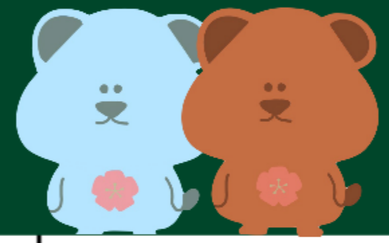


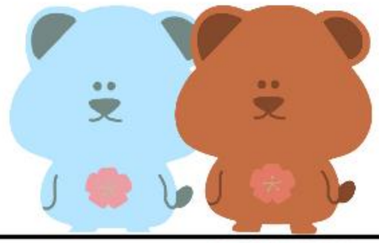
# 주차수요 예측 AI 경진대회

한예송, 홍재령

EWHA EURON

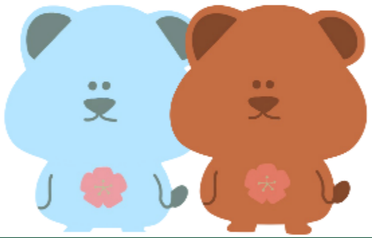
# 목차





EWHA EURON

# Introduction



EWHA EURON

# #1 대회 소개

**주제 : 유형별 임대주택 설계 시 단지 내 적정 주차 수요 예측**

**데이터셋**

- train.csv - 학습용 데이터 (단지코드, 총세대수, 지역, 전용면적 등)
- test.csv - 테스트 데이터
- age_gender_info.csv - 지역 임대주택 나이별, 성별 인구 분포
- sample_submission.csv - 제출 양식

**평가산식 :** MAE(Mean Absolute Error)

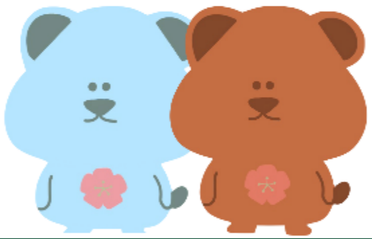
현재 장래주차수요는 '주차원단위' 와 '건축연면적' 을 기초로 하여 산출됨
'주차원단위' - 신규 건축예정 부지 인근의 유사 단지를 피크 시간대 방문하여 주차된 차량대수를 세는 방법으로 조사
→ 인력조사로 인한 오차발생, 현장조사 시점과 실제 건축시점과의 시간차 등의 문제로 과대 또는 과소 산정의 가능성을 배제할 수 없음

EWHA EURON

# #1 1등 솔루션 소개

- EDA를 굉장히 많이 함
  → 데이터를 이해하고, 이상치들을 확인하는 데에 큰 도움이 됨
- **Pycaret으로 모델 생성**
  - 해당 대회가 워낙 데이터도 이상하고 이상치들이 많이 존재해 모델을 너무 잘 만들었다간 private 점수가 많이 안 좋을 것으로 예상
  - model에 대한 신경↓ & **Pycaret**에 집어넣고 **가장 성능이 잘 나오는 5개로 블랜딩해서 최종 모델을 생성**
  - 모델 튜닝을 더 진행하고 파라미터 값 등을 더 좋게 변경해줬을 때 public score가 더 떨어지거나 cv 값이 더 떨어짐

# AutoML & Pycaret



EWHA EURON

# #2 AutoML이란?

## Automated Machine Learning

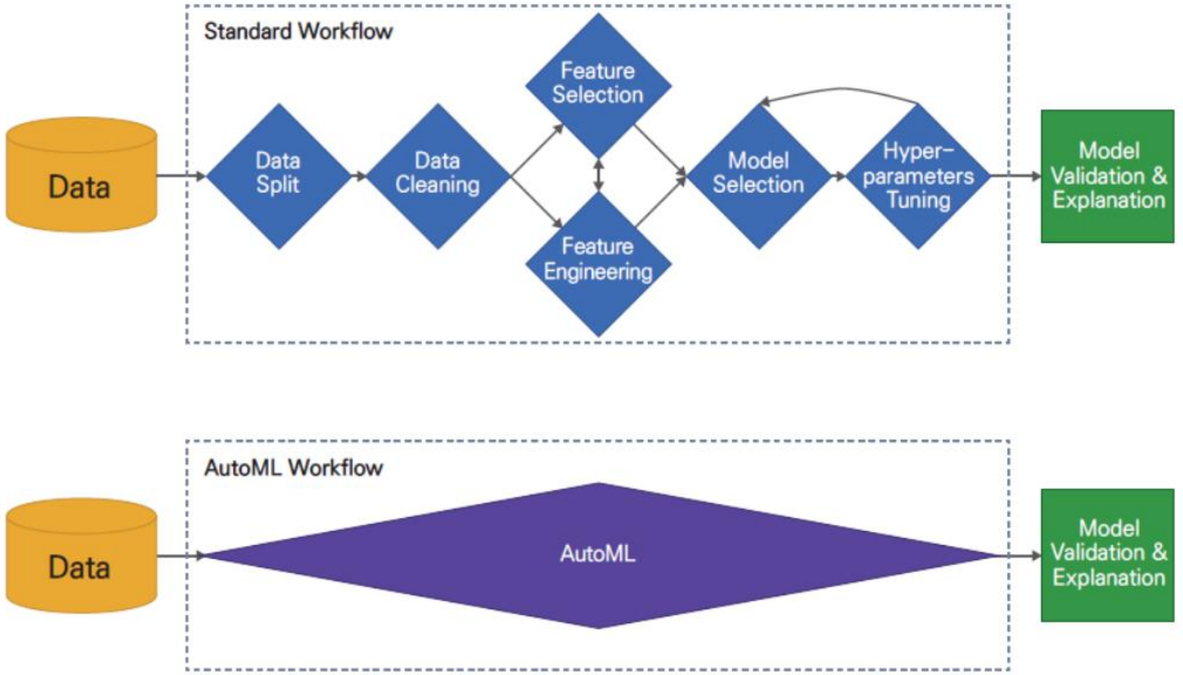
**시간 소모적이고 반복적인 기계 학습 모델 개발 작업을 자동화하는 프로세스**

