller charger les données et les conserver sous forme de session :

Et c'est dans la fonction « main() » qui est importée depuis « Dashboard_v2.py » que l'on va utiliser les données conservées dans la session pour afficher les modules et les présentations qui correspondent à l'étudiant qui s'est identifié

```python
def main():

    st.set_page_config(page_title="Meilleur site", page_icon=":mortar_board:")
    st.header("Projet Learning Analytics")

    dataset_dict = st.session_state.data

    id_student = int(st.session_state.id_student)
    st.sidebar.markdown(f"<p style='text-align:center; font-size:1.8rem;'><b>Bonjour n°{id_student}</b></p>", unsafe_allow_html=True)

    st.balloons()

    student_registration = dataset_dict["studentRegistration"][dataset_dict["studentRegistration"]["id_student"] == id_student]

    all_module = student_registration["code_module"].unique()
    code_module = st.sidebar.multiselect("Quels modules à analyser ?", all_module, default=all_module)

    if not code_module:
        st.warning("Veuillez sélectionner un ou plusieurs module(s)")
        st.stop()

    all_presentation = code_presentation = student_registration[student_registration["code_module"].isin(code_module)]["code_presentation"].unique()
    code_presentation = st.sidebar.multiselect("Quelle présentation à analyser ?", all_presentation, default=all_presentation)

    if not code_presentation:
        st.warning("Veuillez sélectionner une ou plusieurs présentation(s)")
        st.stop()

    df_filtered_MP = getOneCourse(dataset_dict, code_module, code_presentation)
    df_filtered_MP = restructure(df_filtered_MP, 14)
    df_filtered_MPS = df_filtered_MP.reset_index()[df_filtered_MP.reset_index().id_student==id_student]
```

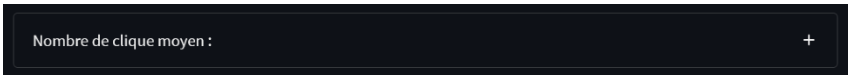
Une fonctionnalité StreamLit permet également de **garder en mémoire le résultat d'une fonction,** permettant ainsi d'éviter les chargements superflus (par exemple la table « vle », volumineuse, qui ne charge désormais qu'une fois, à la connexion) :

```python
@st.experimental_memo(suppress_st_warning=True, show_spinner=False)
def getOneCourse(dataset_dict, code_module, code_presentation):...
```

Une autre façon de diminuer le temps de chargement sont les widgets « expander » :

```python
    with st.expander("Nombre de clique moyen :"):
```

Ces widgets permettent de ne charger les graphiques uniquement si l'utilisateur veut les voir.



De manière générale, les fonctionnalités de la librairie StreamLit nous ont permis de construire une interface interactive proposant à l'étudiant une expérience immersive sur ses activités sur le VLE.