Sujet : Est-il possible de prédire la réussite d'un apprenant en analysant ses traces numériques au sein de la plateforme ARCHE ?

Jean-Claude Faber

Table des matières

1. Structure du projet

- Détermination de l'algorithme principal
- Règle à respecter pour chaque bloc de programmation
- Description de chaque chaque bloc de programmation
- Conclusion

2. Règle à respecter pour chaque blocs :

Gestion des erreurs :
Chaque bloque dont c'est nécessaire va renvoyer le couple (None,None) au bloc suivant pour avoir une propagation des erreurs .

3. Détermination de l’algorithme principal :

Algorithme : Programme_Principal_Analyse

Variables :

- df_logs_brut, df_notes_brut : Tableaux (DataFrames)
- df_logs_clean,df_notes_clean : Tableaux (DataFrames)
- df_final: Tableau (DataFrame de synthèse avec indicateurs)
- mon_arbre: Modèle (Structure d'arbre de décision)

DÉBUT

1. **ÉTAPE 1 : Chargement**
    - AFFICHER "Début du chargement des fichiers..."
    - Appeler l'algorithme secondaire charger_donnees()
    - Stocker les retours dans (df_logs_brut, df_notes_brut)
2. **ÉTAPE 2 : Nettoyage**
    - SI df_logs_brut et df_notes_brut ne sont pas nuls ALORS :
        - AFFICHER "Nettoyage des données en cours..."
        - Appeler l'algorithme secondaire nettoyage_donnees avec (df_logs_brut, df_notes_brut)
        - Stocker les retours dans (df_logs_clean, df_notes_clean)
    - SINON :
        - AFFICHER "Abandon : fichiers sources introuvables."
        - STOPPER l'algorithme
    - FIN SI
3. **ÉTAPE 3 : Calcul des Indicateurs**
    - SI df_logs_clean n'est pas nul ALORS :
        - AFFICHER "Calcul des indicateurs de suivi..."
        - resultat <---- Appeler l'algorithme calculer_indicateurs avec (df_logs_clean, df_notes_clean)
    - SINON :

- AFFICHER "Abandon : le nettoyage a échoué."
- STOPPER l'algorithme
- FIN SI

4. ÉTAPE 4 : Exploration et Visualisation

- Appeler exploration_data(df_logs_clean, df_notes_clean)
- Appeler voir_les_absents(df_logs_clean, df_notes_clean)
- Appeler afficher_graphiques(df_final) (Génération des Scatter Plots)

5. ÉTAPE 5 : Analyses Statistiques et Prédictives (Ce qu'il manquait)

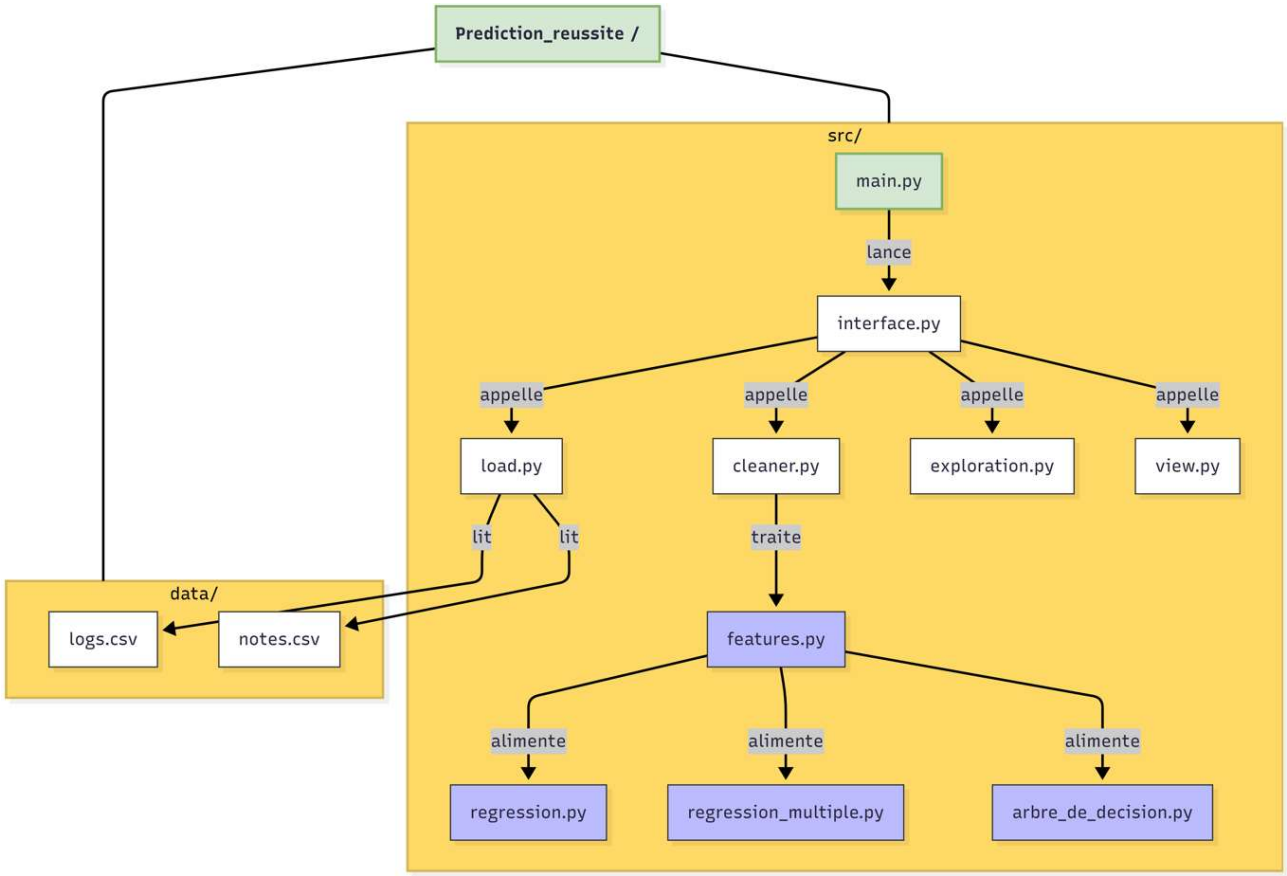
- **A. Régression Simple :**
  - Appeler analyser_regressions(df_final) (Calcul des corrélations r et r^2 par indicateur)
- **B. Régression Multiple :**
  - Appeler analyser_regression_multiple(df_final) (Modèle global $Y = X\beta+\varepsilon$)
- **C. Arbre de Décision :**
  - mon_arbre <---- Appeler entrainer_arbre_decision(df_final) (Classification Réussite/Échec)

6. ÉTAPE 6 : Affichage final

- AFFICHER "Analyse terminée avec succès."
- AFFICHER Statistiques de précision du modèle.

FIN

4. Schéma de l’algorithme principal :



5. Fichier de chargement des datas (load.py) :

Installation du package pandas qui est nécessaire pour la lecture des fichiers csv

- package pandas 3.0.0

Le module load.py assure l'acquisition sécurisée des données en localisant les fichiers logs.csv et notes.csv dans le répertoire dédié. Il utilise la bibliothèque Pandas pour convertir ces ressources brutes en structures de données exploitables (DataFrames). Une gestion d'erreurs est intégrée pour détecter les fichiers manquants, vides ou verrouillés par un autre programme. Enfin, il confirme le succès de l'opération en affichant le volume de lignes lues, ce qui permet de garantir l'intégrité de la base de travail pour les étapes suivantes.

load.py

Entrées : les chemins sont fixés en interne

Sorties : df_logs(Tableau), df_notes(Tableau) ou Nul en cas d'erreur

DÉBUT

1. **Initialisation des chemins :**
    - chemin_logs <---- "../data/logs.csv"
    - chemin_notes <---- "../data/notes.csv"
2. **Vérification de l'existence physique des fichiers :**
    - SI (le fichier chemin_logs n'existe pas) OU (le fichier chemin_notes n'existe pas) ALORS :
        - SI chemin_logs est absent : AFFICHER "ERREUR : Fichier logs manquant"
        - SI chemin_notes est absent : AFFICHER "ERREUR : Fichier notes manquant"
        - RETOURNER (Nul, Nul)
    - FIN SI
3. **Tentative de lecture (Bloc de protection) :**
    - **ESSAYER :**
        - Ouvrir et lire le contenu de chemin_logs <---- stocker dans df_logs
        - Ouvrir et lire le contenu de chemin_notes <---- stocker dans df_notes
        - AFFICHER "Chargement réussi"
        - AFFICHER le nombre de lignes lues pour chaque fichier
        - RETOURNER (df_logs, df_notes)

4. **Gestion des erreurs (Exceptions) :**
    - **SI** "Fichier non trouvé" (FileNotFoundError) :
        - AFFICHER "Vérifiez que le dossier 'data' est présent"
    - **SINON SI** "Fichier vide" (EmptyDataError) :
        - AFFICHER "Erreur : Le fichier CSV est vide"
    - **SINON SI** "Problème de permission" (PermissionError) :
        - AFFICHER "Erreur : Le fichier est ouvert ailleurs. Fermez-le."
    - **SINON** (Erreur inconnue) :
        - AFFICHER "Erreur imprévue : [Détails de l'erreur]"
    - **FIN SI**
5. **Fin de secours :**
    - RETOURNER (Nul, Nul)

FIN

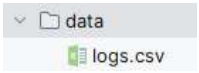
fichier load.py :

```
import pandas as pd
import os

# chemin vers les fichiers
def charger_donnees():
    chemin_logs= "../data/logs.csv"
    chemin_notes="../data/notes.csv"

# message d'erreur si on ne trouve pas les fichiers
    if not os.path.exists(chemin_logs) or not os.path.exists(chemin_notes):
        if not os.path.exists(chemin_logs):
            print ("ERREUR : Le fichier des logs est manquant")
            print ("Vérifie que tu as bien copié les fichiers dans data")
        if not os.path.exists(chemin_notes):
            print("ERREUR : Le fichier des notes est manquant")
            print("Vérifie que tu as bien copié les fichiers dans data")
            return None, None

    else :
        try :#lecture des fichiers
            df_logs= pd.read_csv(chemin_logs)
            df_notes= pd.read_csv(chemin_notes)
            print ("les données ont été chargées avec succès!")
            print (f"Fichier logs :{len(df_logs)} lignes lues")
            print(f"Fichier notes :{len(df_notes)} lignes lues")

            return df_logs, df_notes

        except FileNotFoundError:
            print("ERREUR : Le fichier est introuvable vérifie si le dossier data est présent?")
            return None, None

        except pd.errors.EmptyDataError:
            print(" ERREUR : Le fichier CSV est vide ! ajoute le fichier")
            return None, None

        except PermissionError:
            print(" ERREUR : Le fichier est déjà ouvert dans un autre programme merci de le fermer pour que cela
fonctionne")
            return None, None

        except Exception as erreur_inconnue:
             print(f" ERREUR IMPRÉVUE lors du chargement des données: {erreur_inconnue} ")
             return None, None

if __name__ == "__main__":
    charger_donnees()
```

