📚 SESSÃO 1 - CONFIGURAÇÃO DO AMBIENTE

📖 AULA 1.1 WSL

🔗 Links:

Como instalar o Linux no Windows com o WSL:
https://learn.microsoft.com/pt-br/windows/wsl/install

🖱️ Comandos:

Powershell:

```
wsl --version
wsl --set-default-version 2
wsl --list --online
wsl --install -d Ubuntu-24.04
python3 --version
```

📖 AULA 1.2 Conda

🔗 Links:

Installing Miniconda: https://www.anaconda.com/docs/getting-started/miniconda/install

🖱️ Comandos:

WSL:

```
cd ~
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-
x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh
source ~/.bashrc
conda config --set auto_activate_base false
rm Miniconda3-latest-Linux-x86_64.sh
conda --version
```

📖 AULA 1.3 Git

🔗 Links:

Downloads: https://git-scm.com/downloads

🖱️ Comandos:

WSL:

```
sudo apt-get install git
git --version
```

📖 AULA 1.4 Docker

🔗 Links:

Downloads: https://docs.docker.com/engine/install

🖱️ Comandos:

WSL:

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")
stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin

sudo usermod -aG docker $USER
```

```
newgrp docker

docker --version
```

📖 AULA 1.5 VSCode

🔗 Links:

VSCode: https://code.visualstudio.com

📖 AULA 1.6 GitHub, DagsHub, DockerHub

🔗 Links:

GitHub: https://github.com/?locale=pt-br
DagsHub: https://dagshub.com
DockerHub: https://hub.docker.com

📚 SESSÃO 2 - PROJETO BASE

📖 AULA 2.1 Estrutura do projeto

🔗 Links:

Repositório no GitHub: https://github.com/jonesgranatyr/mlops_project

📖 AULA 2.2 Preparo local

🖱️ Comandos:

Terminal (WSL):

```
cd ~
mkdir PythonProjects
ls -la
rm -rf .git
```

```
pip install -e .
git init
git config --global user.name <your-nome>
git config --global user.email <your-email>
```

💻 Código:

.gitignore:

```
ml_classifier.egg-info
```

📖 AULA 2.3 Pacote src

Sem material auxiliar.

📖 AULA 2.4 Módulo load_data

🖱️ Comandos:

Debug console:

```
data.isna().sum()/len(data)
```

Terminal:

```
python -m src.data_loading.load_data
```

💻 Código:

.gitignore:

```
data/raw/raw.csv
```

📖 AULA 2.5 Módulo preprocess_data

🖱️ Comandos:

Debug console:

```
train_features.isna().sum()
train_features_processed.isna().sum()
```

Terminal:

```
python -m src.data_preprocessing.preprocess_data
```

💻 Código:

.gitignore:

```
data/preprocessed/train_preprocessed.csv
data/preprocessed/test_preprocessed.csv
artifacts/\[features\]_mean_imputer.joblib
```

📖 AULA 2.6 Módulo engineer_features

🖱️ Comandos:

Debug console:

```
train_preprocessed
train_processed
```

Terminal:

```
python -m src.feature_engineering.engineer_features
```

💻 Código:

.gitignore:

```
data/processed/train_processed.csv
data/processed/test_rocessed.csv
artifacts/\[features\]_scaler.joblib
```

📖 AULA 2.7 Módulo train_model

🖱️ Comandos:

Debug console:

```
params
```

Terminal:

```
python -m src.model_training.train_model
```

💻 Código:

.gitignore:

```
artifacts/\[target\]_one_hot_encoder.joblib
models/model.keras
metrics/training.json
```

📖 AULA 2.8 Módulo evaluate_model

🖱️ Comandos:

Debug console:

```
y_pred_proba
y_pred
report
cm
```

Terminal:

```
python -m src.model_evaluation.evaluate_model
```

💻 Código:

.gitignore:

```
metrics/evaluation.json
```

📖 AULA 2.9 Pacote app

💻 Código:

.gitignore:

```
# Python
ml_classifier.egg-info
__pycache__

# Data
data/**/*.csv

# Artifacts
artifacts/*.joblib

# Models
models/*.keras

# Metrics
metrics/*.json
```

📖 AULA 2.10 Docker

🖱️ Comandos:

Terminal:

```
docker build -t ml-classifier .
docker image list
docker run -p 5001:5001 ml-classifier
docker ps
docker container stop <container_id>
```

📖 AULA 2.11 Repositório remoto - GitHub

🖱️ Comandos:

Terminal:

