


# 자율주행 센서의 안테나 성능 예측 AI 경진대회

팀 주혁이

전주혁 강동인 민재식 최새한 권수지

# Contents

- 
-  **01** Intro
  - 02** Feature Engineering
  - 03** Validation
  - 04** Modeling
  - 05** Outro
-

## 대회 배경 및 목표

레이더 제조 공정에서 AI 기술을 활용하여  
공정 데이터와 제품 성능간 상관 관계 파악,  
이를 통해 제품의 불량률 예측, 분석하여  
수율 극대화하고자 함



공정 데이터를 활용하여  
제품 성능(Radar 센서의 안테나)을  
예측하는 AI 모델 개발

## 제공데이터

### Train.csv :

학습 데이터셋(39607개),  
ID, X Feature(56개), Y Feature(14개)

### Test.csv :

테스트 데이터셋(39608개),  
ID, X Feature(56개)

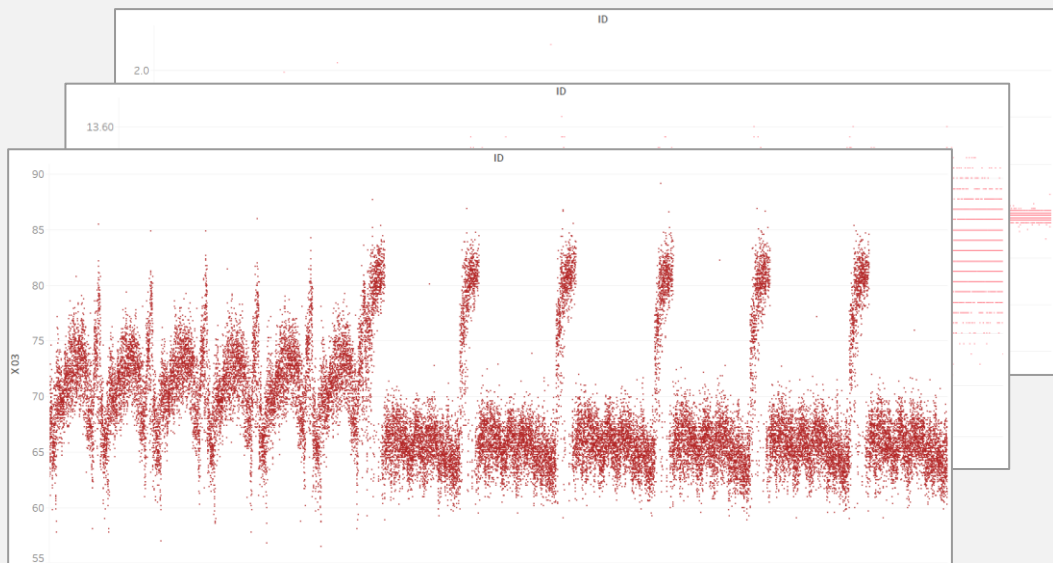
### /meta :

