**Этап 1: Планирование пайплайна**

**Цель проекта**

Разработка автоматизированного ETL-пайплайна для предсказания наличия злокачественной опухоли молочной железы на основе данных Breast Cancer Wisconsin Diagnostic. Проект реализует полную обработку данных, обучение модели, расчет метрик и оркестрацию через Apache Airflow.

**ML-задача**

**Тип задачи:** бинарная классификация  
**Целевая переменная:** diagnosis  
**Цели:** классификация опухолей на M (malignant – злокачественная) и B (benign – доброкачественная) на основе набора признаков, вычисленных из изображения клеток опухоли.

**Архитектура пайплайна**

graph TD

A[Загрузка данных] --> B[Очистка и предобработка]

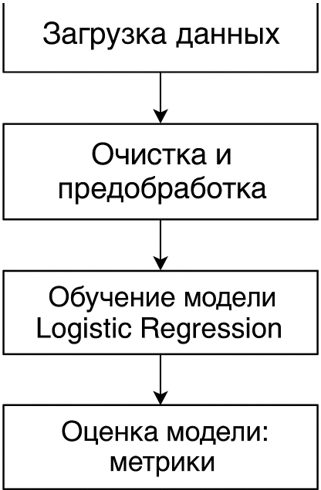
B --> C[Обучение модели Logistic Regression]

C --> D[Оценка модели: метрики]

D --> E[Сохранение результатов]

**Описание шагов пайплайна**

| **Шаг** | **Название** | **Описание** |
| --- | --- | --- |
| 1 | Загрузка данных | Чтение CSV-файла, базовая проверка структуры и размеров |
| 2 | Предобработка | Удаление лишних колонок (например, id), преобразование diagnosis в 0/1, нормализация признаков |
| 3 | Обучение модели | Обучение модели LogisticRegression с использованием sklearn |
| 4 | Оценка качества | Расчет метрик: Accuracy, Precision, Recall, F1-score |
| 5 | Сохранение | Сохранение обученной модели и метрик в формате CSV/JSON |

****

**Этап 2: Разработка ETL-компонентов**

**Структура папки etl/**

etl/

├── load\_data.py

├── preprocess\_data.py

├── train\_model.py

├── evaluate\_model.py

└── save\_results.py

**load\_data.py**

import pandas as pd

def load\_dataset(path):

df = pd.read\_csv(path)

return df

if \_\_name\_\_ == "\_\_main\_\_":

df = load\_dataset("data/breast\_cancer\_dataset.csv")

print(df.info())

**preprocess\_data.py**

import pandas as pd

from sklearn.preprocessing import StandardScaler

def preprocess(df):

df = df.drop(columns=["id", "Unnamed: 32"], errors='ignore')

df["diagnosis"] = df["diagnosis"].map({"M": 1, "B": 0})

X = df.drop("diagnosis", axis=1)

y = df["diagnosis"]

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

return X\_scaled, y, scaler

if \_\_name\_\_ == "\_\_main\_\_":

from load\_data import load\_dataset

df = load\_dataset("data/ breast\_cancer\_dataset.csv ")

X, y, \_ = preprocess(df)

print(f"X shape: {X.shape}, y shape: {y.shape}")

**train\_model.py**

from sklearn.linear\_model import LogisticRegression

import joblib

def train(X, y):

model = LogisticRegression(max\_iter=1000)

model.fit(X, y)

return model

if \_\_name\_\_ == "\_\_main\_\_":

from load\_data import load\_dataset

from preprocess\_data import preprocess

df = load\_dataset("data/ breast\_cancer\_dataset.csv ")

X, y, \_ = preprocess(df)

model = train(X, y)

joblib.dump(model, "results/model.joblib")

**evaluate\_model.py**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import json

def evaluate(model, X, y):

preds = model.predict(X)

metrics = {

"accuracy": accuracy\_score(y, preds),

"precision": precision\_score(y, preds),

"recall": recall\_score(y, preds),

"f1\_score": f1\_score(y, preds),

}

return metrics

if \_\_name\_\_ == "\_\_main\_\_":

import joblib

from load\_data import load\_dataset

from preprocess\_data import preprocess

df = load\_dataset("data/ breast\_cancer\_dataset.csv ")

X, y, \_ = preprocess(df)

model = joblib.load("results/model.joblib")

metrics = evaluate(model, X, y)

print(metrics)

**save\_results.py**

import json

import os

def save\_metrics(metrics: dict, filepath="results/metrics.json"):

os.makedirs(os.path.dirname(filepath), exist\_ok=True)

with open(filepath, "w") as f:

json.dump(metrics, f, indent=4)

if \_\_name\_\_ == "\_\_main\_\_":

from train\_model import train

from evaluate\_model import evaluate

from load\_data import load\_dataset

from preprocess\_data import preprocess

df = load\_dataset("data/ breast\_cancer\_dataset.csv ")

