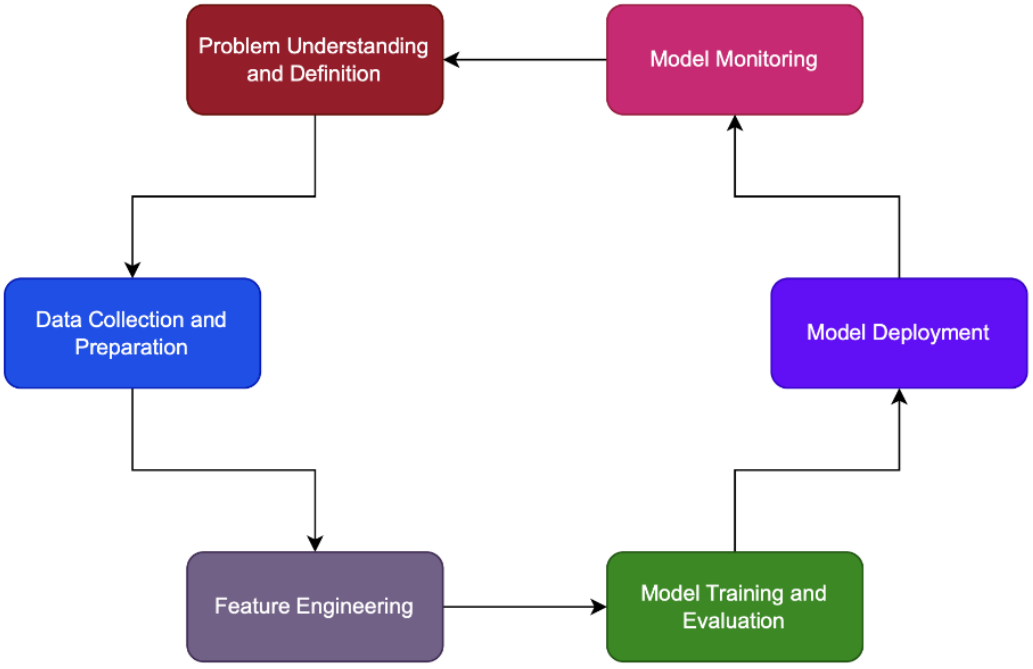
End-to-End Machine Learning

Design and Exploration

1. Designing an End-to-End Machine Learning Use Case



- Machine learning life cycle
  - 문제 이해 및 정의와 명확한 목표 설정
  - 데이터 수집 및 준비
  - 모델 개발 및 튜닝
  - 모델 평가
  - 배포
  - 모니터링

  → 계속 반복적으로 이루어짐. 시간에 지남에 따라 데이터가 변하기 때문

2. Exploratory Data Analysis

해결하려는 문제와 관련된 데이터 수집이 이루어진 후 데이터에 대한 이해를 위해 EDA 과정이 필요함

- pandas 메서드를 사용한 데이터 유형 확인, 계층 불균형, 누락된 값, 이상치, 데이터 시각화

3. Data preparation

- 데이터 세트에 누락된 값, 이상치, 불균형이 있는 경우에 어떻게 처리할 것인지
- 예시) imputation 사용, pandas - fillna(), drop_duplicates(), drop()

Model Training and Evaluation

1. Feature engineering and selection

- 데이터 준비 단계와 함께 있음
- ML 모델의 성능을 향상시키는 피처를 만드는 프로세스임
- 올바른 피처를 선택하는 것이 중요함
  - 예측 정확도를 개선하기 위해 모델링에 관련 피처를 사용해야 함
  - 유사한 지표를 나타내는 여러 피처를 사용하는 것도 도움 안됨
- 방법
  - 정규화
    - 숫차 피처를 0~1 사이의 스케일로 조정함
    - 피처 범위가 다르고 KNN 또는 입력 스케일에 민감한 알고리즘에 유용함
  - 표준화
    - 피처의 평균을 0으로, 분산을 1로 조정함
    - SVM, 선형회귀와 같은 알고리즘에 유리함
- feature 선택 도구
  - sklearn 의 feature_selection

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Define the random forest model and fit to the training
data
rf = RandomForestClassifier(n_jobs=-1, class_weight='bal
anced', max_depth=5)
rf.fit(X_train, y_train)

# Define the feature selection object
model = SelectFromModel(rf, prefit=True)

# Transform the training features
X_train_transformed = model.transform(X_train)

original_features = heart_disease_df.columns[:-1]  # Ass
uming the last column is the target
print(f"Original features: {original_features}")

# Select the features deemed important by the SelectFrom
Model
features_bool = model.get_support()

selected_features = original_features[features_bool]
print(f"\nSelected features: {selected_features}")