**( = 머신러닝 모델을 개발을 자동화하는 분야)**

머신러닝 모델 개발 과제에서 반복적으로 수행하게 되는
데이터 분할, 정제, 특징 선택 및 추출, 모델 선택, 하이퍼 파라미터 튜닝 등을 함수화하여 자동화
→ 비효율적인 작업을 최대한 자동화하여 생산성 & 효율 ↑
→ 모델 개발자의 개입을 최소화 → 효율적인 모델 개발

## AutoML 기술로 해결하고자 하는 문제

1) 다양한 알고리즘들과 연관된 하이퍼 파라미터들을 실험하고 성능을 비교하여 최상의 성능을 갖는 모델을 찾는 과정을 자동화 (Combined Algorithm Selection and Hyper-parameter optimization, CASH)
2) 문제에 적합한 architecture를 찾는 과정을 자동화 (Neural Architecture Search, NAS)



# #2 Pycaret이란?

**기존에 있던 Scikit-learn, spaCy 등 여러가지 머신러닝 라이브러리를 ML High-Level API로 제작한 라이브러리**
**몇 줄만에 데이터 분석 및 머신러닝 모델 성능 비교 & Log를 생성하여 이력을 남겨줌**

- xgb, lightgbm, extratrees 등의 알고리즘을 이것저것 넣어서 성능을 평가해보고 모델별로 성능을 나열
  → 그 중에 가장 좋은 모델을 선택
  → 모델 최적화하는 과정을 단 몇 줄로 만듦
- classification, regression, clustering, anomaly detection, NLP, association rules, datasets 등의 패키지

**적용 순서**

1. 데이터 준비 (setup)
2. 모델 생성 및 비교
3. 모델 최적화
4. 학습된 모델 분석

Classification

```python
# load dataset
import pandas as pd
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# init setup
from pycaret.classification import *
s = setup(train, target = 'target')

# model training and selection
best = compare_models()

# analyze best model
evaluate_model(best)

# predict on new data
predictions = predict_model(best, data = test)

# save best pipeline
save_model(best, 'my_best_pipeline')
```

# #2 Pycaret - (1) 데이터 준비

## setup()

로드되어진 데이터를 머신러닝에 사용할 수 있도록 **로드 및 전처리하는 기능**

데이터 전처리 단계에서 적용하는 기법들에 대해 파라미터로 간단하게 사용할 수 있도록 구현됨

```
reg = setup(data=train_data,
            target='주차면수대비등록확률',
            session_id = 201,
            numeric_imputation = 'mean',
            fold_shuffle = True,
            numeric_features=list(train_data.drop(columns = ['주차면수대비등록확률']).columns),
            ignore_low_variance = True,
            combine_rare_levels = True, rare_level_threshold = 0.05,
            remove_multicollinearity = True, multicollinearity_threshold = 0.90,
            normalize = True,
            silent= True)
```

| | Description | Value |
|---|---|---|
| 0 | session_id | 201 |
| 1 | Target | 주차면수대비등록확률 |
| 2 | Original Data | (397, 82) |
| 3 | Missing Values | False |
| 4 | Numeric Features | 81 |
| 5 | Categorical Features | 0 |
| 6 | Ordinal Features | False |
| 7 | High Cardinality Features | False |
| 8 | High Cardinality Method | None |
| 9 | Transformed Train Set | (277, 40) |
| 10 | Transformed Test Set | (120, 40) |
| 11 | Shuffle Train-Test | True |
| 12 | Stratify Train-Test | False |

