# 데이터마이닝 팀과제 자료 조사

2025-03-25

# Table of contents

# 1. Airflow 소개 (1분 30초)

## 1.1 Airflow란?

- **Apache Airflow**: 워크플로우 작성, 스케줄링 및 모니터링을 위한 오픈소스 플랫폼
- 2014년 Airbnb에서 개발, 2016년 Apache 재단으로 이관
- Python으로 작성된 데이터 파이프라인 오케스트레이션 도구

## 1.2 핵심 개념

- **DAG(Directed Acyclic Graph)**: 작업 흐름을 표현하는 방향성 비순환 그래프
- **Task**: 개별 작업 단위 (데이터 추출, 변환, 적재 등)
- **Operator**: 작업 실행 방법을 정의 (PythonOperator, BashOperator 등)
- **Scheduler**: 작업 실행 시점 관리
- **Web Server**: 대시보드를 통한 모니터링 인터페이스

## 1.3 데이터 분석에서의 중요성

- 복잡한 데이터 워크플로우의 자동화 및 오케스트레이션
- 작업 간 의존성 관리 및 모니터링
- 실패한 작업의 자동 재시도 및 알림

# 2. 현업 데이터 분석 문제점과 Airflow 해결책 (2분)

## 2.1 데이터 분석 현업의 문제점

- **수동 프로세스의 한계**: 반복 작업의 비효율성과 인적 오류
- **복잡한 의존성**: 여러 시스템과 데이터 소스 간의 조율 어려움
- **스케줄링 이슈**: 정기적인 데이터 처리와 의존성 관리
- **에러 처리**: 실패 시 복구 및 알림 메커니즘 부재
- **모니터링 부족**: 작업 진행 상황과 성능 추적 어려움

## 2.2 Airflow의 해결책

- **코드형 워크플로우**: Python으로 DAG 정의 (GitOps 가능)
- **시각적 모니터링**: 웹 UI를 통한 직관적인 작업 흐름 파악
- **스마트 스케줄링**: Cron 기반 스케줄링 + 의존성 기반 실행
- **강력한 확장성**: 다양한 시스템과의 통합 (AWS, GCP, Azure, Databricks 등)
- **견고한 에러 처리**: 자동 재시도, 알림, 대체 경로 설정

# 3. 주요 활용 사례 (3분 30초)

## 3.1 ETL 프로세스 자동화

- **사례**: 일일 매출 데이터 통합 파이프라인

## 3.2 매출 데이터 ETL DAG 예시

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import warnings

warnings.filterwarnings("ignore")

default_args = {
    'owner': 'data_team',
    'depends_on_past': False,
    'retries': 3,
    'retry_delay': timedelta(minutes=5)
}

def extract_sales_data(**kwargs):
    return {"sales_data": "extracted"}

def transform_data(**kwargs):
    return {"transformed_data": "ready"}

def load_to_warehouse(**kwargs):
    print("Data loaded successfully")

with DAG(
    'daily_sales_etl',
    default_args=default_args,
    schedule_interval='0 2 * * *',
    start_date=datetime(2023, 1, 1)
) as dag:
```

```
    extract_task = PythonOperator(
        task_id='extract_sales_data',
        python_callable=extract_sales_data
    )

    transform_task = PythonOperator(
        task_id='transform_sales_data',
        python_callable=transform_data
    )

    load_task = PythonOperator(
        task_id='load_to_warehouse',
        python_callable=load_to_warehouse
    )
    extract_task >> transform_task >> load_task
```

- **효과**:
  - 여러 데이터 소스(POS, 온라인 스토어, 외부 판매 채널)를 자동 통합
  - 일관된 데이터 처리 및 변환 보장
  - 작업 의존성 자동 관리

## 3.3 데이터 품질 관리

- **사례**: 데이터 검증 및 알림 시스템
  - 데이터 완전성, 정확성, 일관성 검증
  - 임계값 초과 시 자동 알림
- **구현 방식**:
  - Great Expectations과 같은 도구와 통합
  - 검증 실패 시 Slack, Email로 알림
  - 데이터 품질 측정 및 대시보드 제공

## 3.4 ML 모델 파이프라인

- **사례**: 추천 모델 정기 학습 및 배포
  - 새로운 사용자 행동 데이터 수집
  - 정기적인 모델 재학습
  - 성능 검증 후 자동 배포
  - 모델 버전 관리 및 롤백
- **효과**:

- 모델 신선도 유지
- 일관된 학습 및 평가 파이프라인
- 버전 관리 및 추적 용이성

# 4. 실습 데모와 기술적 차별점 (1분 30초)

## 4.1 간단한 DAG 구조 데모

- 웹 UI 통한 DAG 시각화
- 태스크 상태 확인 및 관리
- 로그 및 오류 모니터링

## 4.2 경쟁 도구와의 차이점

- **Airflow vs Prefect**:
  - Airflow: 성숙한 생태계, 풍부한 커넥터
  - Prefect: 더 현대적인 API, 로컬 개발 편의성
- **Airflow vs Luigi**:
  - Airflow: 풍부한 UI, 스케줄링 강점
  - Luigi: 더 가벼운 구조, 더 간단한 설정

## 4.3 Airflow의 기술적 장단점

- **장점**:
  - 광범위한 커뮤니티와 풍부한 사용 사례
  - 다양한 시스템과의 통합 용이성
  - 강력한 모니터링 및 알림 기능
- **단점**:
  - 상대적으로 가파른 학습 곡선
  - 가벼운 워크로드에는 오버스펙일 수 있음
  - 설정 및 관리의 복잡성

# 5. 결론 및 미래 전망 (30초)

## 5.1 요약

- Airflow는 데이터 분석 현업에서 핵심 워크플로우 관리 도구로 자리매김
- ETL, 데이터 품질 관리, ML 파이프라인 등 다양한 활용 사례
- 채용 시장에서 지속적으로 수요가 증가하는 기술

## 5.2 미래 전망

- 클라우드 네이티브 환경으로의 발전
- Kubernetes와의 통합 강화
- 데이터 거버넌스 기능 확장
- 보다 사용자 친화적인 인터페이스로 발전

## 5.3 마무리

- 데이터 기반 의사결정이 중요해짐에 따라 Airflow의 중요성도 계속 증가할 전망
- 데이터 엔지니어링과 사이언스를 연결하는 핵심 도구로 발전