Gestion des erreurs de load.py :

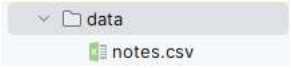
La suppression du fichier notes.csv et logs.csv donne bien :

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\load.py
ERREUR : Le fichier des logs est manquant
Vérifie que tu as bien copié les fichiers dans data
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data

Process finished with exit code 0
```



La suppression du dossier notes.csv donne bien :

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\load.py
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data

Process finished with exit code 0
```



La suppression du dossier logs.csv donne bien :

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\load.py
ERREUR : Le fichier des logs est manquant
Vérifie que tu as bien copié les fichiers dans data

Process finished with exit code 0
```

J’ai vidé le fichier notes j’obtiens bien :

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\load.py
 ERREUR : Le fichier CSV est vide ! ajoute le fichier

Process finished with exit code 0
```

j’ai ouvert en lecture le fichier notes.csv avec Windows PowerShell :
PSC\Users\Etudiant>$file=[System.IO.File]::Open("C\Users\Etudiant\PycharmProjects\Prediction_reussite\data\notes.csv", "Open", "Read", "None")

j’obtiens :

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\load.py
 ERREUR : Le fichier est déjà ouvert dans un autre programme merci de le fermer pour que cela fonctionne

Process finished with exit code 0
```

6. Fichier de nettoyage des données (cleaner.py) :

Le module cleaner.py permet d'éliminer les doublons dans le fichier des notes et supprime les logs d'heure invalides. Il réalise ensuite une jointure pour conserver les étudiants o inscrits dans le fichier des notes. Cette étape de nettoyage est sécurisée par un try … except qui interceptent les erreurs imprévues sans interrompre l'application. Des messages de validation confirment en temps réel le nombre d'étudiants traités.

cleaner.py

Entrées : df_logs(Tableau de logs), df_notes(Tableau de notes)

Sorties : df_logs_clean,df_notes_cleanTableaux nettoyés) ou Null en cas d'erreur

DÉBUT

1. **Vérification de présence :**
    - SI df_logs est vide OU df_notes est vide ALORS
        - AFFICHER "Erreur : lecture des fichiers impossible"
        - RETOURNER (Nul, Nul)
    - FIN SI
2. **Traitement des doublons dans les notes :**
    - nb_avant <----nombre de lignes de df_notes
    - df_notes_clean <---- df_notes sans les lignes identiques
    - doublons <---- nb_avant - nombre de lignes de df_notes_clean
    - SI doublons > 0 ALORS
        - AFFICHER "Nombre de doublons supprimés : ", doublons
    - SINON
        - AFFICHER "Aucun doublon trouvé."
    - FIN SI

3. **Normalisation et nettoyage des heures :**
    - POUR chaque ligne de df_logs :
        - Tenter de convertir la colonne "heure" au format Date/Heure
        - SI conversion impossible ALORS marquer comme "Vide" (NaT)
    - FIN POUR
    - Compter les lignes avec "heure" vide <---- nb_erreurs_date
    - SI nb_erreurs_date > 0 ALORS
        - Supprimer toutes les lignes où "heure" est vide
        - AFFICHER "Nombre de dates invalides supprimées : ", nb_erreurs_date
    - FIN SI
4. **Filtrage par correspondance (Jointure logique) :**
    - etudiants_valides <---- Liste des pseudos uniques présents dans df_notes_clean
    - df_logs_clean <---- Conserver dans df_logs uniquement les lignes où le pseudo est dans etudiants_valides
    - AFFICHER les statistiques de comparaison (avant vs après filtrage)
5. **Gestion des erreurs :**
    - SI une erreur imprévue survient durant le traitement :
        - AFFICHER "ERREUR IMPRÉVUE" + détails de l'erreur
        - RETOURNER (Nul, Nul)
6. **Fin :**
    - RETOURNER df_logs_clean, df_notes_clean

FIN

fichier cleaner.py :

```
import pandas as pd

# récupération des fichiers provenant de load
def nettoyage_donnees (df_logs, df_notes) :
    # si le chargement a échoué un message d'erreur apparait
    if df_logs is None or df_notes is None :
        print ("il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv")
        return None, None
    try :
        # on vérifie si il n'y a pas de duplica dans notes.csv
        nb_avant = len(df_notes)
        df_notes_clean = df_notes.drop_duplicates()
        if (nb_avant - len(df_notes_clean)) == 0:
            print(f"le nombre de doublon dans notes.csv est de {nb_avant - len(df_notes_clean)} ")
            print("il n'y a donc pas de doublon")
        elif (nb_avant - len(df_notes_clean)) > 0:
            print(f"le nombre de doublon dans notes.csv est de {nb_avant - len(df_notes_clean)} ")
            # Vérification finale des doublons
            nb_doublons_final = df_notes_clean.duplicated().sum()
            print(f"VÉRIFICATION : Nombre de doublons restants dans notes : {nb_doublons_final}")
            print("les doublons ont été supprimés")

        # conversion des dates avec coerse si une case n'est pas convertie correctement  une case vide sera mise en place
        df_logs['heure'] = pd.to_datetime(df_logs['heure'], errors='coerce')
        nb_erreurs_date = df_logs['heure'].isna().sum()
        if nb_erreurs_date == 0:
            print("Toutes les dates ont été converties correctement")
        elif nb_erreurs_date > 0:
            print(f"le nombre de date non converties est de {nb_erreurs_date} ")
            # subset cible uniquement la case heure pour les suppressions
            df_logs = df_logs.dropna(subset=['heure'])
            # Vérification finale des dates
            nb_erreurs_date_final = df_logs['heure'].isna().sum()
            print(f"VÉRIFICATION : Nombre de dates non converties (NaT) restantes : {nb_erreurs_date_final}")
            print(f"les {nb_erreurs_date} dates non converties ont été supprimées ")


        # comparaison des etudiants dans logs et notes
        nb_etudiants_notes = df_notes_clean['pseudo'].nunique()
        nb_etudiants_logs_avant = df_logs['pseudo'].nunique()

        print("Comparaison avant nettoyage")
        print(f"Étudiants dans le fichier NOTES : {nb_etudiants_notes}")
        print(f"Étudiants dans le fichier LOGS  : {nb_etudiants_logs_avant}")

        # On regarde les étudiants présent dans logs (étudiant unique)
        etudiants_avec_notes = df_notes_clean['pseudo'].unique()

        # On ne garde que les clics des étudiants qui sont dans le fichier notes
        df_logs_clean = df_logs[df_logs['pseudo'].isin(etudiants_avec_notes)]

        # Comparaison apres nettoyage
        nb_logs_final = df_logs_clean['pseudo'].nunique()

        print("Comparaison après nettoyage")
        print(f"Étudiants dans le fichier NOTES : {nb_etudiants_notes}")
        print(f"Étudiants dans le fichier LOGS  : {nb_logs_final}")
```

```
        return df_logs_clean, df_notes_clean

    except Exception as erreur_inconnue:
        print(f" ERREUR IMPRÉVUE lors du nettoyage des données : {erreur_inconnue} ")
        return None, None

if __name__ == "__main__":
    import pandas as pd

    # valeur incorrecte pour test logs et notes
    data_logstest = {
        'pseudo': ['436', '841', '436','18'],
        'heure': ['2024-07-24 09:48:08', '2024-07-24 09:48:08', 'HEURE', '2024-08-24 09:48:08'],  #  date invalide
        'contexte': ['Cours', 'Cours', 'Cours','Cours']
    }

    data_notestest = {'pseudo': ['436','436', '841'], 'note': [12, 12, 15]}

    # conversion en dataframe
    df_logs_test = pd.DataFrame(data_logstest)
    df_notes_test = pd.DataFrame(data_notestest)

    # test unitaire
    nettoyage_donnees(df_logs_test, df_notes_test)
```

Gestion des erreurs de cleaner.py :

avec `if __name__ == "__main__":`

et les fichiers de test de data avec erreur

```
data_logstest = {
    'pseudo': ['436', '841', '436','18'], # présence du doublon 436
    'heure': ['2024-07-24 09:48:08', '2024-07-24 09:48:08', 'HEURE', '2024-08-24 09:48:08'],  #  date invalide
    'contexte': ['Cours', 'Cours', 'Cours','Cours']
}

data_notestest = {'pseudo': ['436','436', '841'], 'note': [12, 12, 15]}# Etudiant 18 non présent dans data_notetest
```

Test des Doublons (Fichier Notes) :

Le doublon 436 est bien supprimé

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\cleaner.py
le nombre de doublon dans notes.csv est de 1
VÉRIFICATION : Nombre de doublons restants dans notes : 0
les doublons ont été supprimés
le nombre de date non converties est de 1
VÉRIFICATION : Nombre de dates non converties (NaT) restantes : 0
les 1 dates non converties ont été supprimées
Comparaison avant nettoyage
Étudiants dans le fichier NOTES : 2
Étudiants dans le fichier LOGS  : 3
Comparaison après nettoyage
Étudiants dans le fichier NOTES : 2
Étudiants dans le fichier LOGS  : 2

Process finished with exit code 0
```

Test des Dates Invalides (Fichier Logs) :

le format de date invalide a bien été détecté et supprimé

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\cleaner.py
le nombre de doublon dans notes.csv est de 1
VÉRIFICATION : Nombre de doublons restants dans notes : 0
les doublons ont été supprimés
le nombre de date non converties est de 1
VÉRIFICATION : Nombre de dates non converties (NaT) restantes : 0
les 1 dates non converties ont été supprimées
Comparaison avant nettoyage
Étudiants dans le fichier NOTES : 2
Étudiants dans le fichier LOGS  : 3
Comparaison après nettoyage
Étudiants dans le fichier NOTES : 2
Étudiants dans le fichier LOGS  : 2

Process finished with exit code 0
```

Test de Cohérence (Étudiants Inconnus) :

L'étudiant 18 est présent dans les logs et n'est pas présent dans les notes il doit donc être supprimé ce qui est le cas :

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\cleaner.py
le nombre de doublon dans notes.csv est de 1
VÉRIFICATION : Nombre de doublons restants dans notes : 0
les doublons ont été supprimés
le nombre de date non converties est de 1
VÉRIFICATION : Nombre de dates non converties (NaT) restantes : 0
les 1 dates non converties ont été supprimées
Comparaison avant nettoyage
Étudiants dans le fichier NOTES : 2
Étudiants dans le fichier LOGS  : 3
Comparaison après nettoyage
Étudiants dans le fichier NOTES : 2
Étudiants dans le fichier LOGS  : 2

Process finished with exit code 0
```

7. Fichier des paramètres à suivre (features.py) :

Description des paramètres de suivi :

1er Volume d'activités (nombre total d'actions)
Cela permet de déterminer la présence numérique de l'étudiant sur la plateforme.
Un volume faible peut être un signe de décrochage scolaire.

