Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique

Manipulation

Filière :**2ITE**
Niveau : **3ème Année**
Année universitaire : **2020/2021**

Subject :

MLOPS :Automatisation des pipelines ML avec Airflow

Réalisé par :
Berthe Mariam Tiotio
DKAKI ABDERRAHIM

Encadré par :
Prof .Kalloubi Fahd

I. DataSet

- Comme Dataset nous avons utilisé **Adult Salary Dataset** ,ses données ont été collecté en 1994 . Le revenu annuel d'un individu résulte de divers facteurs, il est influencé par le niveau d'éducation de l'individu, son âge, son sexe, sa profession, son pays natal … etc.

II. Installation et Configuration

- Installation du airflow dans WSL[Ubuntu 20.4]
  - export AIRFLOW_HOME=~/airflow
  - sudo pip install apache-airflow
  - airflow db init

- Ensuite Création du l'utilisateur admin
  airflow users create -r Admin -u admin -e email@email.com -f admin -l user -p admin

- Bibliothéques utilisées:
  -pip3 install pandas
  -pip install numpy
  -pip3 install scikit-learn
  -pip3 install matplotlib
  -pip3 install seaborn
  -pip3 install imblearn
  -pip3 install pickle

- Dans la ligne de commande Ubuntu tapez explorer.exe . pour ouvrir les fichiers de WSL .

- Ouvrez le fichier airflow.cf pour savoir l'emplacement ou Apache Airflow lit les DAG (normalent c'est /home/{nom_du_host}/airflow/dags)

- Créer le fichier dags s'il n'existe pas

- Coller le contenu du fichiers dags[ressources] dans le fichiers crée :
- On démarre apache airflow
  airflow webserver –p 8080

  airflow scheduler

- L'authentification :login=admin , password=admin

III. Création du DAG

❖ Initialisation du DAG :

```
default_args = {
    'owner': 'Berthe-Dkaki',
    'depends_on_past': False,
    'email': ['e-dkaki.a@ucd.ma'],
    'email_on_failure': False,
    'email_on_retry': False,
    'email_on_success': False,
    #'retries': 1,
    #'retry_delay': timedelta(minutes=5),
    # 'end_date': datetime(2020, 1, 30),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,

}
dag = DAG(
    dag_id = 'MLOPS',
    start_date = datetime(2020,1,1),
    default_args = default_args,
    description='MLOPS :automatisation des pipelines machines learnings à l aide de airflo
w '

    )
```

❖ Différents taches à excécuter :

```
get_data1 = PythonOperator(
    task_id = 'get_train_data1',
    python_callable = get_train_data1,
    xcom_push=True,
    provide_context=True,
    dag = dag)

get_data2 = PythonOperator(
    task_id = 'get_train_data2',
    python_callable = get_train_data2,
    xcom_push=True,
    provide_context=True,
    dag = dag)

get_data3 = PythonOperator(
    task_id = 'get_train_data3',
    python_callable = get_train_data3,
    xcom_push=True,
    provide_context=True,
    dag = dag)

data_merging = PythonOperator(
```

```
    task_id = 'data_merging',
    python_callable = merging,
    xcom_push=True,
    provide_context=True,
    dag = dag)

data_preprocessing = PythonOperator(
    task_id = 'data_preprocessing',
    python_callable = preprocessing,
    provide_context=True,
    dag = dag)

data_visualisation = PythonOperator(
    task_id = 'data_visualisation',
    python_callable = visualisation,
    provide_context=True,
    dag = dag)

feauture_engineering = PythonOperator(
    task_id = 'data_feautures',
    python_callable = selection,
    provide_context=True,
    dag = dag)

LR_modeling = PythonOperator(
    task_id = 'modele_LR',
    python_callable = modeling_LR,
    provide_context=True,
    dag = dag)

RF_modeling = PythonOperator(
    task_id = 'modele_RF',
    python_callable = modeling_LR,
    provide_context=True,
    dag = dag)

SVM_modeling = PythonOperator(
    task_id = 'modele_SVM',
    python_callable = modeling_SVM,
    provide_context=True,
    dag = dag)

NB_modeling = PythonOperator(
    task_id = 'modele_NB',
    python_callable = modeling_NB,
    provide_context=True,
    dag = dag)

saving_modele = PythonOperator(
    task_id = 'save_model',
    python_callable = save_model,
    provide_context=True,
```

```
    dag = dag)
```