비식별화된 X,Y feature에 대한  
세부 설명 자료

## 평가 항목

- 모델 성능
- Feature 상관관계분석
- Validation set 구축 전략
- 모델 적용 가능성

## 시계열적 접근방법 적용 배경



< X03 feature 외 데이터 산점도 >

접근

실제 공정은 연속적 프로세스임에 집중

아이디어

각 공정 프로세스는 대규모의 시계열 과정일 것임

분석

EDA 결과 2개의 주기 존재, 각 주기는 6개 작은 주기로 구성

검증

주기성을 띤다는 것은 연속적인 데이터들임을 의미

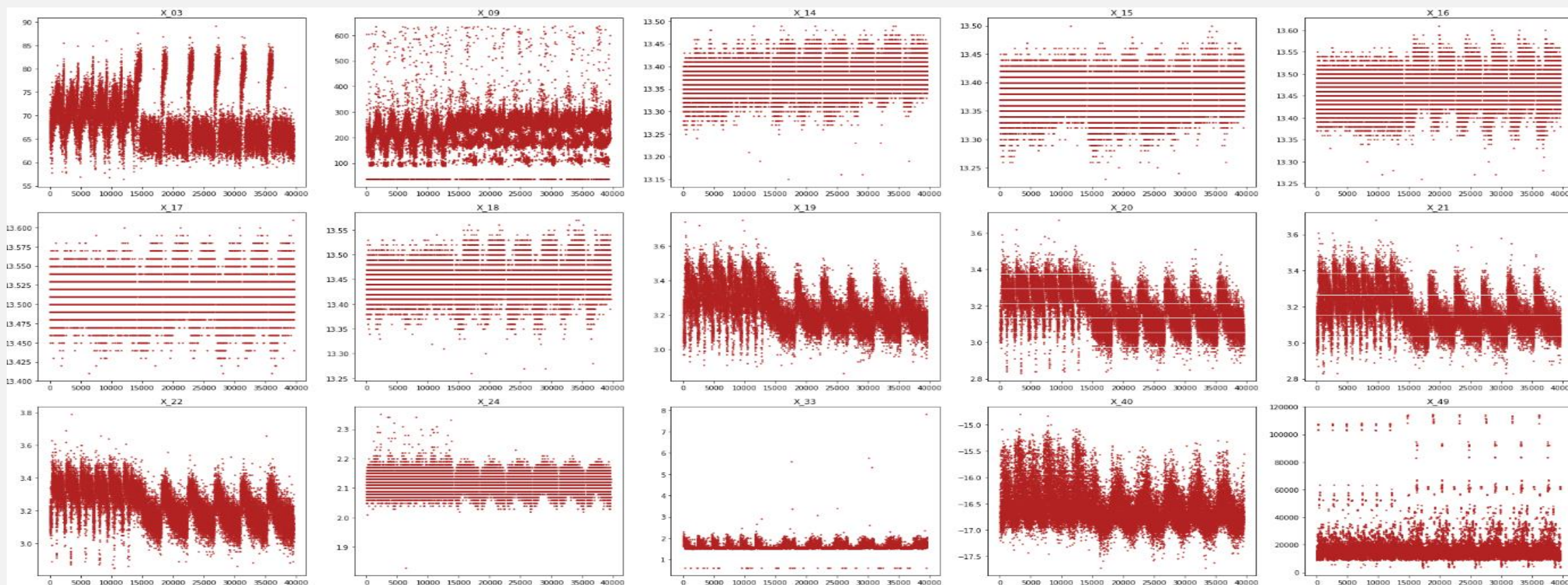
적용

시계열 + 주기성 활용



## [참고]

## 대표적인 15개 X-feature의 산점도



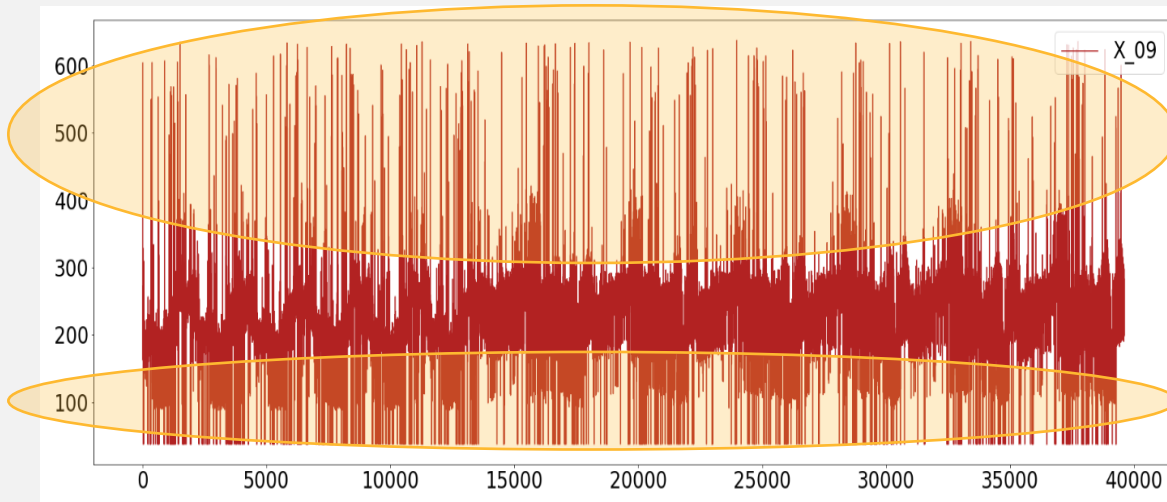


# Contents

- 
- 01 Intro
  - ▶ 02 Feature Engineering
  - 03 Validation
  - 04 Modeling
  - 05 Outro
-