2ème Temps d'activité réel (Intervalle de 5 minutes)
Somme des durées séparant deux clics consécutifs, à condition que cet intervalle soit inférieur à 5 min.
Cela permet d'estimer le temps de travail effectif et cela filtre les périodes d'inactivité.

3ème Régularité (nombre de jour actif)
Décompte du nombre de jours uniques où au moins une action a été enregistrée.
Cela permet de visualiser la répartition de l'apprentissage.

4ème diversité des ressources consultés :
Nombre de catégories de ressources différentes consultées.
Permet de visualiser l'intérêt de l'étudiant.

5ème Etendue de l'exploration (contexte unique consulté) :
Permet de visualiser la couverture du programme

6ème Performance (nombre de tests effectués) :
Permet de compter le nombre de test réalisé en auto-évaluation

Le module features.py commence par vérifier la validité des données pour assurer un arrêt propre et sécurisé en cas de fichiers manquants, Il extrait des indicateurs clés de l'engagement des étudiants.
Les paramètres de suivi sont ensuite fusionnées avec le fichier des notes, en remplaçant les valeurs vides par des zéros pour les étudiants n'ayant jamais été sur la plateforme.
Une vérification du tableau paramètre notes est affiché pour vérification ceci est également fait pour les étudiant qui ne se sont pas connecté sur ARCHE

Entrées : df_logs_clean(Tableau des logs),df_notes_clean (Tableau des notes)

Sorties : df_final(Tableau de synthèse regroupant tous les indicateurs)

DÉBUT

1. **Vérification de sécurité :**
    - SI df_logs_clean est nul OU df_notes_clean est nul ALORS
        - AFFICHER "ERREUR : Données absentes"
        - RETOURNER Nul
    - FIN SI
2. **Calcul du Volume (Activité brute) :**
    - Compter le nombre d'occurrences de chaque pseudo dans df_logs_clean
    - Stocker dans un tableau temporaire df_volume (colonnes : pseudo, nb_clics)
3. **Calcul de la Régularité (Présence temporelle) :**
    - Extraire la date (sans l'heure) de la colonne "heure"
    - Pour chaque pseudo, compter le nombre de **dates uniques**
    - Stocker dans df_regularite (colonnes : pseudo, nb_jours)
4. **Calcul de la Diversité (Outils utilisés) :**
    - df_diversite <---- Compter le nombre de **composants uniques** par pseudo.
    - Df-etendue ←-- Compter le nombre de contexte uniques par pseudo
5. **Calcul de l'Étendue (Couverture du cours) :**
    - **TRIER** `df_logs_clean` par `pseudo` puis par `heure` (ordre chronologique).
    - Pour chaque élève, calculer l'écart de temps entre deux actions successives.
    - SI l'écart est inférieur à 300 secondes ALORS

        Garder cet écart pour la somme.
    - FIN SI
    - Faire la somme de tous les petits écarts par étudiant $\rightarrow$ df_temps.
6. **Calcul du Temps (Engagement estimé) :**
    - Trier df_logs_clean par pseudo puis par heure (ordre chronologique)
    - Pour chaque ligne, calculer l'écart de temps avec la ligne précédente
    - SI l'écart est inférieur à 300 secondes (5 minutes) :

        - Conserver l'écart
    - SINON :
        - Ignorer l'écart (on considère que c'est une pause ou une fin de session)
    - Faire la somme de ces écarts par pseudo ----> df_temps

7. **Calcul de la Performance (Évaluations) :**
    - Filtrer les logs pour ne garder que les lignes où "composant" est égal à "Test"
    - Compter ces lignes par pseudo ----> df_tests
8. **Fusion et Finalisation (Jointure) :**
    - Créer df_final en copiant le tableau des notes (df_notes_clean)
    - **Fusionner** df_final avec chacun des tableaux d'indicateurs créés ci-dessus en utilisant le pseudo comme clé commune.
    - Pour toutes les valeurs manquantes (étudiants n'ayant aucune activité) :
        - Remplacer Vide (NaN) par 0
    - AFFICHER l'aperçu et les statistiques du tableau final
9. **Fin :**

    - AFFICHER l'aperçu du tableau final et les statistiques.
    - RETOURNER df_final

FIN

fichier features.py :

```
import pandas as pd
from cleaner import nettoyage_donnees
from load import charger_donnees

def calculer_indicateurs(df_logs_clean, df_notes_clean):
    if df_logs_clean is None or df_notes_clean is None:
        print("\n[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).")
        print("    -> Le calcul des indicateurs ne peut pas être effectué.")
        return None

    print("Création du tableau avec les indicateurs de suivi")

    # ---------------------------------------------------------
    # 1. VOLUME : Nombre total de clics
    # ---------------------------------------------------------
    # Création d'une colonne de comptage de clic par étudiant
    comptage_vol = df_logs_clean['pseudo'].value_counts()
    df_volume = pd.DataFrame({'pseudo': comptage_vol.index,'nb_clics': comptage_vol.values})

    # ---------------------------------------------------------
    # 2. RÉGULARITÉ : Nombre de jours différents de connexion
    # ---------------------------------------------------------
    # Création d'une colonne  du nombre de jours différents de connexion par étudiant
    df_logs_clean['jour'] = df_logs_clean['heure'].dt.date
    comptage_reg = df_logs_clean.groupby('pseudo')['jour'].nunique()
    df_regularite = pd.DataFrame({'pseudo': comptage_reg.index,'nb_jours': comptage_reg.values})




    # ---------------------------------------------------------
    # 3. DIVERSITÉ : Nombre d'outils différents (Test, Fichier...)
    # ---------------------------------------------------------
    # Création d'une colonne  du nombre d'outils différents utilisé par étudiant
    comptage_div = df_logs_clean.groupby('pseudo')['composant'].nunique()
    df_diversite = pd.DataFrame({'pseudo': comptage_div.index,'nb_composants': comptage_div.values})

    # ---------------------------------------------------------
    # 4. ÉTENDUE : Nombre de ressources/chapitres consultés
    # ---------------------------------------------------------
    # Création du nombre de ressources/chapitres consultés par étudiant
    comptage_ete = df_logs_clean.groupby('pseudo')['contexte'].nunique()
    df_etendue = pd.DataFrame({'pseudo': comptage_ete.index,'nb_contextes': comptage_ete.values})

    # ---------------------------------------------------------
    # 5. TEMPS : Somme des sessions (Règle des 300 secondes)
    # ---------------------------------------------------------
    #  les actions par élève sont rangées par ordre chronologique
    df_logs_clean = df_logs_clean.sort_values(by=['pseudo', 'heure'])

    # calcule du temps écoulé (en secondes) entre deux clics consécutifs
    df_logs_clean['ecart'] = df_logs_clean.groupby('pseudo')['heure'].diff().dt.total_seconds()

    #  les écarts < 300s sont gardés (on ignore les pauses de plus de 300 s (5 min))
    petits_ecarts = df_logs_clean[df_logs_clean['ecart'] < 300]

    # On additionne les petits écarts pour chaque élève
    somme_tps = petits_ecarts.groupby('pseudo')['ecart'].sum()
```

```
    # Création du tableau de synthèse
    df_temps = pd.DataFrame({'pseudo': somme_tps.index,'temps_total_sec': somme_tps.values})

    #-----------------------------------------------------------
    # 6. PERFORMANCE : Nombre d'évaluations (tests)
    # -----------------------------------------------------------
    #  les lignes où l'activité est un 'test' sont captées
    df_logs_tests = df_logs_clean[df_logs_clean['composant'] == 'Test']

    # Comptage des lignes par étudiant
    comptage_tests = df_logs_tests['pseudo'].value_counts()

    df_tests = pd.DataFrame({'pseudo': comptage_tests.index, 'nb_tests': comptage_tests.values})

    # -----------------------------------------------------------
    # 7. FUSION FINALE (injection des zéros)
    # -----------------------------------------------------------
    # On part du tableau des NOTES (les 99 étudiants officiels)
    df_final = df_notes_clean.copy()

    # On fusionne chaque petit tableau un par un avec aussi les 4 qui n'ont pas de clics
    df_final = pd.merge(df_final, df_volume, on='pseudo', how='left')
    df_final = pd.merge(df_final, df_regularite, on='pseudo', how='left')
    df_final = pd.merge(df_final, df_diversite, on='pseudo', how='left')
    df_final = pd.merge(df_final, df_etendue, on='pseudo', how='left')
    df_final = pd.merge(df_final, df_temps, on='pseudo', how='left')
    df_final = pd.merge(df_final, df_tests, on='pseudo', how='left')

    # remplacement de  tous les vides (NaN) par 0 pour ceux qui ne se sont pas connectés à ARCHE
    df_final = df_final.fillna(0)

    print(f"l'étude va porter sur {len(df_final)} étudiants ")

    #  Vérifications finales
    print("VÉRIFICATION DU TABLEAU FINAL")
    print(f"Nombre total de lignes : {len(df_final)} ")
    print("\nAperçu des 5 premières lignes :")
    print(df_final.head())

    # Verification de ceux qui ne se sont pas connectés sur ARCHE sont bien présent  filtre sur nb_clics == 0
    print("\nÉtudiants qui n'ont jamais ouvert ARCHE (Vérification injection) :")
    print(df_final[df_final['nb_clics'] == 0])


    print(f"l'étude va porter sur {len(df_final)} étudiants ")
    return df_final


if __name__ == "__main__":
    # Chargement des données brutes
    df_logs_brut, df_notes_brut = charger_donnees()

    # nettoyage des données
    df_logs_clean, df_notes_clean = nettoyage_donnees(df_logs_brut, df_notes_brut)

    # calcule des indicateurs
    resultat = calculer_indicateurs(df_logs_clean, df_notes_clean)
```