# #2 Pycaret - (2) 모델 생성 및 비교

**머신러닝 모델을 선언해서 사용하거나, 전처리한 데이터셋에 맞는 모델 비교**

- **model()**
  각각 머신러닝 기법(Classification, Regression 등)에 따라 구현된 모델들을 나열
- **compare_models()**
  setup된 데이터를 각각 머신러닝 모델에 적용 후 비교
- **create_model()**
  model()에 적힌 머신러닝 모델을 선택해서 생성

```
best_5_l = compare_models(sort='MAE', n_select=5)
```

| | Model | MAE | MSE | RMSE | R2 | RMSLE | MAPE | TT (Sec) |
|---|---|---|---|---|---|---|---|---|
| **rf** | Random Forest Regressor | 0.1985 | 0.0753 | 0.2706 | 0.2295 | 0.1391 | 0.2628 | 0.575 |
| **catboost** | CatBoost Regressor | 0.2004 | 0.0767 | 0.2736 | 0.2085 | 0.1404 | 0.2624 | 5.027 |
| **br** | Bayesian Ridge | 0.2052 | 0.0765 | 0.2719 | 0.2132 | 0.1399 | 0.2699 | 0.016 |
| **lightgbm** | Light Gradient Boosting Machine | 0.2078 | 0.0810 | 0.2809 | 0.1605 | 0.1431 | 0.2664 | 0.042 |
| **gbr** | Gradient Boosting Regressor | 0.2120 | 0.0845 | 0.2854 | 0.1368 | 0.1462 | 0.2755 | 0.117 |
| **et** | Extra Trees Regressor | 0.2120 | 0.0915 | 0.2987 | 0.0583 | 0.1501 | 0.2758 | 0.461 |
| **knn** | K Neighbors Regressor | 0.2126 | 0.0805 | 0.2812 | 0.1620 | 0.1450 | 0.2832 | 0.063 |
| **ridge** | Ridge Regression | 0.2150 | 0.0891 | 0.2925 | 0.0802 | 0.1476 | 0.2834 | 0.013 |
| **omp** | Orthogonal Matching Pursuit | 0.2152 | 0.0820 | 0.2824 | 0.1645 | 0.1442 | 0.2788 | 0.016 |
| **lr** | Linear Regression | 0.2190 | 0.0969 | 0.3031 | 0.0032 | 0.1497 | 0.2893 | 0.016 |

# #2 Pycaret - (3) 모델 최적화

(2)에서 compare_models나 create_model을 사용해서 각각의 모델들을 생성하거나 비교하였으니
그 결과를 바탕으로 성능이 좋은 모델들을 조합하여 실험해볼 수 있는 모듈을 제공

- tune_model()
  모델의 하이퍼파라미터를 최적화하는 모듈
  하이퍼파라미터 반복 횟수/최적화할 매트릭 선택
- ensemble_model()
  ensemble 기법을 구현한 모듈
- blend_models()
  Voting 알고리즘을 구현한 모듈
  compare_models()에서 성능이 잘 나온 모델들을 선택하는 파라미터를 적용(n_select)시켜서 사용
- stack_models()
  stacking ensemble 방법을 구현한 모듈
  compare_models()에서 성능이 잘 나온 모델들을 선택하는 파라미터를 적용(n_select)시켜서 사용

```
blended_l = blend_models(estimator_list= best_5_l, fold=5, optimize='MAE')
pred_holdout = predict_model(blended_l)
final_model_l = finalize_model(blended_l)
pred_esb_l = predict_model(final_model_l, test_data)
```

| | MAE | MSE | RMSE | R2 | RMSLE | MAPE |
|---|---|---|---|---|---|---|
| 0 | 0.1819 | 0.0901 | 0.3002 | 0.1841 | 0.1353 | 0.1908 |
| 1 | 0.2162 | 0.0808 | 0.2842 | 0.2599 | 0.1549 | 0.3462 |
| 2 | 0.1704 | 0.0505 | 0.2247 | 0.2862 | 0.1147 | 0.2010 |
| 3 | 0.2037 | 0.0712 | 0.2668 | 0.2749 | 0.1437 | 0.3007 |
| 4 | 0.2060 | 0.0705 | 0.2656 | 0.3145 | 0.1358 | 0.2479 |
| Mean | 0.1956 | 0.0726 | 0.2683 | 0.2639 | 0.1369 | 0.2573 |
| SD | 0.0169 | 0.0132 | 0.0252 | 0.0437 | 0.0132 | 0.0591 |

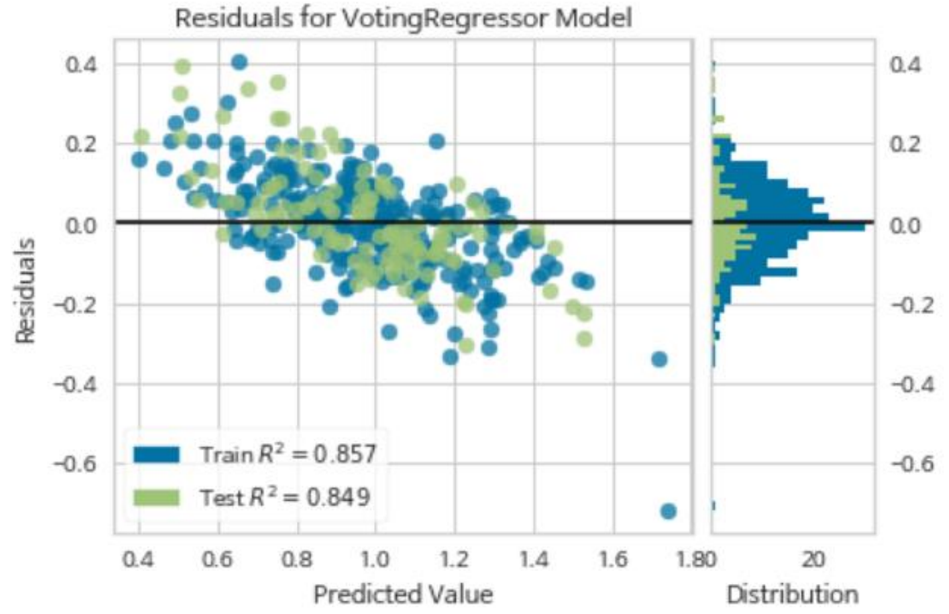
| | Model | MAE | MSE | RMSE | R2 | RMSLE | MAPE |
|---|---|---|---|---|---|---|---|
| 0 | Voting Regressor | 0.2075 | 0.0819 | 0.2862 | 0.2505 | 0.1575 | 0.3819 |