```
git status
git add .
git commit -m "first commit"
git remote add origin <remote-url>
git branch -M main
git push -u origin main
git push
```

📚 SESSÃO 3 - DVC

📖 AULA 3.1 Arquivos de configuração

🖱️ Comandos:

Terminal:

```
pip install -e .
dvc init
```

💻 Código:

pyproject.toml:

```
version = "0.2"

<...>
```

```
"dvc==3.59.1",
```

.dockerignore:

```
# DVC
.dvc
```

dvc.yaml:

```
stages:
  load_data:
    cmd: python -m src.data_loading.load_data
    deps:
      - src/data_loading/load_data.py
    outs:
      - data/raw/raw.csv

  preprocess_data:
    cmd: python -m src.data_preprocessing.preprocess_data
    deps:
      - src/data_preprocessing/preprocess_data.py
      - data/raw/raw.csv
    outs:
      - data/preprocessed/train_preprocessed.csv
      - data/preprocessed/test_preprocessed.csv
      - artifacts/[features]_mean_imputer.joblib
    params:
      - preprocess_data.test_size
      - preprocess_data.random_seed

  engineer_features:
    cmd: python -m src.feature_engineering.engineer_features
    deps:
      - src/feature_engineering/engineer_features.py
      - data/preprocessed/train_preprocessed.csv
      - data/preprocessed/test_preprocessed.csv
    outs:
      - data/processed/train_processed.csv
      - data/processed/test_processed.csv
      - artifacts/[features]_scaler.joblib

  train:
    cmd: python -m src.model_training.train_model
    deps:
      - src/model_training/train_model.py
      - data/processed/train_processed.csv
    outs:
      - models/model.keras
      - artifacts/[target]_one_hot_encoder.joblib
    metrics:
      - metrics/training.json
    params:
      - train.learning_rate
      - train.hidden_layer_1_neurons
      - train.hidden_layer_2_neurons
      - train.dropout_rate
      - train.epochs
      - train.batch_size

  evaluate:
    cmd: python -m src.model_evaluation.evaluate_model
    deps:
      - src/model_evaluation/evaluate_model.py
      - models/model.keras
      - artifacts/[target]_one_hot_encoder.joblib
      - data/processed/test_processed.csv
    metrics:
      - metrics/evaluation.json
```

📖 AULA 3.2 Execução automática da pipeline

🖱️ Comandos:

Terminal:

```
dvc repro
```

```
git add dvc.lock
dvc config core.autostage true
dvc pull
```

📖 AULA 3.3 Versionamento de dados

🖱️ Comandos:

Terminal:

```
dvc repro
dvc repro load_data
git checkout <hash_id>
dvc checkout
git checkout main
```

💻 Código:

dvc.yaml:

```
always_changed: True
```

load_data:

```
import time

<...>

data = data.sample(frac=1, random_state=int(time.time())).reset_index(drop=True)
```

📖 AULA 3.4 Experimentos

🖱️ Comandos:

Terminal:

```
dvc exp run -S train.batch_size=16
dvc exp run -S preprocess_data.test_size=0.25 --queue
dvc exp run -S train.dropout_rate=0.25 --queue
dvc exp run -S train.random_seed=24 --queue
dvc exp run --run-all
dvc exp show
dvc exp apply <experiment_id>
```

📖 AULA 3.5 Repositório remoto - DagsHub

🖱️ Comandos:

Terminal:

```
dvc remote add origin <dagshub-url>
dvc remote modify origin --local auth basic
dvc remote modify origin --local user <username>
dvc remote modify origin --local password <token>
dvc remote list
dvc remote default origin
dvc push
dvc pull
```

📚 SESSÃO 4 - MLFLOW

📖 AULA 4.1 Experimento e run

🖱️ Comandos:

Terminal:

```
pip install -e .
dvc repro
```

💻 Código:

pyproject.toml:

```
version = "0.3"

<...>

"mlflow==2.22.1",
```

train_model.py:

```
import mlflow

<...>

# Set up MLflow experiment
mlflow.set_experiment("ml_classification")

# Set up Keras autolog
mlflow.keras.autolog()

with mlflow.start_run():
     # Log parameters to MLflow
     mlflow.log_params(params)

     <...>

     # Log preprocessing artifacts
     mlflow.log_artifact("artifacts/[features]_mean_imputer.joblib")
     mlflow.log_artifact("artifacts/[features]_scaler.joblib")