## Problem

< X\_09 Index 기준 Lineplot >



값이 튀는 값이 존재

## Hypothesis & Insight

- 시계열 데이터 특징 활용
- 튀는 값은 공정 과정 중 발생한 오차로 생각
- 측정값의 분산을 낮춘다면 더욱 정확한 예측 가능할 것임



측정 데이터의 분산을 줄이기 위해  
이동 평균값, 이동 중앙값 이용

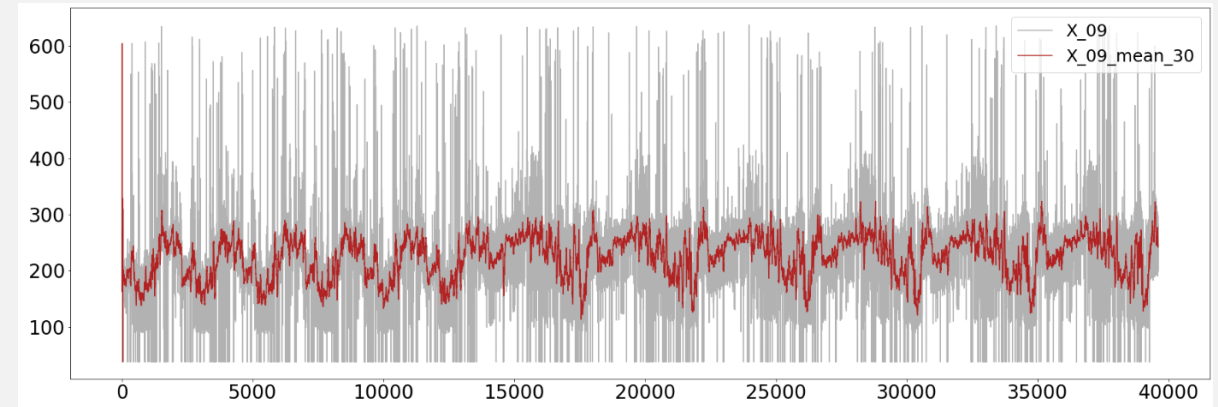
Moving\_Avg&median KalmanFilter Etc.