# #2 Pycaret - (4) 학습된 모델 분석

**학습한 이후에 데이터셋이 잘 학습되었는지 검증을 도와주는 모듈을 제공**

- plot_model()
  학습한 모델에 대한 각종 지표들을 시각화한 플롯을 그려주는 모듈
- interpret_model()
  모델이 예측한 결과에 대해서 각 파라미터들이 얼마나 영향을 줬는지 시각화
- evaluate_model()
  모델 분석 후 각 플롯을 볼 수 있도록 사용자 인터페이스를 제공
- get_leaderboard()
  setup 이후 훈련된 모든 모델을 출력
- dashboard()
  모델 분석에 대한 사용자 인터페이스를 제공
- eda()
  데이터셋 분석 결과를 제공

```
plot_model(final_model_l, plot='residuals')
```

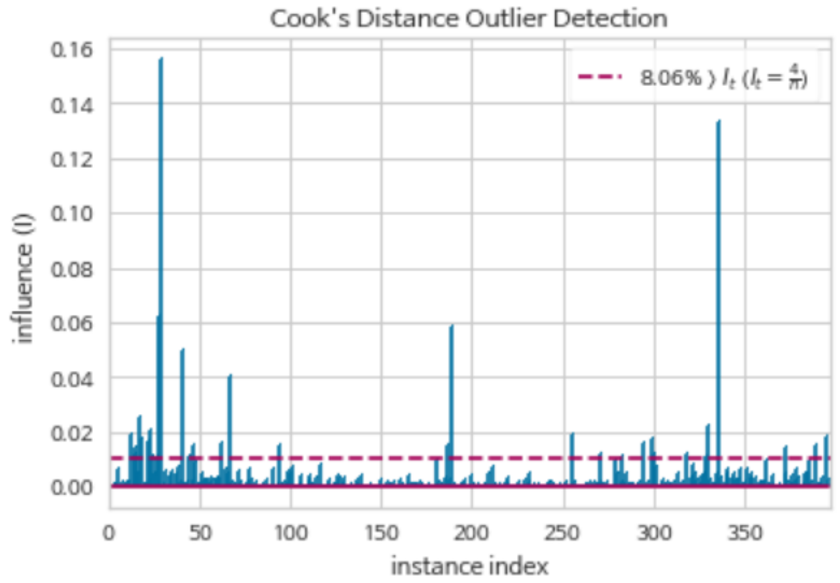


# #2 Pycaret - (4) 학습된 모델 분석

**학습한 이후에 데이터셋이 잘 학습되었는지 검증을 도와주는 모듈을 제공**

- plot_model()
  학습한 모델에 대한 각종 지표들을 시각화한 플롯을 그려주는 모듈
- interpret_model()
  모델이 예측한 결과에 대해서 각 파라미터들이 얼마나 영향을 줬는지 시각화
- evaluate_model()
  모델 분석 후 각 플롯을 볼 수 있도록 사용자 인터페이스를 제공
- get_leaderboard()
  setup 이후 훈련된 모든 모델을 출력
- dashboard()
  모델 분석에 대한 사용자 인터페이스를 제공
- eda()
  데이터셋 분석 결과를 제공

```
plot_model(final_model_l, plot='cooks') ## 심한 아웃 라이어들이 존재
```