❖ Priorité des taches :

```
#data_merging.set_upstream(get_data1)
#data_merging.set_upstream(get_data2)
#data_merging.set_upstream(get_data3)
#data_preprocessing.set_upstream(data_merging)
#data_visualisation.set_upstream(data_preprocessing)
#feauture_engineering.set_upstream(data_visualisation)

get_data1 >> data_merging
get_data2 >> data_merging
get_data3 >> data_merging
data_merging >> data_preprocessing
data_preprocessing >> data_visualisation
data_visualisation >> feauture_engineering
feauture_engineering >> LR_modeling
feauture_engineering >> RF_modeling
feauture_engineering >> SVM_modeling
feauture_engineering >> NB_modeling
LR_modeling >> saving_modele
RF_modeling >> saving_modele
SVM_modeling >> saving_modele
NB_modeling >> saving_modele
```

❖ Extraction des données :

```
import pandas as pd
def get_train_data1(**kwargs):
    dataset1= pd.read_csv('/home/mariam/airflow/dags/datasets/adult1.csv')

    kwargs['ti'].xcom_push(key='dataset1', value=dataset1)
    dataset1.head()
    dataset1.info()
def get_train_data2(**kwargs):
    dataset2= pd.read_csv('/home/ mariam /airflow/dags/datasets/adult2.csv')

    kwargs['ti'].xcom_push(key='dataset2', value=dataset2)
    dataset2.head()
    dataset2.info()

def get_train_data3(**kwargs):
    dataset3= pd.read_csv('/home/ mariam /airflow/dags/datasets/adult3.csv')

    kwargs['ti'].xcom_push(key='dataset3', value=dataset3)
    dataset3.head()
    dataset3.info()
```

❖ Fusion des données :

```
import sys
import pandas as pd
```

```
def merging(**kwargs):

    df1 =kwargs['ti'].xcom_pull(task_ids='get_train_data1',key='dataset1')
    df2 =kwargs['ti'].xcom_pull(task_ids='get_train_data2',key='dataset2')
    df3 =kwargs['ti'].xcom_pull(task_ids='get_train_data3',key='dataset3')

    df = df1.merge(df2,on='id').merge(df3,on='id')
    print(df.head(30))

    kwargs['ti'].xcom_push(key='df', value=df)
```

❖ Nettoyage des données :

```
from merge import merging
import pandas
import numpy

def preprocessing(**kwargs):
    df =kwargs['ti'].xcom_pull(task_ids='data_merging',key='df')

    print(df.info())
    print(df.describe().T)
    print(round((df.isnull().sum() / df.shape[0]) * 100, 2).astype(str) + ' %')
    print(round((df.isin(['?']).sum() / df.shape[0])
      * 100, 2).astype(str) + ' %')

    col_names = df.columns

    for c in col_names:
      df = df.replace("?", numpy.NaN)
    df = df.apply(lambda x:x.fillna(x.value_counts().index[0]))


    kwargs['ti'].xcom_push(key='df', value=df)
```

❖ Graphes :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


def visualisation(**kwargs):
    df =kwargs['ti'].xcom_pull(task_ids='data_preprocessing',key='df')

    income = df['income'].value_counts()

    plt.style.use('seaborn-whitegrid')
    plt.figure(figsize=(7, 5))
    sns.barplot(income.index, income.values, palette='bright')
    plt.title('Distribution of Income', fontdict={
```

```
        'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
    plt.xlabel('Income', fontdict={'fontname': 'Monospace', 'fontsize': 15})
    plt.ylabel('Number of people', fontdict={
        'fontname': 'Monospace', 'fontsize': 15})
    plt.tick_params(labelsize=10)

    plt.savefig("/home/ mariam /airflow/dags/chart/distribution-
income.pdf", bbox_inches='tight')

    age = df['age'].value_counts()

    plt.figure(figsize=(10, 5))
    plt.style.use('fivethirtyeight')
    sns.distplot(df['age'], bins=20)
    plt.title('Distribution of Age', fontdict={
        'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
    plt.xlabel('Age', fontdict={'fontname': 'Monospace', 'fontsize': 15})
    plt.ylabel('Number of people', fontdict={
        'fontname': 'Monospace', 'fontsize': 15})
    plt.tick_params(labelsize=10)
    plt.savefig("/home/ mariam /airflow/dags/chart/Distribution-
Age.pdf", bbox_inches='tight')
```

- Feautures-selection :