Gestion des erreurs de features.py :

Suppression du fichier notes.csv

Load.py cherche les fichiers et ne les trouve pas il affiche alors le message :

ERREUR : Les fichiers sont introuvables dans le dossier data

Vérifie que tu as bien copié les fichiers dans data

Load.py retourne le couple (None , None) qui va être envoyé à l'étape suivante

Cleaner.py va recevoir les deux None qui va a son tour renvoyer le message :
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv
Cleaner.py retourne le couple (None , None) qui va être envoyé à l'étape suivante

Calculer _indicateur reçoit ces (None,none), déclenche sa propre sécurité et s'arrête proprement.

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\features.py
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
    -> Le calcul des indicateurs ne peut pas être effectué.

Process finished with exit code 0
```

8. Fichier d’exploration des données (exploration.py) :

Installation du package matplotlib 3.10.8

Le module exploration.py permet de visualiser les données nettoyées pour identifier les tendances majeures de l'activité sur ARCHE. Il quantifie les ressources consultées, dénombre les étudiants actifs et extrait le "Top 5" des actions les plus fréquentes pour comprendre l'usage global de la plateforme. Il permet de lister les étudiants présents dans le fichier des notes mais absents des logs, isolant ainsi ceux n'ayant aucune activité numérique. Le script intègre des affichages de contrôle en console pour visualiser la cohérence des effectifs avant l'étape de modélisation.

Entrées : df_logs_clean (Tableau), df_notes_clean(Tableau)

DÉBUT

1. **Calcul des statistiques :**
    - Compter le nombre de valeurs uniques dans la colonne "pseudo" de df_logs_clean.
    - Compter le nombre de valeurs uniques dans la colonne "contexte" de df_logs_clean.
2. **Affichage :**
    - AFFICHER le nombre d'étudiants actifs.
    - AFFICHER le nombre de ressources différentes consultées.
3. **Analyse de fréquence :**
    - Trier les types d'actions ("evenement") par ordre décroissant de fréquence.
    - AFFICHER les 5 premières lignes de ce classement. **FIN**

Entrées : df_logs_clean, df_notes_clean

Sorties : pseudos_absents (Une liste)

DÉBUT

1. **Préparation :**
    - liste_actifs <---- Extraire tous les pseudos uniques présents dans df_logs_clean.
    - Initialiser une liste vide pseudos_absents.
2. **Recherche des étudiants sans activité (Boucle de parcours) :**
    - POUR CHAQUE pseudo_etudiant unique présent dans df_notes_clean :
        - **SI** pseudo_etudiant n'est PAS présent dans liste_actifs **ALORS** :
            - Ajouter pseudo_etudiant à la liste pseudos_absents.
        - **FIN SI**

- FIN POUR CHAQUE

3. Extraction et Affichage :

- Créer un tableau df_resultat contenant uniquement les lignes de df_notes_clean dont le pseudo est dans pseudos_absents.
- AFFICHER le nombre total d'absents.
- AFFICHER les colonnes "pseudo" et "note" de df_resultat.
- RETOURNER pseudos_absents.

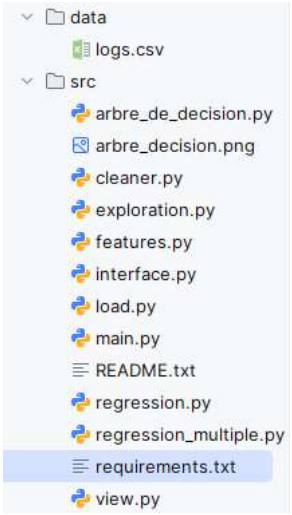
FIN

fichier exploration.py :

```
import pandas as pd
import matplotlib.pyplot as plt

def exploration_data(df_logs_clean, df_notes_clean):

    # Nombre d'élèves différents
    nb_eleves = df_logs_clean['pseudo'].nunique()
    print(f"Nombre d'étudiants actifs dans les logs : {nb_eleves} ")

    # Nombre de cours différents
    nb_cours = df_logs_clean['contexte'].nunique()
    print(f"Nombre de ressources (cours/chapitres) consultées : {nb_cours} ")

    # Top 5 des activités les plus fréquentes
    print("Top 5 des types d'actions :")
    print(df_logs_clean['evenement'].value_counts().head(5).to_string(name=False, dtype=False))


def voir_les_absents(df_logs_clean, df_notes_clean):
    print("Visaulisation des étudiant sans activité sur ARCHE")

    # liste des étudiant qui ont eu au moins une activité
    liste_actifs = list(df_logs_clean['pseudo'].unique())

    # création liste vide de ceux qui n'ont pas utilisé ARCHE
    pseudos_absents = []

    # Pour chaque étudiant présent dans le fichier des notes
    for pseudo_etudiant in df_notes_clean['pseudo'].unique():
        # si l'étudiant n'apparaît pas dans la liste des actifs
        if pseudo_etudiant not in liste_actifs:
            # Alors ajout aux absents
            pseudos_absents.append(pseudo_etudiant)

    # Affichage de ceux qui ne se sont pas connecté sur ARCHE avec leur note respective
    df_resultat = df_notes_clean[df_notes_clean['pseudo'].isin(pseudos_absents)]

    print(f"Voici les {len(df_resultat)} étudiants qui n'ont jamais ouvert ARCHE :")
    print(df_resultat[['pseudo', 'note']])

    print(" 5 premières lignes du tableau de logs nettoyé :")
    print(df_logs_clean.head(5))

    return pseudos_absents


if __name__ == "__main__":
    from load import charger_donnees
    from cleaner import nettoyage_donnees

    # récupération des données brutes (df_logs, df_notes)
    df_logs, df_notes = charger_donnees()

    if df_logs is not None:
        # récupération  des données nettoyées (df_logs_clean, df_notes_clean)
        df_logs_clean, df_notes_clean = nettoyage_donnees(df_logs, df_notes)
```

```
# exploration avec les données nettoyées
exploration_data(df_logs_clean, df_notes_clean)

# affichage des étudiants qui n'ont pas été sur arche et de leur note
absents = voir_les_absents(df_logs_clean, df_notes_clean)
```

Gestion des erreurs de exploration.py :



Lorsque je supprime le fichier notes.csv et que je lance `if __name__ == "__main__":` du module exploration .py , le message d'erreur du module load.py s'affiche car le script détecte l'absence de la ressource avant de commencer l'analyse

Exploration.py appelle la fonction charger-données() du module load.py celui-ci vérifie physiquement la présence des fichiers via os.path.exists qui se trouve dans load.py
Ce qui bloque le flux, puisque notes.csv est manquant et renvoie la valeur None.
Le module exploration.py ne contient pas de gestion d'erreurs redondante car il bénéficie de la sécrité centralisée du module load.py. En cas d'absence des fichiers sources, le module de chargement bloque le flux de données en amont

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\exploration.py
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data

Process finished with exit code 0
```

9. Fichier de régression unitaire des données (regression.py) :

Installation du package scipy 1.17.0

Le module regression.py calcule la régression linéaire simple de chaque indicateur d'activité. Il utilise la bibliothèque SciPy pour déterminer l'équation de la droite de tendance, le coefficient de corrélation et le coefficient de détermination. Ces valeurs permettent d'évaluer la correspondance entre un comportement spécifique et la note finale. Pour assurer la robustesse de l'analyse, le script traite les données nettoyées avant d'afficher les résultats détaillés en console.

Entrée : df (Le tableau final contenant les indicateurs et les notes)

Sortie : Affichage des statistiques de corrélation

DÉBUT

1. **Vérification de sécurité :**
    - SI df est inexistant (Nul) ALORS
        - AFFICHER "ERREUR : Les données sont manquantes."
        - ARRÊTER l'algorithme
    - FIN SI
2. **Initialisation :**
    - Définir la liste des indicateurs à tester : ["nb_clics", "nb_jours", "nb_composants", "nb_contextes", "temps_total_sec", "nb_tests"]
3. **Boucle de calcul (Parcours des colonnes) :**
    - POUR CHAQUE colonne présente dans la liste indicateurs :
        - **A. Extraction des variables :**
            - x <---- Valeurs de la colonne actuelle (Indicateur)
            - y <---- Valeurs de la colonne "note"
        - **B. Calcul statistique (Régression Linéaire) :**
            - Calculer la pente (slope), l'ordonnée à l'origine (intercept) et le coefficient de corrélation (r) entre x et y.
        - **C. Affichage des résultats :**
            - AFFICHER "Indicateur : [nom de la colonne]"
            - AFFICHER "Coefficient de corrélation (r) : ", r
            - AFFICHER "Coefficient de détermination (r^2) : ", r * r
            - AFFICHER "Équation : Note = [pente] * [indicateur] + [ordonnée]"
    - FIN POUR CHAQUE

FIN

fichier regression.py :

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from features import calculer_indicateurs

def analyser_regressions(df):
    if df_final is None:
        print(" ARRÊT : Les données sont manquantes. Vérifiez vos fichiers logs.csv ou notes.csv.")
        return  # Arrêt propre ici

    print("\n--- ANALYSE PAR RÉGRESSION LINÉAIRE ---")

    #  indicateurs à tester
    indicateurs = ['nb_clics', 'nb_jours', 'nb_composants', 'nb_contextes', 'temps_total_sec','nb_tests' ]

    for col in indicateurs:
        # identification des données
        x = df[col]
        y = df['note']