## 이동 평균값

특정 크기의 부분 집합을 연속적으로 이동하며 산출한 평균

- ▶ 인덱스 순서에 따른 추세 반영

적용결과

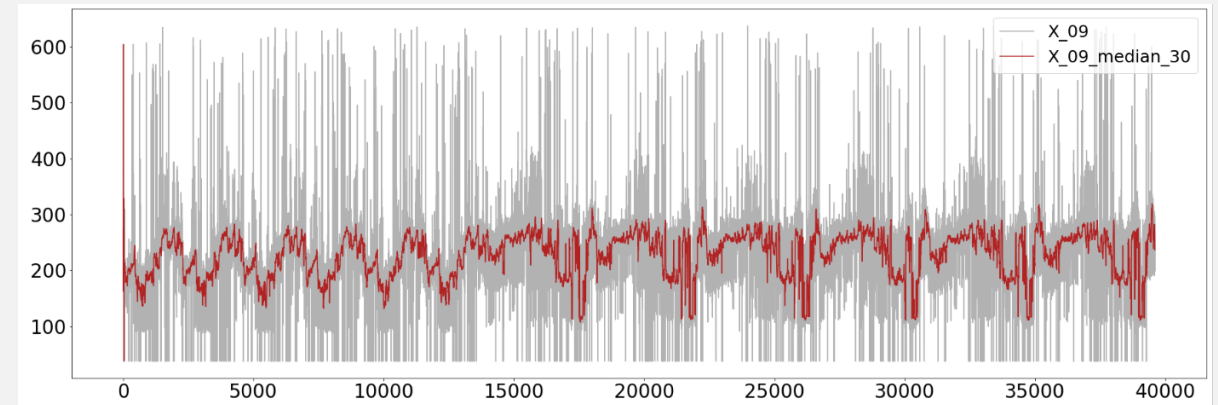


## 이동 중앙값

특정 크기의 부분 집합을 연속적으로 이동하며 산출한 중앙값

- ▶ 튀는 값 보정

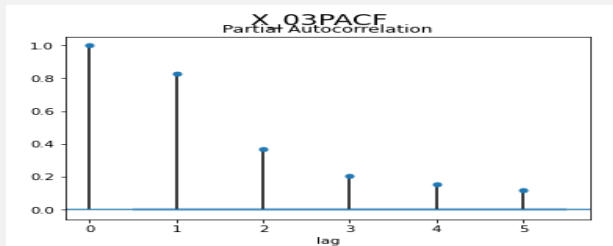
적용결과





**가설** 측정값은 공정 과정 중 공구의 마모 혹은 공정 세팅 등으로 인해 이전 값에 영향을 받을 것임

### 자기상관성



✓ PACF<sup>1)</sup> 결과 이전값과의 상관성 존재

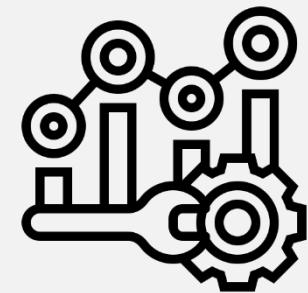
+

### 시계열 데이터 특성 활용



+

### 측정 오차 보정 필요



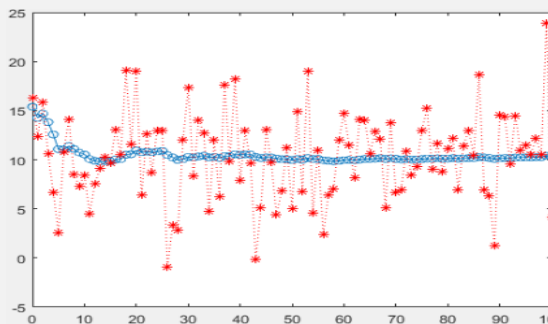
위 3가지 요소를 반영할 수 있는 변수 필요

*Kalman Filter 활용*

1) PACF : 자기상관성(과거 데이터와의 상관관계), 1에 가까울수록 상관관계 높음

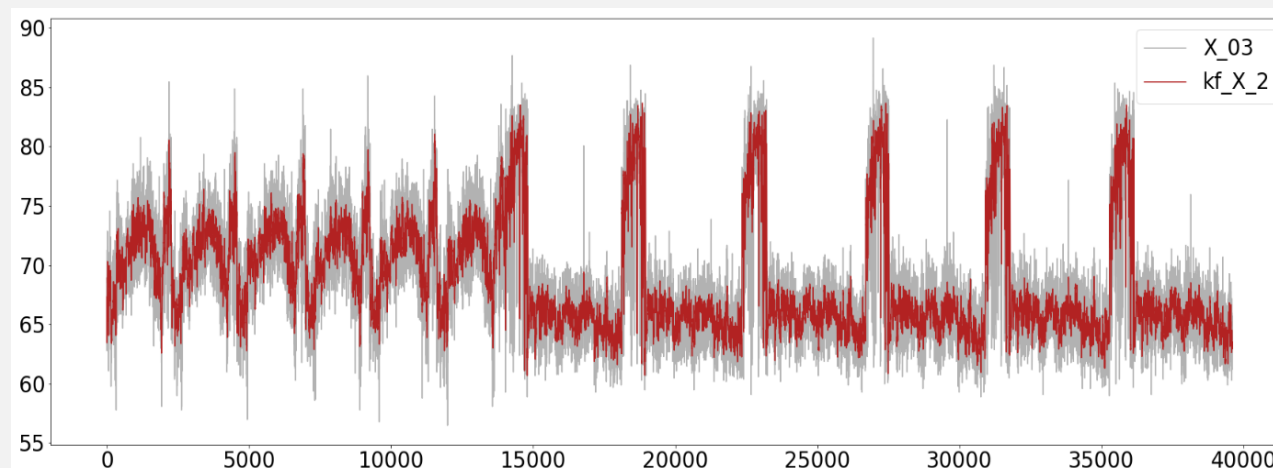
## 칼만 필터(Kalman Filter)