```
ef selection(**kwargs):
    df =kwargs['ti'].xcom_pull(task_ids='data_preprocessing',key='df')

    for col in df.columns: #Label Encoding
        if df[col].dtypes == 'object':
           encoder = LabelEncoder()
           df[col] = encoder.fit_transform(df[col])

    X = df.drop('income', axis = 1)
    Y = df['income']

    selector = ExtraTreesClassifier(random_state = 42)
    selector.fit(X, Y)
    feature_imp = selector.feature_importances_
    for index, val in enumerate(feature_imp):
        print(index, round((val * 100), 2))

    X = X.drop(['workclass', 'education', 'race', 'gender', 'capital-loss', 'native-
country'], axis = 1)

    for col in X.columns:
      scaler = StandardScaler()
      X[col] = scaler.fit_transform(X[col].values.reshape(-1, 1))
    round(Y.value_counts(normalize=True) * 100, 2).astype('str') + ' %'

    ros = RandomOverSampler(random_state=42)
```

```
    ros.fit(X, Y)
    X_resampled, Y_resampled = ros.fit_resample(X, Y)
    print(round(Y_resampled.value_counts(normalize=True) * 100, 2).astype('str') + ' %')

    X_train, X_test, Y_train, Y_test = train_test_split(X_resampled, Y_resampled, test_siz
e = 0.2, random_state = 42)

    print("X_train shape:", X_train.shape)
    print("X_test shape:", X_test.shape)
    print("Y_train shape:", Y_train.shape)
    print("Y_test shape:", Y_test.shape)

    kwargs['ti'].xcom_push(key='X_train',value=X_train)
    kwargs['ti'].xcom_push(key='X_test',value=X_test)
    kwargs['ti'].xcom_push(key='Y_train',value=Y_train)
    kwargs['ti'].xcom_push(key='Y_test',value=Y_test)
```

- Model du regression logistique :

```
ef modeling_LR(**kwargs):


    X_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_train')
    X_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_test')
    Y_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_train')
    Y_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_test')

    log_reg = LogisticRegression(random_state=42)
    log_reg.fit(X_train, Y_train)

    Y_pred_log_reg = log_reg.predict(X_test)

    print('Logistic Regression:')
    print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_log_reg) * 100, 2))
    print('F1 score:', round(f1_score(Y_test, Y_pred_log_reg) * 100, 2))
```

- Model du SVM :

```
def modeling_SVM(**kwargs):


    X_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_train')
    X_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_test')
    Y_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_train')
    Y_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_test')

    svc = SVC(random_state=42)
```

```
    svc.fit(X_train, Y_train)
    Y_pred_svc = svc.predict(X_test)
    print('Support Vector Classifier:')
    print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_svc) * 100, 2))
    print('F1 score:', round(f1_score(Y_test, Y_pred_svc) * 100, 2))
```

- ❖ Model RandomForest ：

```
def modeling_RF(**kwargs):


    X_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_train')
    X_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_test')
    Y_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_train')
    Y_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_test')

    ran_for = RandomForestClassifier(random_state=42)

    ran_for.fit(X_train, Y_train)
    Y_pred_ran_for = ran_for.predict(X_test)
    print('Random Forest Classifier:')
    print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_ran_for) * 100, 2))
    print('F1 score:', round(f1_score(Y_test, Y_pred_ran_for) * 100, 2))

    kwargs['ti'].xcom_push(key='ran_for', value=ran_for)
```

- ❖ Model SVM:

```
def modeling_SVM(**kwargs):


    X_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_train')
    X_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_test')
    Y_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_train')
    Y_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_test')

    svc = SVC(random_state=42)
    svc.fit(X_train, Y_train)
    Y_pred_svc = svc.predict(X_test)
    print('Support Vector Classifier:')
    print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_svc) * 100, 2))
    print('F1 score:', round(f1_score(Y_test, Y_pred_svc) * 100, 2))
```

- ❖ Model NB ：

```
def modeling_NB(**kwargs):