        # Calcul de la régression
        res = stats.linregress(x, y)

        # Affichage des résultats
        print(f"\nIndicateur : {col} ")
        print(f"  Coefficient de corrélation (r) : {res.rvalue:.2f} ")
        print(f"  Coefficient de détermination (r²) : {res.rvalue ** 2:.2f} ")
        print(f"  Équation : Note = {res.slope:.2f}  * {col}  + {res.intercept:.2f} ")


if __name__ == "__main__":
    from load import charger_donnees
    from cleaner import nettoyage_donnees

    # 1. Préparation des données (Chargement -> Nettoyage -> Indicateurs)
    df_logs, df_notes = charger_donnees()
    df_logs_clean, df_notes_clean = nettoyage_donnees(df_logs, df_notes)
    df_final = calculer_indicateurs(df_logs_clean, df_notes_clean)

    # 2, Lancement de la régression
    analyser_regressions(df_final)
```

Gestion des erreurs de regression.py :

Suppression du fichier notes.csv

Lorsque je supprime le fichier notes.csv et que je lance `if __name__ == "__main__":` du module regression.py.
Load.py retourne le couple (None,None) qui va être envoyé à l'étape suivante

Cleaner.py va recevoir les deux None qui va a son tour renvoyer le message :
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

Features.py qui reçoit (None,None) en entrée déclenche alors sa propre sécurité interne. Les données nettoyées sont absentes ce qui affiche un message d'erreur

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
-> Le calcul des indicateurs ne peut pas être effectué.

features.py revoie une valeur None unique qui permet dans regression.py d'effectuer un arrêt propre avec la vérification du paramètre d'entrée df.

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\regression.py
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
    -> Le calcul des indicateurs ne peut pas être effectué.
 ARRÊT : Les données sont manquantes. Vérifiez vos fichiers logs.csv ou notes.csv.

Process finished with exit code 0
```

10. Fichier de regression multiple (regression_multiple.py) :

Installation du package statsmodels 0.14.6

Le module regression_multiple.py permet de donner les notes en fonction des combinaisons simultanément de l'ensemble des six indicateurs d'activité (clics, temps, jours, outils, contextes et tests). Contrairement à la régression simple, il utilise la bibliothèque statsmodels pour calculer une équation globale permettant de mesurer l'impact relatif de chaque variable tout en neutralisant les autres. Le modèle fournit un coefficient de détermination global qui indique la part de la note finale expliquée par l'activité numérique sur ARCHE. Le script analyse la p-value (F-statistic) : une valeur inférieure à 0,05 confirme que le modèle est statistiquement significatif pour prédire la réussite.
Le module prend en compte la cascade de sécurité.

Entrée : df (Tableau de synthèse contenant la note et les 6 indicateurs)

Sortie : Diagnostic statistique du modèle de prédiction

DÉBUT

1. **Vérification de sécurité (Cours 3) :**
    - SI df est inexistant ALORS
        - AFFICHER "ARRÊT : Données manquantes pour la régression."
        - RETOURNER
    - FIN SI
2. **Définition du modèle mathématique :**
    - Définir la formule de calcul : la "note" est expliquée par la somme pondérée de :
        - (nb_clics + nb_jours + nb_composants + nb_contextes + temps_total_sec + nb_tests)
3. **Calcul statistique (Traitement des données) :**
    - Appliquer la méthode des **Moindres Carrés Ordinaires (OLS)** :
        - Trouver les meilleurs coefficients β (poids de chaque indicateur) pour que l'erreur entre la note réelle et la note prédite soit la plus petite possible.
    - Générer le modele fité.

4. **Analyse de la performance (R^2) :**
    - Récupérer R-squared (Coefficient de détermination).
    - AFFICHER le pourcentage de la note expliqué par l'activité sur ARCHE (R^2 x 100).
5. **Test de validité (P-Value) :**
    - Récupérer la p_value globale du modèle.
    - **SI** p_value < 0,05 **ALORS** :
        - AFFICHER "Le modèle est statistiquement significatif (fiable)."
    - **SINON** :
        - AFFICHER "Le modèle n'est pas significatif."
    - FIN SI
6. **Fin :**
    - AFFICHER le résumé complet des coefficients.

FIN

fichier regression_multiple.py :

```
import pandas as pd
import statsmodels.formula.api as smf


def analyser_regression_multiple(df):
    if df is None:
        print(" ARRÊT : Les données sont manquantes pour la régression multiple.")
        return

    # régression linéaire multiple
    print(" RÉGRESSION LINÉAIRE MULTIPLE ")


    # Equation  régression multiple :
    # Note = β0 + (β1 * clics) + (β2 * jours) + (β3 * composants) + (β4 * contextes) + (β5 * temps)+ (β6 * nombre de
test)
    formule = "note ~ nb_clics + nb_jours + nb_composants + nb_contextes + temps_total_sec+ nb_tests"

    # Calcul des meilleurs coefficients β pour minimiser l'erreur
    modele = smf.ols(formula=formule, data=df).fit()

    # Affichage du résumé statistique
    print(modele.summary())

    # Explication des formules
    print("FORMULES UTILISÉES")
    print(f"1. Modèle : Y  = Xβ + ε (où Y  est la note et X  les indicateurs)")
    print(f"2. R-squared (R²) : {modele.rsquared:.2f} ")
    print(f" Cela signifie que {modele.rsquared *  100:.0f} % de la note est expliquée par l'activité ARCHE.")

    # Interprétation de la p_value (F-statistic)
    p_value = modele.f_pvalue
    print(f"3. Prob (F-statistic) : {p_value:.4e} ")

    if p_value < 0.05:
        print(f"Comme la p-value ({p_value:.4e}) est inférieure à 0.05,")
        print(" le modèle global est statistiquement significatif.")
    else:
        print(f"Comme la p-value ({p_value:.4e}) est supérieure à 0.05,")
        print("le modèle n'est pas considéré comme statistiquement significatif.")


# Bloc de test
if __name__ == "__main__":
    from load import charger_donnees
    from cleaner import nettoyage_donnees
    from features import calculer_indicateurs
```

Gestion des erreurs de regression_multiple.py :

Si le fichier notes.csv ou load.csv est manquant, load.py retourne le couple (None,None) qui va être envoyé à l'étape suivante
Cleaner.py va recevoir les deux None qui va a son tour renvoyer le message :
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

Features.py qui reçoit (None,None) en entrée déclenche alors sa propre sécurité interne. Les données nettoyées sont absentes ce qui affiche un message d'erreur

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
-> Le calcul des indicateurs ne peut pas être effectué.

features.py revoie une valeur None unique qui permet dans regression.py d'effectuer un arrêt propre avec la vérification du paramètre d'entrée df.

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\regression_multiple.py
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
    -> Le calcul des indicateurs ne peut pas être effectué.
 ARRÊT : Les données sont manquantes pour la régression multiple.

Process finished with exit code 0
```

11. Arbre de décision (arbre_de decision;py) :

Installation du package scikit-learn 1.8.0

Le module arbre_de_decision.py utilise un classifieur supervisé de la bibliothèque sciki-learn. Pour ce modèle il est nécessaire de transformer les notes en classes binaires (échec (0) ou réussite (1)) pour identifier les profils d'étudiants. La profondeur de l'arbre est limitée à 3 niveaux ce qui permet de visualiser les seuils de décision critiques basés sur les indicateurs d'activité. Le script calcule la métrique d'accuracy (précision globale) pour mesurer l'efficacité du modèle et génère une représentation graphique. Cette méthode permet de capturer des stratégies de réussite plus complexes sur la plateforme ARCHE. Le module prend en compte la cascade de sécurité.

Entrée : df (Tableau de synthèse contenant les indicateurs et les notes)

Sortie : arbre(Modèle prédictif) et une visualisation graphique

DÉBUT

1. **Vérification de sécurité (Cours 3) :**
    - SI df est inexistant ALORS
        - AFFICHER "ARRÊT : Données manquantes."
        - RETOURNER
    - FIN SI
2. **Préparation des catégories (Classification binaire) :**
    - Initialiser une liste vide resultats
    - **POUR CHAQUE** note présente dans la colonne 'note' de df :
        - SI note >= 10 ALORS :
            - Ajouter **1** (Réussite) à resultats
        - SINON :
            - Ajouter **0** (Échec) à resultats
        - FIN SI
    - FIN POUR CHAQUE
    - Ajouter la liste resultats au tableau sous le nom de colonne "reussite".
3. **Sélection des données d'apprentissage :**
    - X (Variables explicatives) <---- Colonnes ['nb_clics', 'nb_jours', 'nb_composants', 'nb_contextes', 'temps_total_sec', 'nb_tests']
    - y (Variable à prédire) $<----Colonne "reussite"

4. **Construction de l'Arbre :**
    - Créer un objet arbre de type classifieur.
    - Limiter la profondeur à **3** (pour éviter le sur-apprentissage et garder une structure lisible).
    - **ENTRAÎNER** l'arbre en lui fournissant x et y.
5. **Évaluation du modèle :**
    - Calculer les prédictions basées sur x.
    - Comparer les prédictions avec les valeurs réelles y pour obtenir la **Précision (Accuracy)**.
    - AFFICHER la précision du modèle.
6. **Visualisation et Sauvegarde :**
    - Générer un graphique représentant les nœuds de décision (tests sur les indicateurs).
    - Colorer les nœuds selon la classe majoritaire (Échec/Réussite).
    - Sauvegarder le graphique sous le nom "arbre_decision.png".
    - AFFICHER le graphique à l'écran.
7. **Fin :**
    - RETOURNER l'objet arbre.

FIN

Fichier arbre_de_decision.py :

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score


from load import charger_donnees
from cleaner import nettoyage_donnees
from features import calculer_indicateurs


def entrainer_arbre_decision(df):
    if df is None:
        print(" ARRÊT : Les données sont manquantes pour l'arbre de décision.")
        return
    """
     Fonction transforme les notes en classes (0 ou 1)
     et entraîne un arbre de décision pour prédire la réussite.
    """
    print(" ANALYSE PAR ARBRE DE DÉCISION")

    #  colonne 'reussite' : 1 si note >= 10, sinon 0
    df_arbre = df.copy()
    # liste vide pour stocker nos 0 et 1
    resultats = []

    #  chaque note est transformée une à une
    for note in df_arbre['note']:
        if note >= 10:
            resultats.append(1)  # Réussite
        else:
            resultats.append(0)  # Échec

    # la liste est ajoutée comme une nouvelle colonne dans le tableau
    df_arbre['reussite'] = resultats

    # selection des paramètres
    indicateurs = ['nb_clics', 'nb_jours', 'nb_composants', 'nb_contextes', 'temps_total_sec', 'nb_tests']
    X = df_arbre[indicateurs]
    y = df_arbre['reussite']

    #  profondeur limité à 3 pour avoir une image  lisible
    arbre = DecisionTreeClassifier(max_depth=3, random_state=42)
    arbre.fit(X, y)

    # Evaluation
    predictions = arbre.predict(X)
    precision = accuracy_score(y, predictions)
    print(f"Précision du modèle (Accuracy) : {precision:.2f} ")

    # Visualisation de l'arbre
    plt.figure(figsize=(15, 8))
    plot_tree(arbre,
              feature_names=indicateurs,
              class_names=['Échec', 'Réussite'],
              filled=True,
              rounded=True,
```

```
            fontsize=10)
    plt.title("Arbre de Décision : Stratégies de réussite sur ARCHE")

    # Sauvegarde de l'image
    plt.savefig("arbre_decision.png")
    print("L'image de l'arbre a été sauvegardée sous 'arbre_decision.png'")

    plt.show()

    return arbre


if __name__ == "__main__":
    from load import charger_donnees
    from cleaner import nettoyage_donnees
    from features import calculer_indicateurs

     # 1. Préparation des données
    df_logs, df_notes = charger_donnees()
    df_logs_clean, df_notes_clean = nettoyage_donnees(df_logs, df_notes)
    df_final = calculer_indicateurs(df_logs_clean, df_notes_clean)

    # 2. Lancement de l'analyse
    entrainer_arbre_decision(df_final)
```