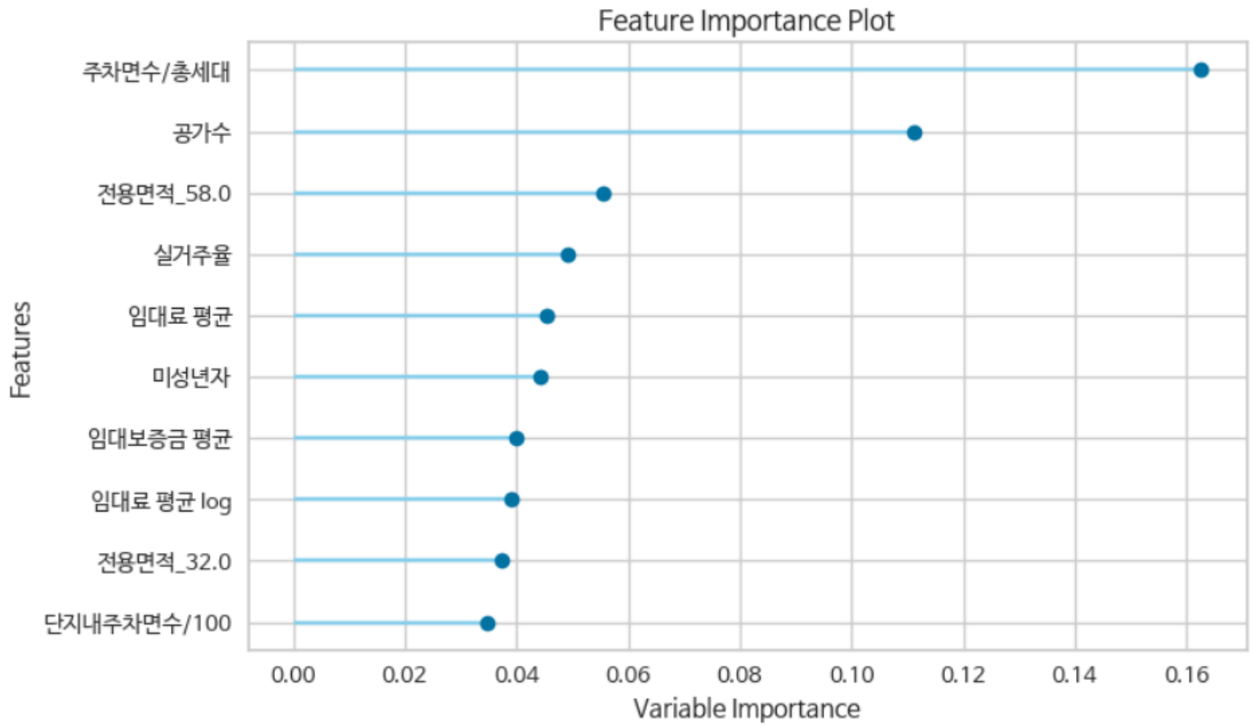


# #2 Pycaret

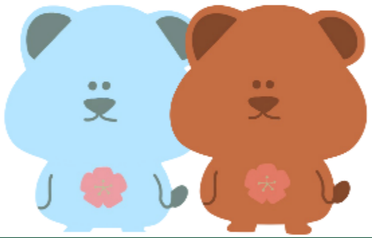
```
rf_model = create_model('rf')
```

| | MAE | MSE | RMSE | R2 | RMSLE | MAPE |
|---|---|---|---|---|---|---|
| 0 | 0.1888 | 0.0577 | 0.2402 | 0.3248 | 0.1269 | 0.2528 |
| 1 | 0.1871 | 0.1375 | 0.3709 | -0.0827 | 0.1568 | 0.1522 |
| 2 | 0.1994 | 0.0634 | 0.2518 | 0.3384 | 0.1316 | 0.2618 |
| 3 | 0.2322 | 0.0933 | 0.3054 | 0.2266 | 0.1710 | 0.4205 |
| 4 | 0.1543 | 0.0371 | 0.1926 | 0.4983 | 0.1051 | 0.2118 |
| 5 | 0.2024 | 0.0745 | 0.2730 | -0.1471 | 0.1362 | 0.2142 |
| 6 | 0.1910 | 0.0603 | 0.2455 | 0.4116 | 0.1347 | 0.2923 |
| 7 | 0.2013 | 0.0756 | 0.2749 | 0.1881 | 0.1479 | 0.3013 |
| 8 | 0.2033 | 0.0605 | 0.2459 | 0.3254 | 0.1257 | 0.2310 |
| 9 | 0.2254 | 0.0936 | 0.3059 | 0.2113 | 0.1550 | 0.2904 |
| Mean | 0.1985 | 0.0753 | 0.2706 | 0.2295 | 0.1391 | 0.2628 |
| SD | 0.0204 | 0.0262 | 0.0460 | 0.1941 | 0.0179 | 0.0680 |

```
plot_model(rf_model, plot='feature')
```