    X_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_train')
    X_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='X_test')
    Y_train =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_train')
    Y_test =kwargs['ti'].xcom_pull(task_ids='data_feautures',key='Y_test')
```

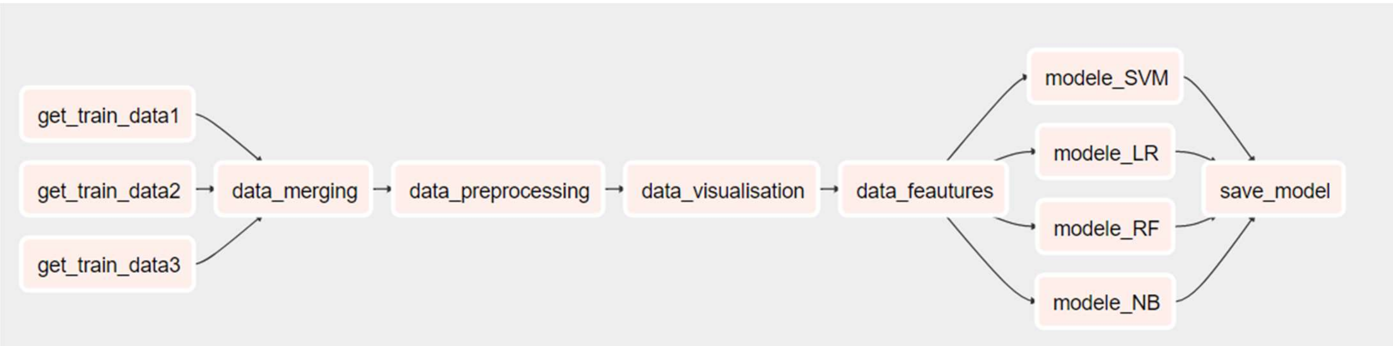
```python
    nb = GaussianNB()
    nb.fit(X_train, Y_train)
    Y_pred_nb = nb.predict(X_test)

    print('Naive Bayes Classifier:')
    print('Accuracy score:', round(accuracy_score(Y_test, Y_pred_nb) * 100, 2))
    print('F1 score:', round(f1_score(Y_test, Y_pred_nb) * 100, 2))
```

❖ Sauvegarde de meilleur model :

```python
def save_model(**kwargs):


    ran_for =kwargs['ti'].xcom_pull(task_ids='modele_RF',key='ran_for')



    Pkl_Filename = "Model.pkl"