Gestion des erreurs de arbre _de décision.py :

Si le fichier notes.csv ou load.csv est manquant, load.py retourne le couple (None,None) qui va être envoyé à l'étape suivante
Cleaner.py va recevoir les deux None qui va a son tour renvoyer le message :
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

Features.py qui reçoit (None,None) en entrée déclenche alors sa propre sécurité interne. Les données nettoyées sont absentes ce qui affiche un message d'erreur

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
-> Le calcul des indicateurs ne peut pas être effectué.

features.py revoie une valeur None unique qui permet dans arbre_de_decision.py d'effectuer un arrêt propre avec la vérification du paramètre d'entrée df.

```
C:\Users\Etudiant\PycharmProjects\Prediction_reussite\.venv\Scripts\python.exe C:\Users\Etudiant\PycharmProjects\Prediction_reussite\src\arbre_de_decision.py
ERREUR : Le fichier des notes est manquant
Vérifie que tu as bien copié les fichiers dans data
il n'est pas possible de nettoyer les données car il y a un souci pour la lecture des fichiers logs.csv et notes.csv

[!] ERREUR (features.py) : Les données nettoyées sont absentes (None).
    -> Le calcul des indicateurs ne peut pas être effectué.
 ARRÊT : Les données sont manquantes pour l'arbre de décision.

Process finished with exit code 0
```

12. Interface (interface.py) :

Le module interface.py centralise l'ensemble du projet via une interface graphique développée avec Tkinter. Il orchestre le flux de travail, chaque bouton de l'interface appelle une fonction spécifique issue des modules techniques :

- Analyse des logs de connexion
- Charger & nettoyer les données
- Calculer les indicateurs
- Explorer les données
- Afficher les graphiques
- Lancer les régressions simples
- Lancer la régression multiple
- Arbre de décision

Pour traiter les données les boîtes de dialogue assurent la validation des flux en informant l'utilisateur du succès des opérations ou des erreurs de fichiers. Cette architecture garantit la cohérence de l'analyse en vérifiant que chaque étape critique est validée avant de passer à la suivante.

Variables de l'objet (Globales à la fenêtre) :

- fenetre: Objet Graphique
- df_logs_clean,df_notes_clean,df_final: Tableaux de données

DÉBUT ALGORITHME PRINCIPAL

1. **Initialisation de l'Interface :**
    - Créer une fenêtre nommée ApplicationAnalyse
    - Titre <----"Prédiction de Réussite - ARCHE"
    - Taille <---- 450x350 pixels
2. **Création du Menu (Boutons) :**
    - AFFICHER Titre "Analyse des logs de connexion"
    - CRÉER Bouton 1 : "Charger & Nettoyer" ----> Appelle etape_nettoyage
    - CRÉER Bouton 2 : "Calculer Indicateurs" ----> Appelle etape_calcul
    - CRÉER Bouton 3 : "Explorer Données" ----> Appelle etape_exploration
    - CRÉER Bouton 4 : "Afficher Graphiques" ----> Appelle etape_visu
    - CRÉER Bouton 5 : "Régression Simple" ----> Appelle etape_stats
    - CRÉER Bouton 6 : "Régression Multiple" ----> Appelle etape_stats_multiple
    - CRÉER Bouton 7 : "Arbre de Décision" ----> Appelle etape_arbre

3. **Lancement :**
    - Lancer la boucle d'écoute des événements (Attendre un clic de l'utilisateur)

FIN ALGORITHME PRINCIPAL

APPEL DES FONCTIONS :

Action : etape_nettoyage

- Appeler charger_donnees
- SI succès ALORS :
    - df_logs_clean, df_notes_clean <---- Appeler nettoyage_donnees
    - AFFICHER Message "Succès : Données nettoyées !"
- SINON :
    - AFFICHER Erreur "Fichiers introuvables"
- FIN SI

Action : etape_calcul

- SI df_logs_clean existe ALORS :
    - df_final <---- Appeler calculer_indicateurs
    - AFFICHER Message "Indicateurs calculés pour X étudiants"
- SINON :
    - AFFICHER Avertissement "Chargez d'abord les données"
- FIN SI

Action : etape_visu / etape_stats / etape_arbre

- SI df_final existe ALORS :
    - Appeler la fonction correspondante (Graphiques, Régression ou Arbre)
- SINON :
    - AFFICHER Avertissement "Calculs non effectués"
- FIN SI

Fichier interface.py :

```
import tkinter as tk
from tkinter import messagebox
from load import charger_donnees
from cleaner import nettoyage_donnees
from features import calculer_indicateurs
from regression import analyser_regressions
from regression_multiple import analyser_regression_multiple
from view import afficher_graphiques
from arbre_de_decision import entrainer_arbre_decision
from exploration import exploration_data, voir_les_absents

class ApplicationAnalyse(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Prédiction de Réussite - ARCHE")
        self.geometry("450x350")

        # Titre dans la fenêtre
        tk.Label(self, text="Analyse des logs de connexion", font=("Arial", 14, "bold")).pack(pady=10)

        # Bouton 1 : Charger et Nettoyer les données
        tk.Button(self, text="1. Charger & nettoyer les données", command=self.etape_nettoyage, width=35,
bg="#FFF3E0").pack(pady=5)

        # Bouton 2 : Calculer les indicateurs
        tk.Button(self, text="2. Calculer les indicateurs", command=self.etape_calcul, width=35,
bg="#F3E5F5").pack(pady=5)

        #  Bouton 3 : Exploration des données
        tk.Button(self, text="3. Explorer les données ", command=self.etape_exploration,
width=35,bg="#E0F2F1").pack(pady=5)

        # Bouton 4 : Voir les Graphiques
        tk.Button(self, text="4. Afficher les graphiques", command=self.etape_visu, width=35,
bg="#D1E8FF").pack(pady=5)

        # Bouton 5 : Régression simple
        tk.Button(self, text="5. Lancer les régressions simples", command=self.etape_stats, width=35,
bg="#D1FFD1").pack(pady=5)

        # Bouton 6 : Régression Multiple
        tk.Button(self, text="6. Lancer la régression multiple", command=self.etape_stats_multiple, width=35,
bg="#FFFFD1").pack(pady=5)

        #  BOUTON 7 : Arbre de décision
        tk.Button(self, text="7. Arbre de décision ", command=self.etape_arbre, width=35, bg="#D1C4E9").pack(pady=5)

        # Variables pour stocker les données entre les étapes
        self.df_final = None

    def etape_nettoyage(self):
        # On récupère les données brutes (df_logs, df_notes)
        df_logs, df_notes = charger_donnees()

        if df_logs is not None:
            # On stocke les données nettoyées dans l'application (self.df_logs_clean, self.df_notes_clean)
            self.df_logs_clean, self.df_notes_clean = nettoyage_donnees(df_logs, df_notes)
            messagebox.showinfo("Succès", "Données chargées et nettoyées !")
```