X, y, \_ = preprocess(df)

model = train(X, y)

metrics = evaluate(model, X, y)

save\_metrics(metrics)

**Этап 3: Оркестрация пайплайна с помощью Airflow**

**Структура**

dags/

└── pipeline\_dag.py # основной DAG

**Название DAG**

breast\_cancer\_etl\_pipeline

**Описание зависимостей между задачами**

Пайплайн состоит из 5 задач, связанных по цепочке:

1. **load\_data** — загрузка CSV-файла
2. **preprocess\_data** — очистка, нормализация и кодировка
3. **train\_model** — обучение модели логистической регрессии
4. **evaluate\_model** — расчет метрик
5. **save\_results** — сохранение модели и метрик

Логирование

Логи автоматически сохраняются в папку logs/, настроенную в конфигурации Airflow.

Каждая задача имеет свой собственный лог-файл, доступный как из UI, так и в файловой системе.

**Этап 4: Интеграция с хранилищем**

**Структура хранения**

results/

├── model.joblib # обученная модель

├── metrics.json # метрики качества

├── metrics.pkl # временный бинарный формат

└── preprocessed.pkl # масштабированные данные и целевая переменная

**Формат сохранения**

* **Модель** сохраняется в формате .joblib для последующей загрузки в любой момент.
* **Метрики** сохраняются:
  + в metrics.pkl (временный файл для Airflow),
  + и в metrics.json — для удобного анализа.

Пример содержимого metrics.json:

{

"accuracy": 0.9649,

"precision": 0.9506,

"recall": 0.9756,

"f1\_score": 0.9629

}

**Реализация в коде**

import json

import os

def save\_metrics(metrics: dict, filepath="results/metrics.json"):

os.makedirs(os.path.dirname(filepath), exist\_ok=True)

with open(filepath, "w") as f:

json.dump(metrics, f, indent=4)

**Этап 5. Анализ ошибок и устойчивости**

Потенциальные точки сбоя и исключения

| **Этап пайплайна** | **Возможный сбой** | **Предусмотренные меры** |
| --- | --- | --- |
| **Загрузка данных** | Файл не найден, ошибка чтения | Обработка через try/except, вывод в лог, настройка retries=2 |
| **Предобработка** | Отсутствие столбца, NaN, неверные типы | Проверка схемы, удаление лишних колонок, логгирование ошибок |
| **Обучение модели** | Ошибка конвергенции, пустой X/y | Контроль размера X/y, лог ошибки |
| **Оценка метрик** | Деление на ноль, невалидные предсказания | Проверка модели перед вызовом .predict() |
| **Сохранение результатов** | Ошибка записи на диск | Проверка прав доступа, создание папки results/ если нет |

**Ответы на контрольные вопросы**

**1. Где может «упасть» процесс?**  
На любом этапе: при загрузке данных, при некорректной структуре, при ошибке обучения или сохранения.

**2. Какие исключения могут возникнуть?**  
FileNotFoundError, ValueError, TypeError, KeyError, ConvergenceWarning, PermissionError.

**3. Что произойдет при потере соединения с источником данных?**  
В текущей реализации используется локальный файл, но при переходе на API можно задать retries, timeout и использовать Airflow Sensor для проверки доступности.

**4. Что будет, если источник отдает невалидные данные?**  
Реализована предобработка: проверка на наличие целевой переменной, удаление неиспользуемых колонок, масштабирование. Можно усилить pydantic-валидацией.

**5. Что произойдет, если модель не обучается или выдает ошибку?**  
Ошибка фиксируется в логах Airflow, пайплайн останавливается, не переходя к следующим шагам. Используются retries.

**Архитектура устойчивости**

**Изолированность шагов:**  
Каждый шаг реализован как отдельный Python-модуль и Airflow-задача. Даже при сбое одной задачи остальные не страдают.

**Логирование:**  
Логи Airflow по каждому шагу сохраняются автоматически. Также возможно логирование в файл через logging.

**Настройки отказоустойчивости в Airflow:**

default\_args = {

'owner': 'airflow',

'start\_date': datetime(2023, 1, 1),

'retries': 2,

'retry\_delay': timedelta(minutes=5),

'depends\_on\_past': False

}

**Валидация и защита:**

* Проверка наличия ключевых колонок (diagnosis);
* Преобразование категориальных признаков;
* Обработка NaN и пустых массивов.

**Идеи для улучшения устойчивости**

* Использовать pydantic для строгой валидации структуры данных;
* Добавить email или Slack-уведомление при падении DAG;
* Сохранять дамп проблемных входных данных для отладки;
* Разделить тренировку и прод-предсказания в разные DAG-и.