    with open(Pkl_Filename, 'wb') as file:
      pickle.dump(ran_for, file)
```

IV- Exécution du DAG

❖ Finalement le shéma de notre DAG :



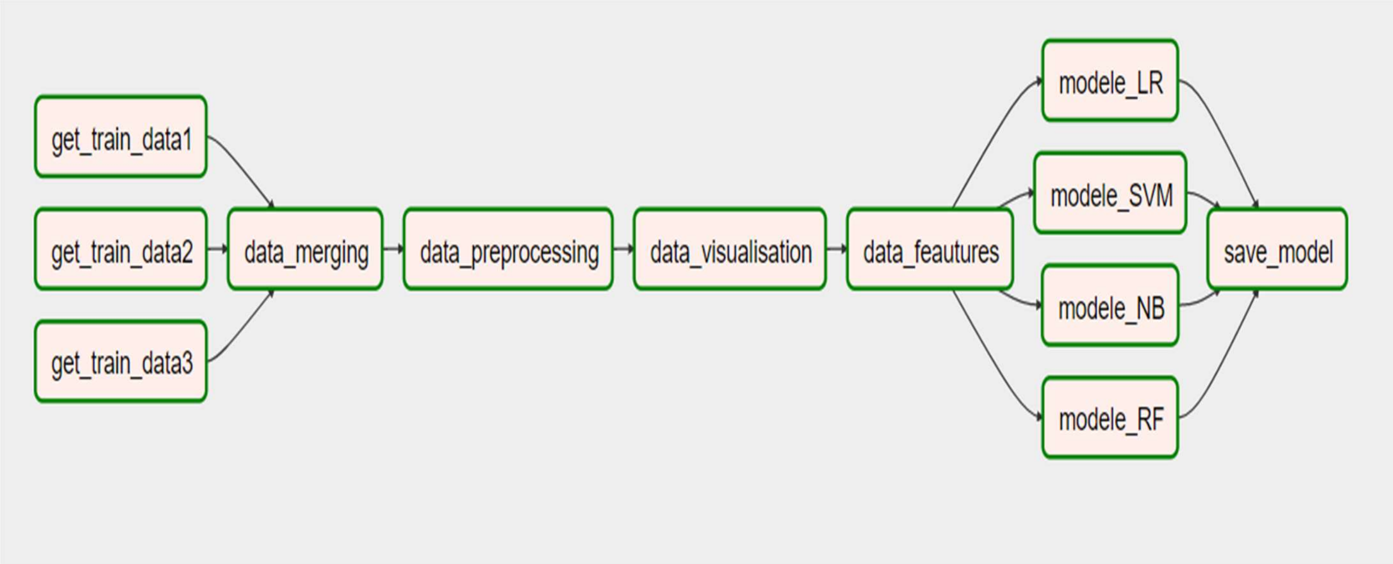
❖ Exécution peut se faire avec deux mèthodes :

Soit avec l'interface du airflow:

Ou avec la commande :

airflow run MLOPS(nom du Dag) **get_train_data1**(première taches) **2020-1-1**(date de début)

❖ Aprés l'excécution :

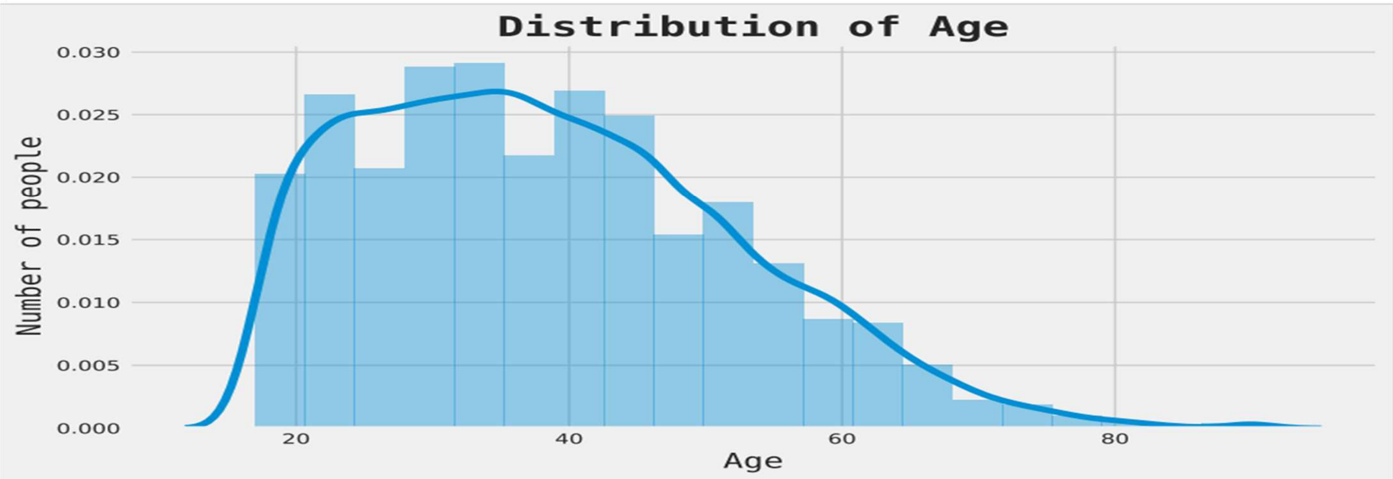


❖ Création du model :

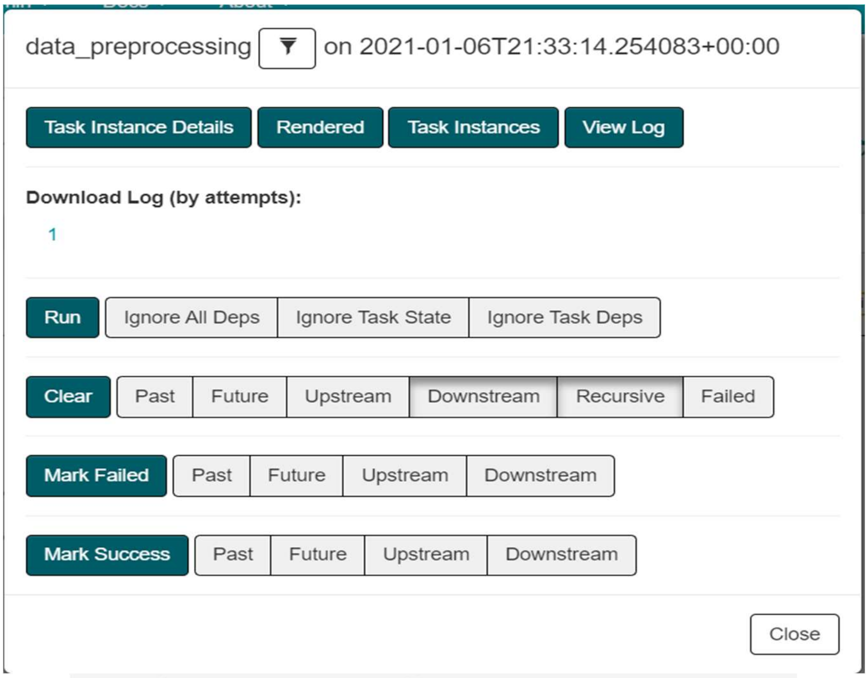
pycache	05/01/2021 14:52	Dossier de fichiers	
chart	04/01/2021 21:58	Dossier de fichiers	
datasets	02/01/2021 20:24	Dossier de fichiers	
python	04/01/2021 20:37	Dossier de fichiers	
income.py	05/01/2021 14:52	Python File	5 Ko
Model.pkl	06/01/2021 22:42	Fichier PKL	1 Ko