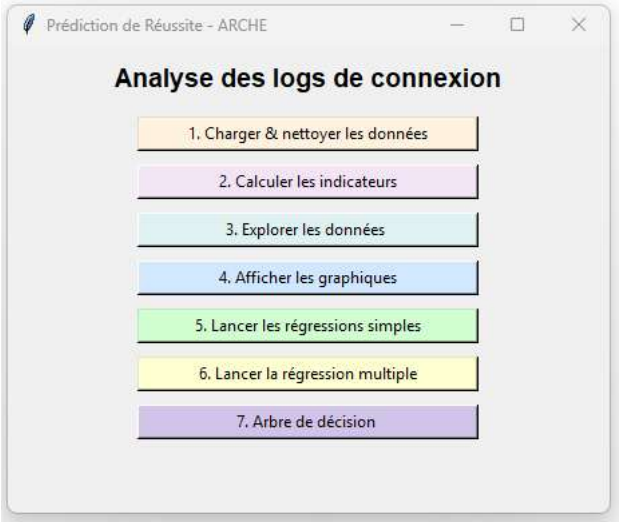
```
            else:
                messagebox.showerror("Erreur", "Fichiers introuvables.")

        def etape_exploration(self):
            # Vérification que les données nettoyées existent
            if hasattr(self, 'df_logs_clean'):
                print("\n--- EXPLORATION DES DONNÉES ---")
                exploration_data(self.df_logs_clean, self.df_notes_clean)
                voir_les_absents(self.df_logs_clean, self.df_notes_clean)
                messagebox.showinfo("Exploration", "L'exploration a été affichée dans la console.")
            else:
                messagebox.showwarning("Attention", "Veuillez d'abord charger et nettoyer les données.")

        def etape_calcul(self):
            # Utilisation des noms complets pour le calcul des indicateurs
            if hasattr(self, 'df_logs_clean'):
                self.df_final = calculer_indicateurs(self.df_logs_clean, self.df_notes_clean)
                messagebox.showinfo("Succès", f"Indicateurs calculés pour {len(self.df_final)} étudiants.")
            else:
                messagebox.showwarning("Attention", "Veuillez d'abord charger les données.")

        def etape_visu(self):
            if self.df_final is not None:
                afficher_graphiques(self.df_final)
            else:
                messagebox.showwarning("Attention", "Calculs non effectués.")

        def etape_stats(self):
            if self.df_final is not None:
                analyser_regressions(self.df_final)
                messagebox.showinfo("Analyse", "Résultats régressions simples générées dans la console.")
            else:
                messagebox.showwarning("Attention", "Données non prêtes.")

        def etape_stats_multiple(self):
            if self.df_final is not None:
                analyser_regression_multiple(self.df_final)
                messagebox.showinfo("Analyse Multiple", "Résultat régression multiple généré dans la console.")
            else:
                messagebox.showwarning("Attention", "Données non prêtes.")

        def etape_arbre(self):
            if self.df_final is not None:
                entrainer_arbre_decision(self.df_final)
                messagebox.showinfo("Arbre", "L'Arbre de Décision a été généré avec succès !")
            else:
                messagebox.showwarning("Attention", "Données non prêtes.")

    if __name__ == "__main__":
        app = ApplicationAnalyse()
        app.mainloop()
```

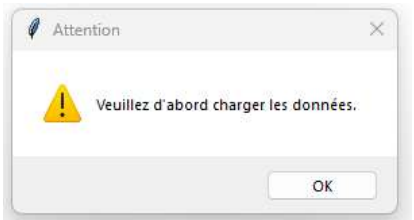
Gestion des erreurs de interface.py:

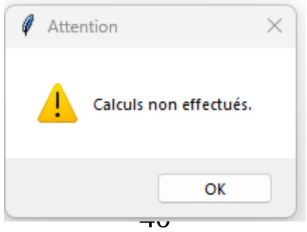
Contrairement aux scripts de console qui s'arrêtent, l'interface reste ouverte mais bloque les actions incorrectes :
Chaque bouton est protégé par une "garde" qui vérifie l'existence des données dans l'objet self avant de solliciter les modules externes.

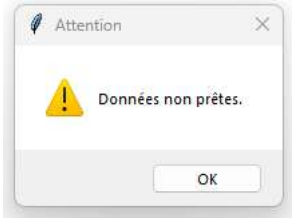
L'arrêt propre est ici remplacé par une interdiction d'agir.

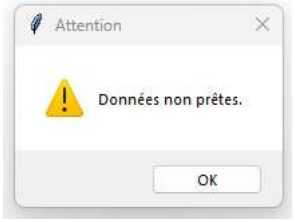
Test absence du fichier notes.csv








Attention
Données non prêtes.
OK


Attention
Données non prêtes.
OK


Attention
Données non prêtes.
OK

13. Interface (main.py) :

Le module main.py sert de point d'entrée unique pour l'exécution de l'ensemble du projet. Il importe la classe ApplicationAnalyse depuis le module d'interface pour instancier l'application graphique. Sa fonction principale consiste à lancer la boucle mainloop qui maintient la fenêtre de prédiction active pour l'utilisateur. Le script intègre des messages de suivi en console pour confirmer le démarrage et la fermeture propre de l'application. Cette structure permet d'isoler le lancement du logiciel de sa logique interne de calcul.

DÉBUT

1. **Importation des ressources :**
    - Charger la définition de la structure `ApplicationAnalyse` (l'interface graphique).
2. **Initialisation du projet :**
    - Créer une instance nommée `mon_interface` à partir du modèle `ApplicationAnalyse`.
    - *Note : À ce stade, la fenêtre est créée en mémoire avec tous ses boutons.*
3. **Lancement (Phase active) :**
    - AFFICHER "=== L'INTERFACE DE PRÉDICTION EST PRÊTE ==="
    - Appeler la méthodemainloop() de mon_interface()
    - *Note : L'algorithme se "met en pause" ici et attend les clics de l'utilisateur sur les boutons (Nettoyer, Calculer, etc.).*
4. **Clôture (Phase de sortie) :**
    - *Cette étape ne s'exécute que lorsque l'utilisateur ferme la fenêtre.*
    - AFFICHER "=== FERMETURE DE L'APPLICATION ==="

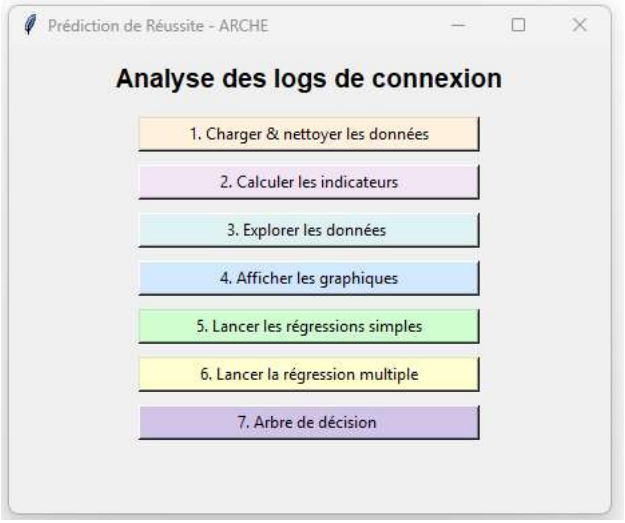
FIN

Fichier main.py :

```
# Importation de la classe ApplicationAnalyse depuis ton fichier interface.py
from interface import ApplicationAnalyse

def main():
    # Création de l'objet mon_interface
    mon_interface = ApplicationAnalyse()

    # Lancement de la boucle de l'application
    print("=== L'INTERFACE DE PRÉDICTION EST PRÊTE ===")
    mon_interface.mainloop()

    # Message affiché une fois que l'utilisateur ferme la fenêtre
    print("=== FERMETURE DE L'APPLICATION ===")

if __name__ == "__main__":
    # Appel de la fonction principale pour démarrer le projet
    main()
```

Gestion des erreurs de main.py:

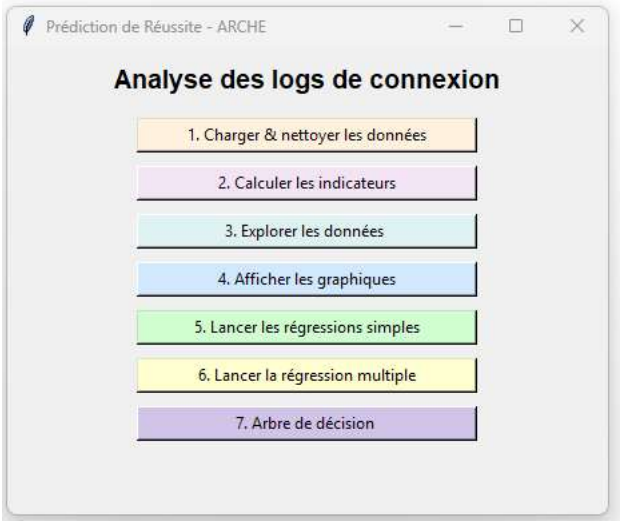
Le programme main.py instancie la classe ApplicationAnalyse et lancer la boucle mainloop() . À cette étape aucune donnée n'est encore lue sur le disque dur.
Le programme se contente de dessiner la fenêtre et d'attendre que l'utilisateur clique sur un bouton.



Et d’afficher le message :
=== L'INTERFACE DE PRÉDICTION EST PRÊTE ===

14. Conclusion :

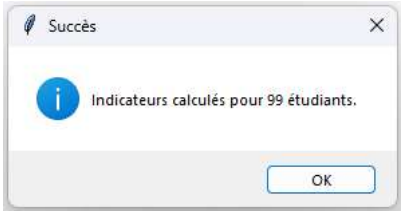
Lancement de l'application via le main :




```
=== L'INTERFACE DE PRÉDICTION EST PRÊTE ===
les données ont été chargées avec succès!
Fichier logs :16227 lignes lues
Fichier notes :99 lignes lues
le nombre de doublon dans notes.csv est de 0
il n'y a donc pas de doublon
Toutes les dates ont été converties correctement
Comparaison avant nettoyage
Étudiants dans le fichier NOTES : 99
Étudiants dans le fichier LOGS  : 109
Comparaison après nettoyage
Étudiants dans le fichier NOTES : 99
Étudiants dans le fichier LOGS  : 95
```

On constate que dans le fichier des notes nous avons 99 étudiants, dans le fichier des logs nous avons plus d'étudiant 109 cet écart est du à 10 personnes en dehors de la promotion qui se sont connectées.

Une fois le nettoyage de ces 10 personnes en dehors de la promotion réalisé, on constate qu'il y a un écart de 4 étudiant entre le fichier notes et le fichier logs, ceci est du au fait que ces 4 étudiants ne se sont pas connéctés sur ARCHE.