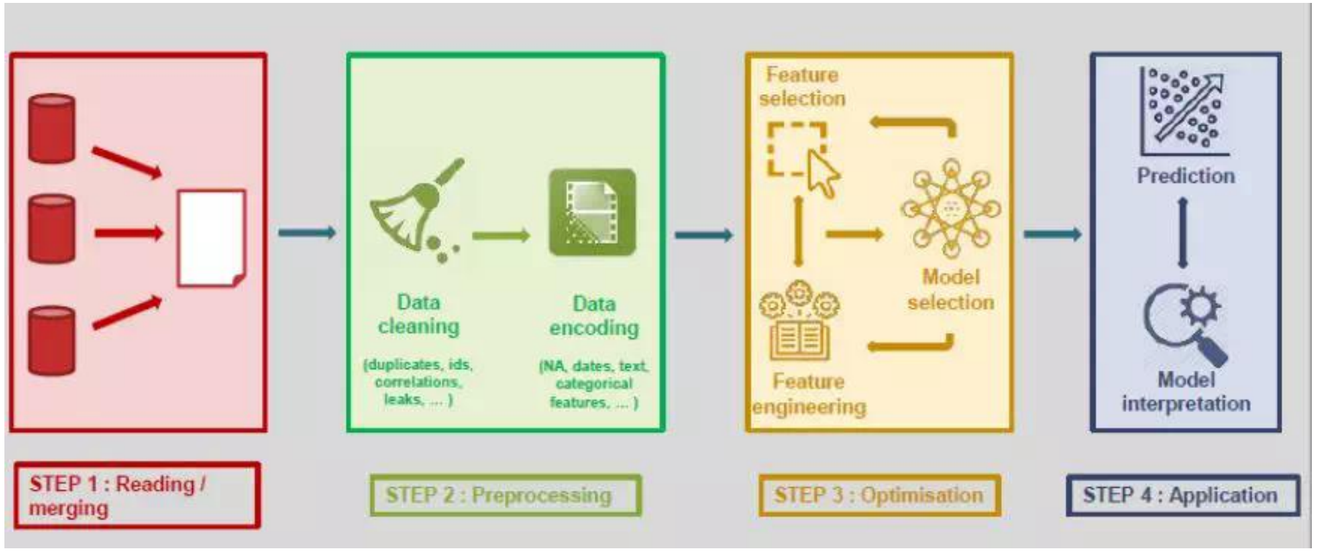


# Other AutoML Frameworks



# #3-0 AutoML frameworks

일반적인 ML model steps
1) Data reading
2) pre-processing
3) Optimization
4) Result prediction




Data collection, prediction이 자동화하기 힘든 부분 – AutoML: prediction을 위한 최적화 및 준비 제공

## AutoML frameworks categories

1) 자동으로 Parameter tuning해주는 AutoML
2) AutoML for non-deep learning: data 전처리, 자동 feature 분석, 자동 feature detection, 자동 feature/model selection에 사용
3) AutoML for deep learning/neural networks: NAS를 포함하여 최적의 architectrue까지 찾아줌