- ✓ 시계열 데이터에서 이전 데이터의 값을 이용하여 측정 과정에서 생긴 오차를 예측
- ✓ 오차를 줄여 더 정확한 데이터로 업데이트하는 과정을 계속해서 반복하여 측정 값을 보정해주는 알고리즘
- ✓ Ex) 노이즈캔슬링, 레이더 추적 등에 활용



< 칼만 필터 효과 예시<sup>1)</sup> >

## 칼만 필터 적용 결과(X\_03 feature)



칼만 필터를 적용한 결과 더욱 안정적인 데이터 분포를 가짐

## 기타 추가된 Feature들

- 칼만 필터
- 이동 평균
- 이동 중앙값



### 로그 변환

- log\_x\_01
- log\_x\_02
- log\_x\_03
- ⋮
- log\_x\_56

### 분산

- 안테나 패드 간의 위치 차이
- 스크류 삽입 깊이 차이
- PCB 체결 시 단계별 누름량 1,2 합
- PCB 체결 시 단계별 누름량 3,4 합
- 방열재료 총면적
- 레이돔치수와 안테나 위치 사이 차이


### 합/차

- 안테나 패드 위치들의 분산
- 스크류 삽입 깊이들의 분산
- 커넥터 핀 치수들의 분산

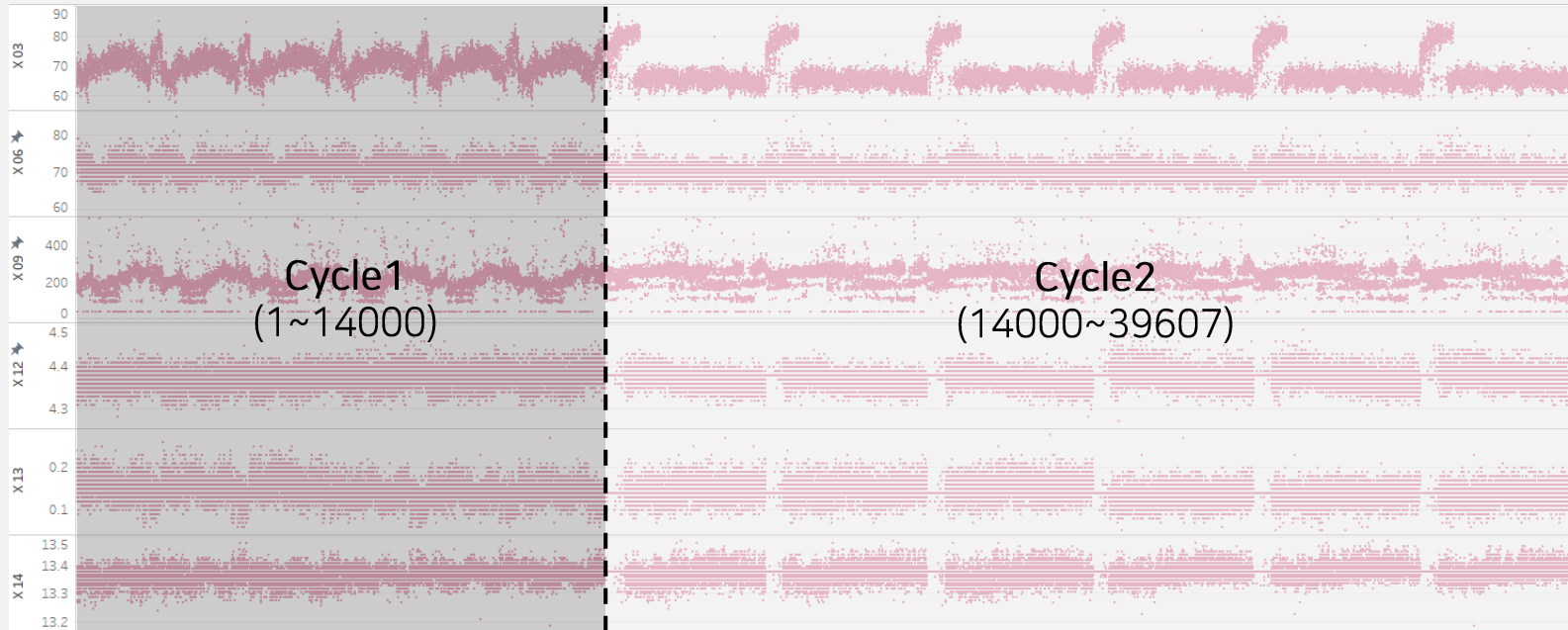
총 322개의 Feature 활용



# Contents

- 
- 01 Intro
  - 02 Feature Engineering
  -  03 Validation
  - 04 Modeling
  - 05 Outro
-