❖ Création des graphes statistiques:

Nom	Modifié le	Type	Taille
Distribution-Age.pdf	06/01/2021 22:38	Microsoft Edge PD...	20 Ko
Distribution-Education.pdf	06/01/2021 22:38	Microsoft Edge PD...	27 Ko
distribution-income.pdf	06/01/2021 22:38	Microsoft Edge PD...	17 Ko
Hours-work-per-week.pdf	06/01/2021 22:38	Microsoft Edge PD...	19 Ko

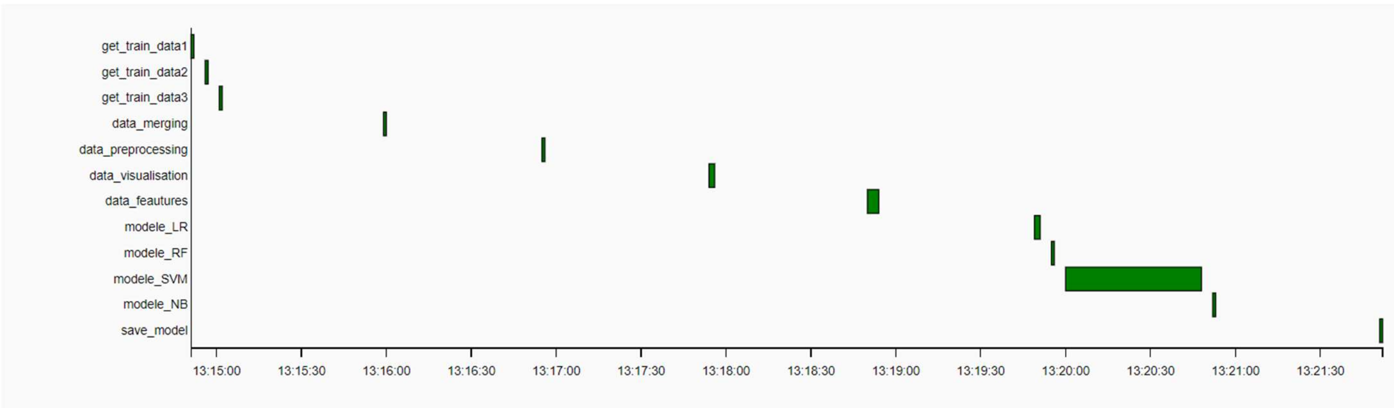


❖ On peut consulter le log de chaque taches : par exemple on clique sur la tache preprocessing et on clique sur View log

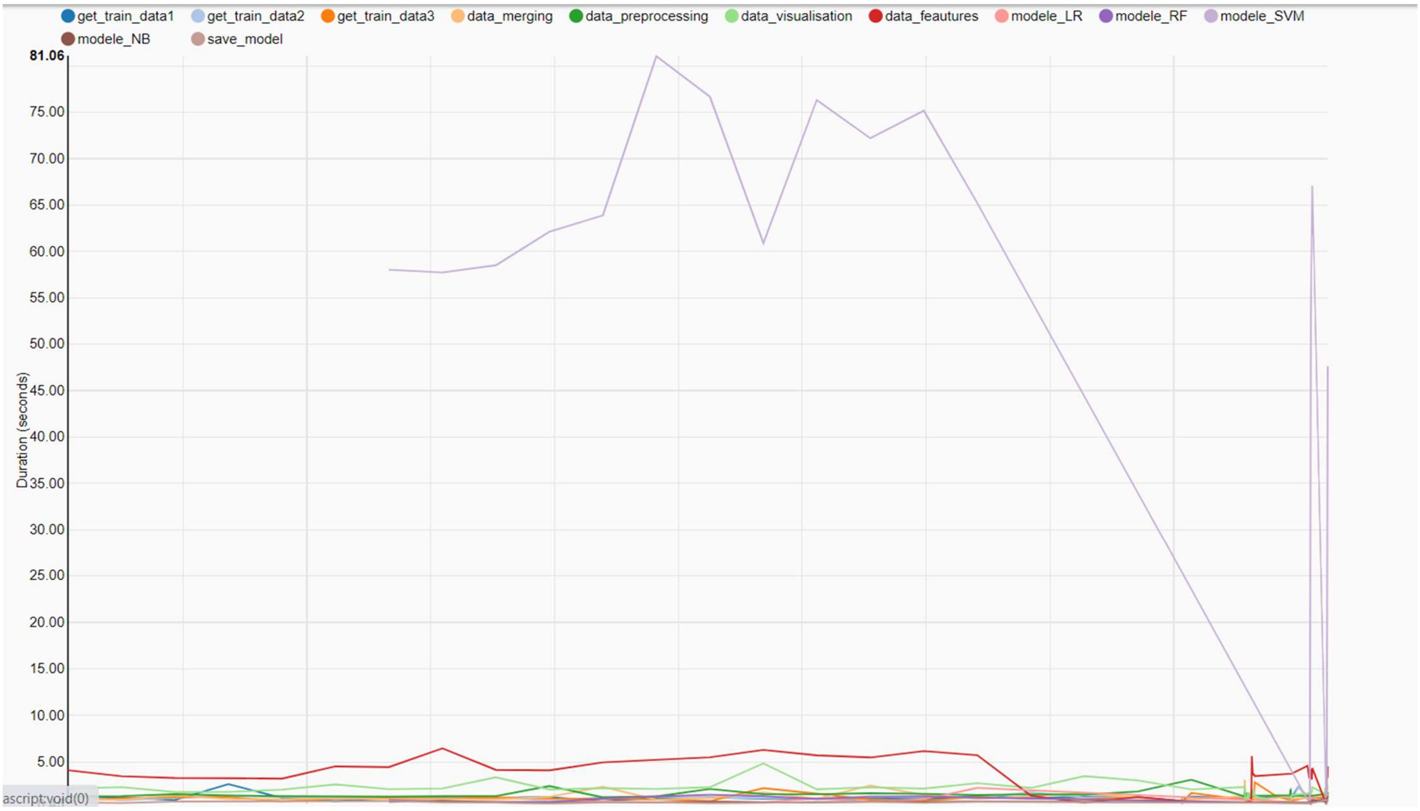



```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                32561 non-null  int64
 1   age               32561 non-null  int64
 2   workclass         32561 non-null  object
 3   fnlwgt            32561 non-null  int64
 4   education         32561 non-null  object
 5   educational-num   32561 non-null  int64
 6   marital-status    32561 non-null  object
 7   occupation        32561 non-null  object
 8   relationship      32561 non-null  object
 9   race              32561 non-null  object
 10  gender            32561 non-null  object
 11  capital-gain      32561 non-null  int64
 12  capital-loss      32561 non-null  int64
 13  hours-per-week    32561 non-null  int64
 14  native-country    32561 non-null  object
 15  income            32561 non-null  object
dtypes: int64(7), object(9)
memory usage: 4.2+ MB
```

❖ Diagramme de Gantt :



❖ Durée cumulative de chaque tache :



POUR L'exécution du DAGS remplacer mariam dans les fichiers python par votre nom du machine
/home/ mariam /airflow/…