Apercu des 5 premières ligne du tableau :

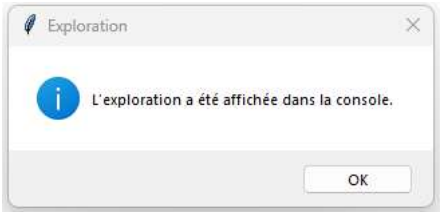
```
Aperçu des 5 premières lignes :
   pseudo    note  nb_clics  ...  nb_contextes  temps_total_sec  nb_tests
0     318  11.050      35.0  ...          12.0           1539.0      17.0
1     717  11.506     383.0  ...          73.0           9008.0     167.0
2     364  10.022     504.0  ...          62.0          11821.0     294.0
3      93  12.160     167.0  ...          25.0           2963.0     102.0
4     543   9.886     205.0  ...          44.0           3013.0     101.0
```

Ainsi que des étudiants qui n’ont pas ouvert ARCHE :

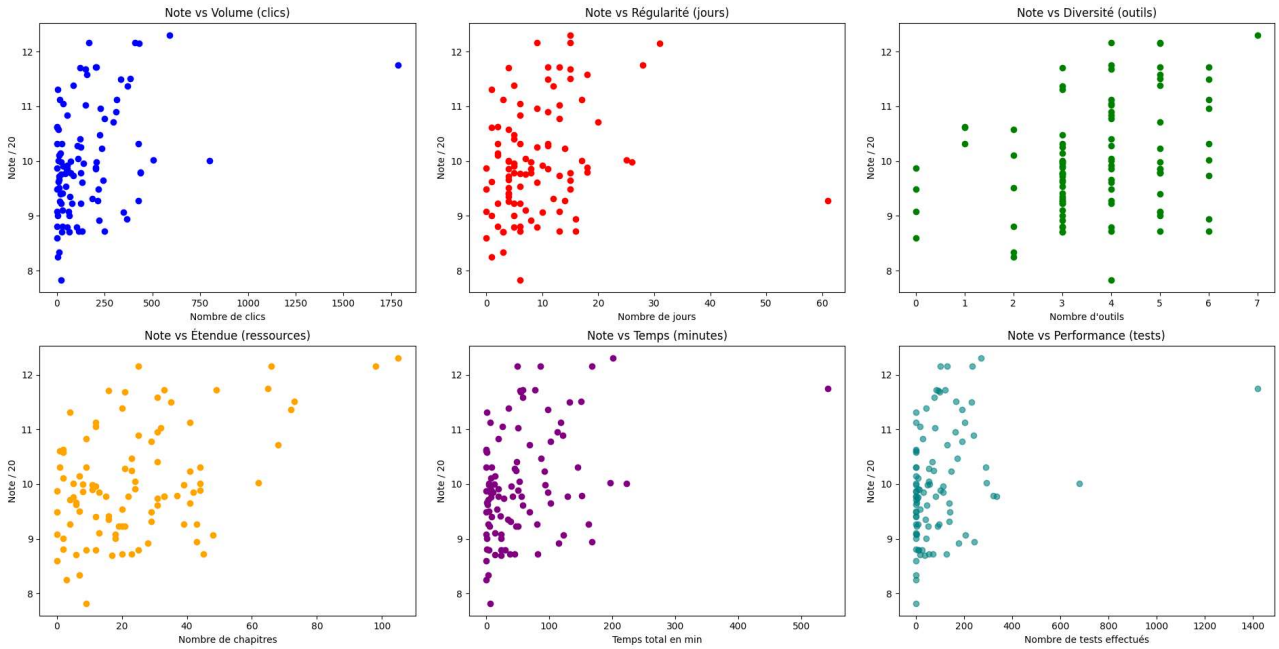
Un facteur de zéro est attribué pour les paramètres de suivi

On remarque que les 4 étudiants qui n’ont pas ouvert ARCHE ont tous des notes inférieures à 10

```
Étudiants qui n'ont jamais ouvert ARCHE (Vérification injection) :
    pseudo   note  nb_clics  ...  nb_contextes  temps_total_sec  nb_tests
14     312  9.078       0.0  ...           0.0              0.0       0.0
30     127  8.600       0.0  ...           0.0              0.0       0.0
47     990  9.873       0.0  ...           0.0              0.0       0.0
52     790  9.487       0.0  ...           0.0              0.0       0.0
```




```
--- EXPLORATION DES DONNÉES ---
Nombre d'étudiants actifs dans les logs : 95
Nombre de ressources (cours/chapitres) consultées : 106
Top 5 des types d'actions :
evenement
Module de cours consulté               4295
Cours consulté                         2924
Tentative de test relue                1433
Tentative de test consultée            1296
Résumé de tentative de test consulté   1169
Visaulisation des étudiant sans activité sur ARCHE
Voici les 4 étudiants qui n'ont jamais ouvert ARCHE :
    pseudo   note
14     312  9.078
30     127  8.600
47     990  9.873
52     790  9.487
```



Visuellement on peut déduire que plus on se déplace vers la droite (plus on s'investit), plus les points ont tendance à monter (meilleure note).

<u>Régression simples :</u>

--- ANALYSE PAR RÉGRESSION LINÉAIRE ---

Indicateur : nb_clics

Coefficient de corrélation (r) : 0.36

Coefficient de détermination (r^2) : 0.13

Équation : Note = 0.00 * nb_clics + 9.74

Indicateur : nb_jours

Coefficient de corrélation (r) : 0.24

Coefficient de détermination (r^2) : 0.06

Équation : Note = 0.03 * nb_jours + 9.74

Indicateur : nb_composants

Coefficient de corrélation (r) : 0.32

Coefficient de détermination (r^2) : 0.10

Équation : Note = 0.22 * nb_composants + 9.19

Indicateur : nb_contextes

Coefficient de corrélation (r) : 0.46

Coefficient de détermination (r^2) : 0.21

Équation : Note = 0.02 * nb_contextes + 9.47

Indicateur : temps_total_sec

Coefficient de corrélation (r) : 0.36

Coefficient de détermination (r^2) : 0.13

Équation : Note = 0.00 * temps_total_sec + 9.72

Indicateur : nb_tests

Coefficient de corrélation (r) : 0.32

Coefficient de détermination (r^2) : 0.10

Équation : Note = 0.00 * nb_tests + 9.83

Chaque facteur séparément ne permet pas d'expliquer la note obtenue, la régression de chaque paramètre unitairement donne de faible fiabilité (r^2) la plus importante atteint 0,21. C'est pour cette raison que je vais faire une régression multiple.

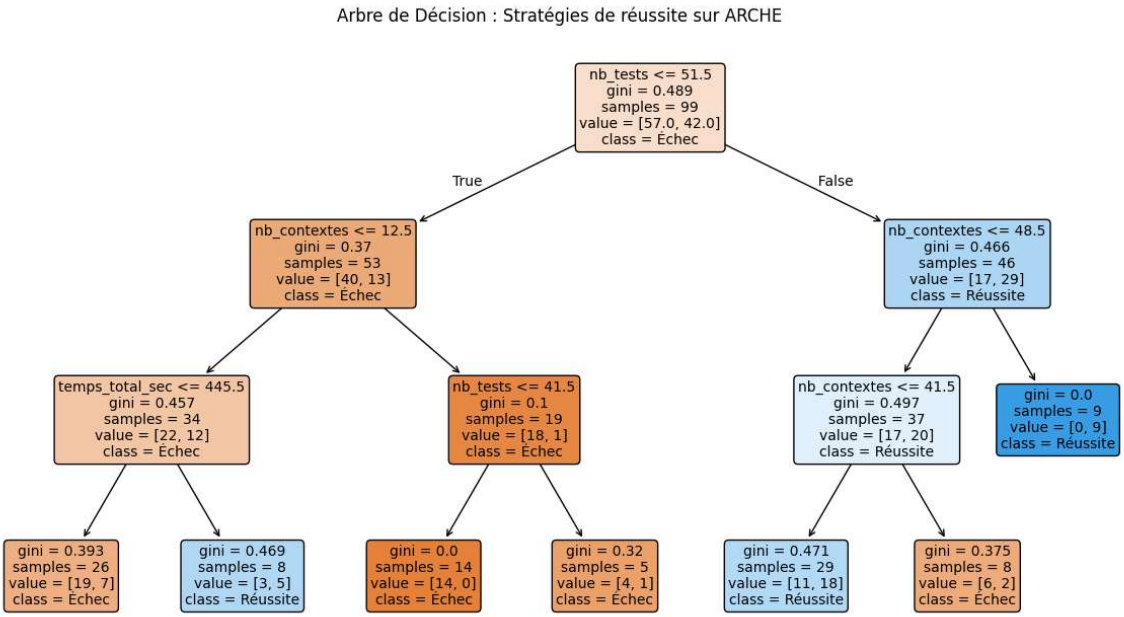
Régresion multiple :

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   note   R-squared:                       0.230
Model:                            OLS   Adj. R-squared:                  0.179
Method:                 Least Squares   F-statistic:                     4.570
Date:                Sun, 15 Feb 2026   Prob (F-statistic):           0.000421
Time:                        12:24:12   Log-Likelihood:                -127.05
No. Observations:                  99   AIC:                             268.1
Df Residuals:                      92   BIC:                             286.3
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept          9.4004      0.264     35.657      0.000       8.877       9.924
nb_clics           0.0012      0.005      0.257      0.797      -0.008       0.010
nb_jours          -0.0180      0.025     -0.723      0.472      -0.068       0.032
nb_composants      0.0524      0.088      0.599      0.551      -0.122       0.226
nb_contextes       0.0209      0.011      1.975      0.051      -0.000       0.042
temps_total_sec -4.582e-05      0.000     -0.437      0.663      -0.000       0.000
nb_tests           0.0003      0.004      0.069      0.945      -0.008       0.009
==============================================================================
Omnibus:                        2.477   Durbin-Watson:                   1.869
Prob(Omnibus):                  0.290   Jarque-Bera (JB):                2.111
Skew:                           0.238   Prob(JB):                        0.348
Kurtosis:                       2.466   Cond. No.                     1.65e+04
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.65e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
FORMULES UTILISÉES
1. Modèle : Y = Xβ + ε (où Y est la note et X les indicateurs)
2. R-squared (R²) : 0.23
 Cela signifie que 23% de la note est expliquée par l'activité ARCHE.
3. Prob (F-statistic) : 4.2111e-04
Comme la p-value (4.2111e-04) est inférieure à 0.05,
 le modèle global est statistiquement significatif.
```

La régression multiple de l’ensemble des paramètres donne également un coefficient de détermination (r^2) de 0,23 qui est relativement faible.

Arbre de décision :



L’arbre de décision permet de déterminer les meilleurs conditions qui permettent d’obtenir le diplôme ces conditions sont les suivantes :

- Faire un nombre de test supérieur à 52
- Voir un nombre de contexte supérieur à 49

Ce devoir m’aura permis de mettre en place une application et de pouvoir sélectionner le meilleur modèle qui dans ce cas précis et l’arbre de décision.