     <...>

     # Log the encoder
     mlflow.log_artifact("artifacts/[target]_one_hot_encoder.joblib")

     <...>
```

```
    # # Log metrics to MLflow
    # mlflow.log_metrics(metrics)
```

evaluate_model.py:

```
import os

<...>

import mlflow

<...>

# Set up MLflow experiment
mlflow.set_experiment("ml_classification")

# Get run_id for latest MLflow run
runs = mlflow.search_runs(
    experiment_ids=[os.getenv("MLFLOW_EXPERIMENT_ID")], order_by=["start_time DESC"]
)
run_id = runs.iloc[0].run_id

with mlflow.start_run(run_id=run_id):

    <...>

    # Log metrics (DVC)

    <...>

    # Log metrics (MLflow)
    mlflow.log_metrics(
        {
            "test_accuracy": report["accuracy"],
            "test_precision_weighted": report["weighted_avg"]["precision"],
            "test_recall_weighted": report["weighted_avg"]["recall"],
            "test_f1_weighted": report["weighted_avg"]["f1-score"],
        }
```

```
      )
```

.gitignore:

```
# MLflow
mlruns
```

.dockerignore:

```
# MLflow
mlruns
```

📖 AULA 4.2 Dashboard

Comandos:

Terminal:

```
mlflow ui
dvc repro
```

📖 AULA 4.3 Experimentos do DVC no MLflow

Comandos:

Terminal:

```
pip install -e .
mlflow ui
dvc exp run -S preprocess_data.test_size=0.25 --queue
dvc exp run -S train.dropout_rate=0.25 --queue
dvc exp run -S train.random_seed=24 --queue
dvc exp run --run-all
```

💻 Código:

pyproject.toml:

```
"python-dotenv==1.1.0",
```

.env:

```
MLFLOW_TRACKING_URI=http://localhost:5000
```

.gitignore:

```
# Project
.env
```

src/__init__.py:

```
from dotenv import load_dotenv

load_dotenv()
```

train_model.py:

```
# Setting MLflow if we are running a DVC experiment
is_experiment = os.getenv("DVC_EXP_NAME") is not None
extra_args = {}
if is_experiment:
    runs = mlflow.search_runs(
        experiment_ids=[os.getenv("MLFLOW_EXPERIMENT_ID")],
        filter_string="tags.dvc_exp = 'True'",
        order_by=["start_time DESC"],
    )
    if runs.empty:
        with mlflow.start_run() as parent_run:
            mlflow.set_tag("dvc_exp", True)
            parent_run_id = parent_run.info.run_id
    else:
        parent_run_id = runs.iloc[0].run_id
    run_name = os.getenv("DVC_EXP_NAME")
```

```
     extra_args = {
          "parent_run_id": parent_run_id,
          "run_name": run_name,
          "nested": True,
     }

with mlflow.start_run(**extra_args):
```

📖 AULA 4.4 Registro de artefatos

🖱️ Comandos:

Debug console:

```
latest_run
parent_run_id
```

💻 Código:

register_artifacts.py:

```
import logging
import mlflow
from mlflow.tracking import MlflowClient
import pandas as pd


logger = logging.getLogger("src.register_artifacts")

client = MlflowClient()


def get_best_run(experiment_id: str, parent_run_id: str) -> pd.Series:
    """Get the best child run based on test accuracy for a given parent run.

    Args:
        client: MLflow client instance
        parent_run_id: ID of the parent run
    Returns:
        The best run as a pandas Series
    """
    # Get all child runs for the parent
    child_runs = client.search_runs(
        experiment_ids=[experiment_id],
        filter_string=f"tags.mlflow.parentRunId = '{parent_run_id}'",
        order_by=["metrics.test_accuracy DESC"],
        max_results=1000
    )
    # Return the run with highest test accuracy
    return child_runs[0]

def register_model() -> None:
    """Register the model that was logged during training."""

    logger.info("Registering model from latest MLflow run")

    # Get the experiment ID for the 'ml_classification' experiment
    experiment_id = client.get_experiment_by_name("ml_classification").experiment_id

    # Get the latest run from the experiment
    latest_run = client.search_runs(
        experiment_ids=[experiment_id],
        order_by=["start_time DESC"],
        max_results=1
    )[0]