# Creating a DataFrame to display feature importances
feature_importance = pd.DataFrame({
    "feature": selected_features,
    "importance": rf.feature_importances_[features_bool]
})

# Plotting feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance["feature"], feature_importan
ce["importance"])
plt.xlabel('Importance')
plt.title('Feature Importances')
plt.show()
```

2. Model training

- 모델을 선택할 때 데이터에 잘 맞는 간단한 모델을 선택해야 한다.
- 모델
    - Logistic Regression
    - Support Vector Classifier
    - Decision Tree
    - Random Forest
    - Deep learning models → Cnn, Gpt
    - K-Nearest Neighbors(KNN)
    - XGBoost
- 학습 시킬 때는 학습 데이터와 테스트 데이터를 7:3 or 8:2 비율로 나누는 것이 좋음.

3. **Logging experiments on MLflow**

- 실험 로킹 및 관리는 중요함
- MLflow
    - 머신 러닝 라이프사이클을 관리하기 위한 오픈 소스 플랫폼
    - ML 엔지니어가 실험 결과를 추적하고 비교하고, 코드를 재현 가능한 실행으로 패키징하고, 모델을 공유하고 배포할 수 있도록 설계됨
    - 웹 기반 사용자 인터페이스 제공

```
- MLflow steps
# Initialize the MLflow experiment
# 실험 이름 설정 -> 지정된 이름으로 실험이 생성되어 해당 특정 범주나 프로젝트의 모든 실험에 대한 작업 공간이 제공됨
mlflow.set_experiment("Logistic Regression Heart Disease Prediction")

# Start a run, log model coefficients and intercept
# 실험 실행 -> 코드의 단일 실행을 나타내며 매개변수, 메트릭, 태그 및 훨씬 더 많은 정보를 포함할 수 있음
with mlflow.start_run():
    for idx, coef in enumerate(model.coef_[0]):
        mlflow.log_param(f"coef_{idx}", coef)
```

```
    mlflow.log_param("intercept", model.intercept_[0])

    run_id = mlflow.active_run().info.run_id
    print(run_id)
```

4. Model evaluation and visualization

- 모델 평가 방법
    - Accuracy
    - Confusion Matrix
    - Cross validation → k-fold
    - 하이퍼 파라미터 튜닝

Model Deployment

1. Testing a model

- 모델을 훈련하고 평가한 후 모델을 배포하고 생산적인 용도로 사용할 수 있게 해야함
- 사전 배포의 중요한 부분은 예상대로 추론이 되는지 테스트하는 것이 필수적임
    - 단위테스트
    - 파이썬 라이브러리인 unittest 사용
    - 단위 테스트를 작성하는 것은 많은 작업처럼 보일 수 있지만, 안정적이고 견고한 시스템을 유지하는 데 매우 중요함
    - 너무 많은 테스트나 중복된 테스트를 작성하는 것은 도움 되지 않음

```
import unittest
import numpy as np

# Create a class called TestModelInference
# unittest.TestCase 를 상속받아 테스트 케이스를 구현
class TestModelInference(unittest.TestCase):
    def setUp(self):
        self.model = model

        # set X_test as a class attribute
```

```
        self.X_test = X_test

    # define a test for prediction output values
    # 실제 테스트 수행
    def test_prediction_output_values(self):
        print("Running test_prediction_output_values test c
ase")