# #3-1 MLBox

#1 빠른 reading과 분산 데이터 전처리 및 cleaning 및 formatting

#2 매우 robust한 feature selection, leak detection

#3 높은 dimensional space에서 hyper-parameter 최적화 정확도 높음

#4 classification과 regression에 SOTA predicted models을 사용할 수 있음

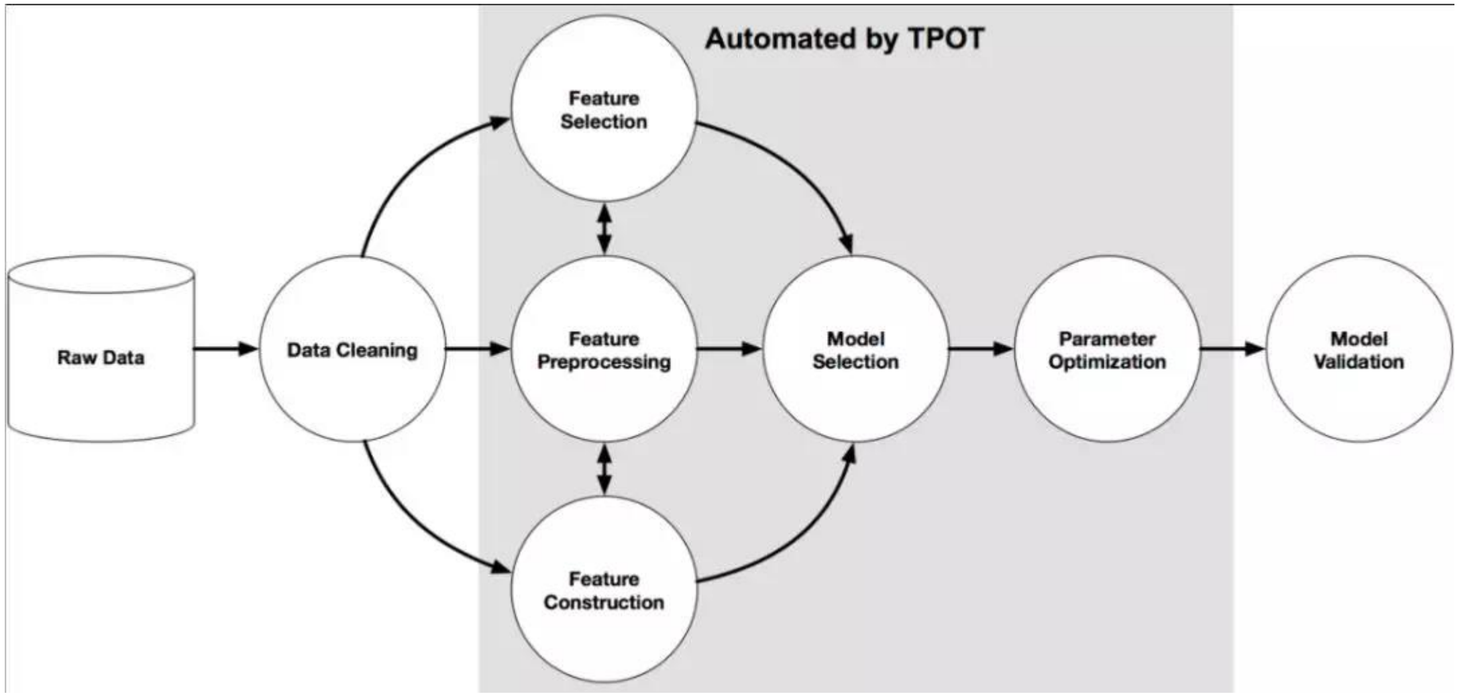

**MLBox,**
**Machine Learning Box**

# #3-2 TPOT

#1 유전 알고리즘을 사용하여 ML 파이프라인을 최적화

#2 가능한 수많은 파이프라인들 중에서 데이터에 가장 맞는 것 찾아줌으로써 ML 자동화

#3 scikit-learn framework로 만들어짐 -> scikit-learn에 익숙한 사람에게 익숙



# #3-3 Auto-sklearn

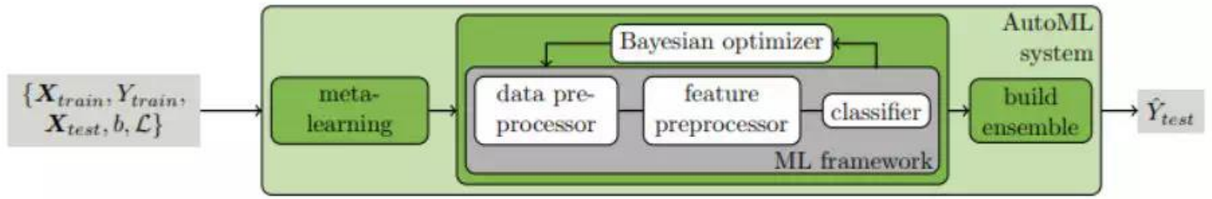
#1 알고리즘 선택과 hyperparameter tuning을 자동으로 해줌

#2 Meta learning: Bayes를 사용하여 optimizer를 초기화 및 설정값을 자동으로 평가하는데 사용

#3 Bayes: channel을 최적화하기 위해 사용

#4 sklearn estimators를 classificatio과 regression 문제에 사용

#5 작은 데이터셋에도 잘 동작, 큰 데이터셋에는 좋은 성능을 잘 내지 못함




Auto-sklearn pipeline

# #3-4 AutoKeras

#1 keras를 기반의 AutoML system

#2 자동으로 최적의 hyper-parameter, architecture 찾아줌

#3 NAS(Neural Architecture Search)를 사용하여 ML를 단순화하는 것이 AutoKeras내의 trend

# #3-5 H2O AutoML

#1 scalable 분산 ML에 사용되는 in-memory platform

#2 R, Python 모두 지원

#3 GLM(Generalized Linear Models), Gradient Boosting Machines, Random Forests, DNN, Stacked Ensembles 등과 같은 많은 유명한 알고리즘 지원

#4 feature engineering, model validation, model adjustment, model selection, model deploymen와 같은 복잡한 데이터 사이언스와 ML 업무에도 사용가능

# #3-6 AutoGluon

#1 다층 Stacking과 repeated k-fold bagging에 의한 강력한 앙상블 학습

#2 자동으로 견고한 predictive performance를 빨리 낼 수 있게 함


#3 높은 성능의 ML과 텍스트, 이미지, tabular의 딥러닝 모델을 몇 줄의 코드로 학습 및 사용 가능

#4 해당 논문에 따르면 H20, TPOT, auto-sklearn보다 더 좋은 성능과 빠르게 동작한다고 함

| Framework | Wins | Losses | Failures | Champion | Avg. Rank | Avg. Rescaled Loss | Avg. Time (min) |
|---|---|---|---|---|---|---|---|
| AutoGluon | - | - | **1** | **23** | **1.8438** | **0.1385** | 201 |
| H2O AutoML | 4 | 26 | 8 | 2 | 3.1250 | 0.2447 | 220 |
| TPOT | 6 | 27 | 5 | 5 | 3.3750 | 0.2034 | 235 |
| GCP-Tables | 5 | 20 | 14 | 4 | 3.7500 | 0.3336 | **195** |
| auto-sklearn | 6 | 27 | 6 | 3 | 3.8125 | 0.3197 | 240 |
| Auto-WEKA | 4 | 28 | 6 | 1 | 5.0938 | 0.8001 | 244 |