    # Check if the latest run has a parent run
    run_id = latest_run.info.run_id
    parent_run_id = latest_run.data.tags.get('mlflow.parentRunId')

    if parent_run_id:
        logger.info(f"Latest run has parent run ID: {parent_run_id}")
        best_run = client.search_runs(
            experiment_ids=[experiment_id],
            filter_string=f"tags.mlflow.parentRunId = '{parent_run_id}'",
            order_by=["metrics.test_accuracy DESC"],
            max_results=1
        )[0]
        run_id = best_run.info.run_id
        logger.info(f"Using best run {run_id} with test_accuracy:
{best_run.data.metrics['test_accuracy']}")

    # Register the model from the run
    logger.info("Registering model")
    try:
        client.create_registered_model("model")
    except mlflow.exceptions.MlflowException:
        logger.debug("Model already exists")

    model_uri = f"runs:/{run_id}/model"
    client.create_model_version(
        name="model",
        source=model_uri,
        run_id=run_id
    )
    logger.info("Registered model successfully")


def main() -> None:
    """Main function to orchestrate the model registration process."""
    register_model()
    logger.info("Model registration completed")


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

main.py:

```
import mlflow

<...>
```

```
from mlflow.tracking import MLflowClient

<...>

# Load model from registry
logger.info("Loading registered model from MLflow Model Registry")
self.model = mlflow.keras.load_model("models:/model/latest")

# Get run_id from model version metadata
client = MlflowClient()
run_id = client.get_registered_model("model").latest_versions[0].run_id

# Load related artifacts
logger.info(f"Loading artifacts from run {run_id}")
artifacts_dir = mlflow.artifacts.download_artifacts(run_id=run_id, artifact_path="")

imputer_path = os.path.join(artifacts_dir, "[features]_mean_imputer.joblib")
self.features_imputer = joblib.load(imputer_path)
scaler_path = os.path.join(artifacts_dir, "[features]_scaler.joblib")
self.features_scaler = joblib.load(scaler_path)
encoder_path = os.path.join(artifacts_dir, "[target]_one_hot_encoder.joblib")
self.target_encoder = joblib.load(encoder_path)

logger.info("Successfully loaded model and related artifacts")
```

📖 AULA 4.5 Repositório remoto - DagsHub

🖱️ Comandos:

Terminal:

```
pip install -e .
dvc repro
python -m src.register_artifacts
```

💻 Código:

pyproject.toml:

```
"dagshub==0.5.9",
```

.env:

```
MLFLOW_TRACKING_URI=<dagshub-url>
```

src/__init__.py:

```
import dagshub

<...>

# Initialize DagsHub with credentials
dagshub.init(
     repo_owner=<your-user-name>,
     repo_name=<your-repo-name>
)
```

📖 AULA 4.6 Docker

🔗 Links:

Criação de tokens DagsHub: https://dagshub.com/user/settings/tokens

🖱️ Comandos:

Terminal:

```
docker build -t ml-classifier .
docker run -p 5001:5001 ml-classifier
dvc push
```

💻 Código:

.dockerignore:

```
# Project
```

```
artifacts
models
```

app/__init__.py:

```
from dotenv import load_dotenv

load_dotenv()
```

.env:

```
DAGSHUB_USER_TOKEN=<dagshub-token>
```

📚 SESSÃO 5 - APACHE AIRFLOW

📖 AULA 5.1 Arquivo dag

🖱 Comandos:

Terminal:

```
pip install -e .
airflow db init
cd ~
cd airflow
ls -la
airflow users create --username admin --password admin --firstname
admin --lastname admin --role Admin --email admin@example.org
airflow webserver
airflow scheduler
cd ~/airflow
cat airflow.cfg
grep load_examples airflow.cfg
sed -i 's|load_examples = True|load_examples = False|g' airflow.cfg
grep dags_folder airflow.cfg
mkdir dags
```

```
cd dags
ln -s $PWD/ml_pipeline_dag.py ~/airflow/dags
```

💻 Código:

pyproject.toml:

```
version = "0.4"

<...>

"apache-airflow==2.10.5",
```

📖 AULA 5.2 Criação de DAG e tasks

🖱️ Comandos:

Debug console:

```
dvc_stages
dvc_tasks
airflow scheduler
airflow webserver
```

💻 Código:

dags/ml_pipeline_dag.py:

```
import yaml
from pathlib import Path

from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.python import PythonOperator


project_root = Path(__file__).resolve().parents[1]
```

```
def get_dvc_stages():
    dvc_yaml_path = project_root / "dvc.yaml"
    with open(dvc_yaml_path) as f:
        dvc_config = yaml.safe_load(f)
    return list(dvc_config["stages"].keys())

def register_artifacts_callable():
    from src.register_artifacts import main
    main()

default_args = {
    "owner": "airflow",
    "retries": 1,
}

with DAG(
    "ml_pipeline",
    default_args=default_args
) as dag:
    # DVC Pipeline Stages
    dvc_stages = get_dvc_stages()

    # Create tasks for each DVC stage
    dvc_tasks = []
    for stage in dvc_stages:
        task = BashOperator(
            task_id=f"dvc_{stage}",
            cwd=project_root,
            bash_command=f"dvc repro {stage}"
        )
        dvc_tasks.append(task)

    # Register artifacts in MLflow
    register_artifacts = PythonOperator(
        task_id="register_artifacts",
        python_callable=register_artifacts_callable
    )
    # Deploy model by building and running Docker container
    create_app_image = BashOperator(
        task_id="create_app_image",
        cwd=project_root,
        bash_command = "docker build -t ml-classifier ."
    )

    # Set dependencies
    for i in range(len(dvc_tasks) - 1):
        dvc_tasks[i] >> dvc_tasks[i + 1]

    # Connect the last DVC task to register_artifacts and then to create_app_image
    dvc_tasks[-1] >> register_artifacts >> create_app_image
```

📖 AULA 5.3 Dashboard

🖱️ Comandos:

Terminal:

```
docker image list
```

📖 AULA 5.4 Docker compose

💻 Código:

docker-compose.airflow.yaml

```
services:
  postgres:
    image: postgres:13
    environment:
      - POSTGRES_USER=airflow
      - POSTGRES_PASSWORD=airflow
      - POSTGRES_DB=airflow
    networks:
      - backend

  dind:
    image: docker:24.0.5-dind
    privileged: true
    expose:
      - 2375
    environment:
      - DOCKER_TLS_CERTDIR=
    networks:
      - backend

  airflow:
    build:
      context: .
      dockerfile: Dockerfile.airflow
    restart: always
    depends_on:
      - postgres
      - dind
    ports:
      - "8080:8080"
    environment:
      - AIRFLOW__CORE__EXECUTOR=LocalExecutor
      -
AIRFLOW__DATABASE__SQL_ALCHEMY_CONN=postgresql+psycopg2://airflow:airflow@postgres/airflow
      - AIRFLOW__CORE__LOAD_EXAMPLES=false
      - AIRFLOW__CORE__DAGS_FOLDER=/home/airflow/mlops_project/dags
      - DOCKER_HOST=tcp://dind:2375
      - DOCKER_HUB_USERNAME=${DOCKER_HUB_USERNAME}
      - DOCKER_HUB_TOKEN=${DOCKER_HUB_TOKEN}
      - DAGSHUB_USER_TOKEN=${DAGSHUB_USER_TOKEN}
    networks:
      - backend
    command: bash -c "
      airflow db init &&
      airflow users create --username admin --password admin --firstname admin --lastname
admin --role Admin --email admin@example.com &&
      (airflow scheduler &) &&
      airflow webserver
      "

networks:
  backend:
```

Dockerfile.airflow:

```
FROM apache/airflow:2.10.5-python3.12

USER root

RUN groupadd airflow && usermod -aG airflow airflow

COPY . /home/airflow/mlops_project
RUN chown -R airflow:airflow /home/airflow/mlops_project

USER airflow

WORKDIR /home/airflow/mlops_project

RUN pip install .
RUN dvc init --no-scm
```

.env:

```
DOCKER_HUB_USERNAME=<your-username>
DOCKER_HUB_TOKEN=<your-token>
```

ml_pipeline_dag.py:

```
# bash_command = "docker build -t ml-classifier ."
bash_command="""
docker build -t ${DOCKER_HUB_USERNAME}/ml-classifier .
echo ${DOCKER_HUB_TOKEN} | docker login -u ${DOCKER_HUB_USERNAME} --password-stdin
docker push ${DOCKER_HUB_USERNAME}/ml-classifier
"""
```

📖 AULA 5.5 Airflow conteinerizado

🔗 Links:

Repositórios no DockerHub: https://hub.docker.com/repositories/<your-username>

🖱️ Comandos:

Terminal:

```
docker compose -f docker-compose.airflow.yaml build
docker image list
docker compose -f docker-compose.airflow.yaml up
docker run -p 5001:5001 <your-username>/ml-classifier
```