        # Get model predictions
        y_pred = self.model.predict(self.X_test)
        unique_values = np.unique(y_pred)
        for value in unique_values:
            self.assertIn(value, [0, 1])
```

2. **Architectural components in end-to-end machine learning frameworks**

- 피처 스토어와 모델 레지스트리의 중요성
- 피처 스토어
  - 큐레이팅되고 변환된 피처를 저장하는 중앙 저장소 역할을 함
  - 다양한 모델에서 피처의 일관성을 보장하여 피처 계산 중복을 방지하고 피처 공유 및 검색을 가능하게 함
  - 훈련 중에 수행된 동일한 피처 변환 및 계산이 프로덕션 중에 수행되도록 보장하는 추가 이점이 있음
  - Feast 는 Gojek 에서 개발한 오픈 소스 프로젝트
    - 피처 세트를 사용하여 피처를 정의하고 등록할 수 있음
- 모델 레지스트리
  - 머신 러닝 모델을 위한 일종의 버전 제어 시스템
  - 머신 러닝 모델의 다양한 버전을 관리하고 추적 가능
  - MLflow 를 사용해도 머신 러닝 실험을 관리하고 추적하고, 모델 성능 메트릭을 기록하고, 교차 비교를 위해 훈련된 모델 아티팩트를 저장할 수도 있음

3. **Packaging and containerization**

- 임상 배포를 위해 모델 패키징
- 배포 단계에는 모델과 종속성을 다양한 환경에서 쉽게 실행할 수 있는 독립형 단위로 패키징하는 것이 포함됨→컨테이너화

- Docker
    - 컨테이너를 사용하여 쉽게 만들고 배포할 수 있는 플랫폼

4. **Continuous integration and continuous deployment (CI/CD)**

- 컨테이너화된 모델을 배포하는 방법
- CI/CD 원칙
    - 지속적인 통합(CI)은 코드 변경 사항을 중앙의 비생산 저장소에 정기적으로 병합하는 것
    - 지속적인 배포(CD)는 일반적으로 지속적인 통합을 따르고 코드베이스의 업데이트 또는 변경 사항을 자동으로 생산에 배포하는 것
- 머신러닝의 CI/CD
    - 프로덕션에 적합한 ML 모델을 개선하고 유지하는 데 중요함
    - 새로운 데이터로 모델을 정기적으로 학습하고, 모델의 성능을 테스트하고, 예상대로 또는 더 나은 성능을 보이는 경우 모델을 배포하는 것을 포함함

Model Monitoring

1. Model Monitoring

- 머신 러닝 모델을 유지 관리하고 개선하는 데 필수적임
- 추론 시간 및 그 이후에 성능 지표를 로깅하고 시각화하여 수행됨
- 파이썬으로 로깅
    - 로깅은 디버깅뿐만 아니라 모델 예측을 기록하고 성능을 장기적으로 분석하는 데 필수적임
    - logging 라이브러리를 사용해 테스트 세트에서의 예측 결과를 기록하고, 추세나 이상 현상을 감지할 수 있음
- 시각화를 사용해서도 모델 성능 변화를 확인할 수 있음

2. Data drift

- 프로덕션에서 많은 모델에 영향을 미침
- 모델의 입력 피처의 통계적 속성이 시간이 지남에 따라 변하는 현상 → 최근의 데이터 분포에서 모델을 학습하는 것이 중요함
- data drift 해결
    - Kolmogorov-Smirnov test

        - 두 데이터 세트 분포 또는 열의 통계적 속성을 비교하고 데이터 드리프트를 나타낼 수 있는 두 가지 간의 유의미한 차이점을 강조함
        - scipy 라이브러리의 ks_2samp() 함수 사용 → p-value 가 0.05보다 작으면 데이터 드리프트가 발생했을 가능성이 큼

3. **Feedback loop, re-training, and labeling**

- 피드백 루프는 모델 예측이나 성능 데이터와 같은 시스템의 다양한 측정값을 입력으로 시스템에 다시 보내는 시스템
- 출력에 대한 모델 예측을 실제로 수행하는 대신, 시스템이 현재 결과나 통계를 지속적으로 사용하여 미래의 동작을 안내하고 형성함
- 피드백 루프 구현
    - 수동, 반자동 또는 완전 자동으로 구현 가능
    - 데이터 드리프트를 감지하면 모델을 최신 레이블이 지정된 데이터의 일부에 대해 재교육하여 오래되지 않도록 하는 피드백 루프를 설정할 수 있음
- 피드백 루프의 위험
    - 모델의 출력이 직접 또는 간접적으로 입력에 영향을 미칠 수 있을 때 위험함
    - 자동화할 때는 신중해야 함

4. Serving the model

- 온디바이스 서빙
    - 인터넷을 통해 모델을 액세스 할 수 없을 때 사용
    - 온디바이스 또는 애플리케이션의 일부로 모델을 제공하는 것이 더 합리적일 수 있음
    - 엣지 컴퓨팅 애플리케이션에서 사용됨 → 안정적인 네트워크 연결 없이 디바이스에서 실행
    - 장점
        - 외부 서버에 의존할 필요가 없으므로 응답시간이 빠름
        - 인터넷 액세스가 필요하지 않기 때문에 데이터 침해 위험을 최소화함
    - 단점
        - 에지 디바이스는 메모리와 처리 능력이 제한될 수 있음 → 모델이 최적화되고 가벼워야 함

        - 클라우드 인프라가 제공하는 종류의 확장성에서 이점을 얻지 못함 → 업데이트가 제한적이어서 사용 통계, 성능 지표, 잠재적 모델 드리프트를 집계하는 것이 어려움
    - 구현 전략
        - pre-trained 모델을 활용하여 파인튜닝
        - TensorFlow Lite, Core ML, ONNX Runtime 과 같은 온디바이스 및 에지 배포에 맞게 조정된 머신 러닝 프레임 워크 사용