# #3-7 MLJAR

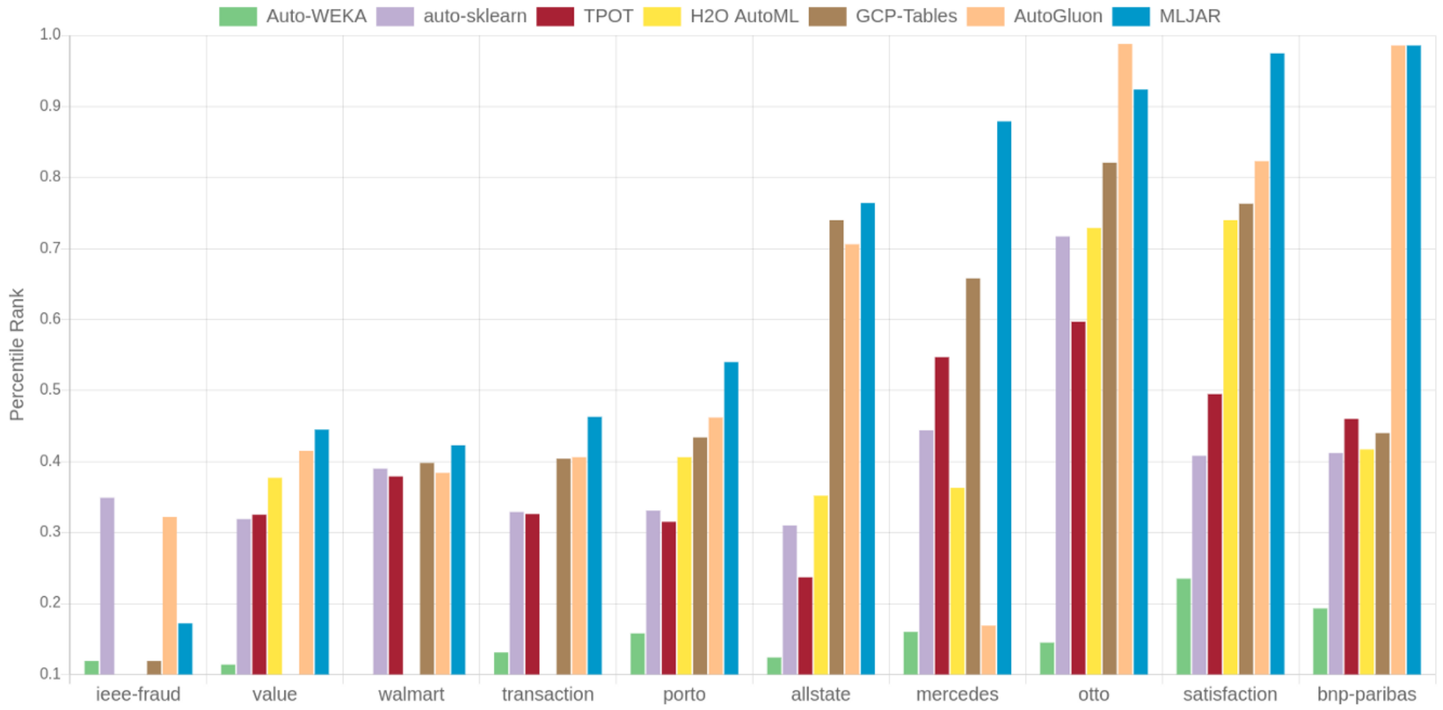
#1 AutoML 학습의 모든 단계의 세부사항을 확인할 수 있음

#2 적합한 알고리즘과 hyper-parameter tuning을 찾아줌

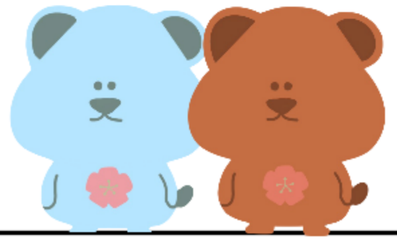
#3 분석내용과 ML모델을 저장, 재실행, 로드할 수 있음

#4 tabular data에 대해 가장 좋은 성능(2021년 기준)

| Dataset | Auto-WEKA | auto-sklearn | TPOT | H2O AutoML | GCP-Tables | AutoGluon | MLJAR |
|---|---|---|---|---|---|---|---|
| ieee-fraud | 0.119 | 0.349 | | | 0.119 | 0.322 | 0.172 |
| value | 0.114 | 0.319 | 0.325 | 0.377 | | 0.415 | 0.445 |
| walmart | | 0.390 | 0.379 | | 0.398 | 0.384 | 0.423 |
| transaction | 0.131 | 0.329 | 0.326 | | 0.404 | 0.406 | 0.463 |
| porto | 0.158 | 0.331 | 0.315 | 0.406 | 0.434 | 0.462 | 0.540 |
| allstate | 0.124 | 0.310 | 0.237 | 0.352 | 0.740 | 0.706 | 0.764 |
| mercedes | 0.160 | 0.444 | 0.547 | 0.363 | 0.658 | 0.169 | 0.879 |
| otto | 0.145 | 0.717 | 0.597 | 0.729 | 0.821 | 0.988 | 0.924 |
| satisfaction | 0.235 | 0.408 | 0.495 | 0.740 | 0.763 | 0.823 | 0.975 |
| bnp-paribas | 0.193 | 0.412 | 0.460 | 0.417 | 0.440 | 0.986 | 0.986 |



# THANK YOU



EWHA EURON