## 데이터 특성에 기반한 Validation set 구축 전략



<Index 기준 X03, X06, X09, X12, X13, X14 산점도>

대부분의 X feature들과 일부 Y feature의  
주기성을 확인할 수 있으며,  
주기는 크기가 다른 2개의 사이클로 이루어짐



이를 반영한,  
Validation set 구축 전략 필요

## 계층별 K-fold 교차 검증 활용

### (회귀) 일반적 방법론 - KFOLD 교차 검증

CV1	Test data	Train data	Train data	Train data	Train data
CV2	Train data	Test data	Train data	Train data	Train data
CV3	Train data	Train data	Test data	Train data	Train data
CV4	Train data	Train data	Train data	Test data	Train data
CV5	Train data	Train data	Train data	Train data	Test data

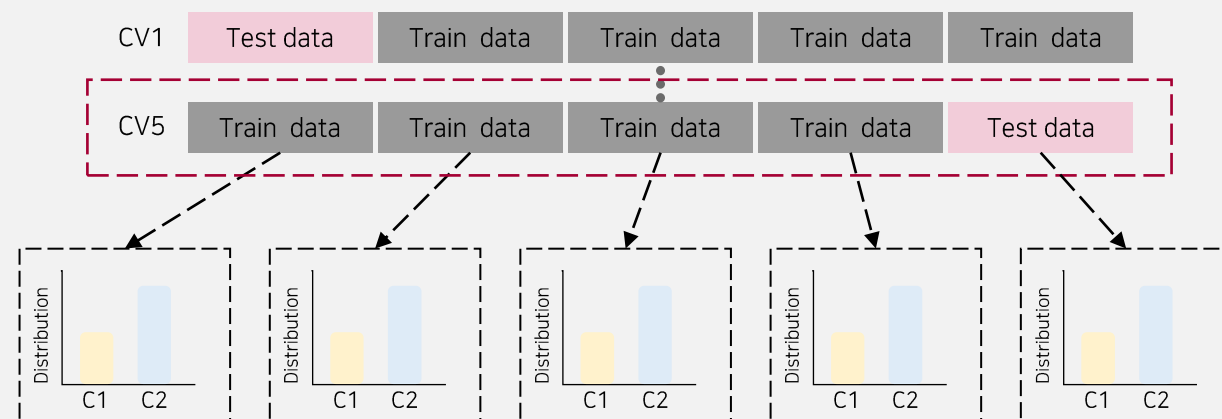
#### 장점

- 특정 데이터셋에 과적합 방지
- 일반화된 모델 생성
- 과소적합 방지(데이터셋 규모가 작을 경우)

#### 단점

- 데이터가 클래스가 불균형한 경우 학습 데이터가 고루 분할되지 못함

### 활용 방법론 - 계층별 KFOLD 교차 검증



주기가 다른 2개의 사이클을 지닌 데이터 특성을 반영하기 위해  
원본 데이터의 분포를 반영하는 계층별 KFOLD 교차 검증 활용



## [참고]

## 계층별 K-fold 교차 검증 활용 코드 적용 예시

```
1 train_y['Class'] = [0 if i<14000 else 1 for i in range(len(train_x))]  
  
kf = StratifiedKFold(n_splits=CFG.fold_num, shuffle=True, random_state=CFG.seed)  
  
2 for f_idx, (train_idx, val_idx) in enumerate(kf.split(train_x, train_y['Class'])):  
    train_input, train_target = train_x.iloc[train_idx, :].copy(), train_y.iloc[train_idx, :].copy()  
    val_input, val_target = train_x.iloc[val_idx, :].copy(), train_y.iloc[val_idx, :].copy()
```

1

약 14000번째에서 데이터의 사이클이 구분됨




Train\_y값을 14000번을 기준으로 Class 분리

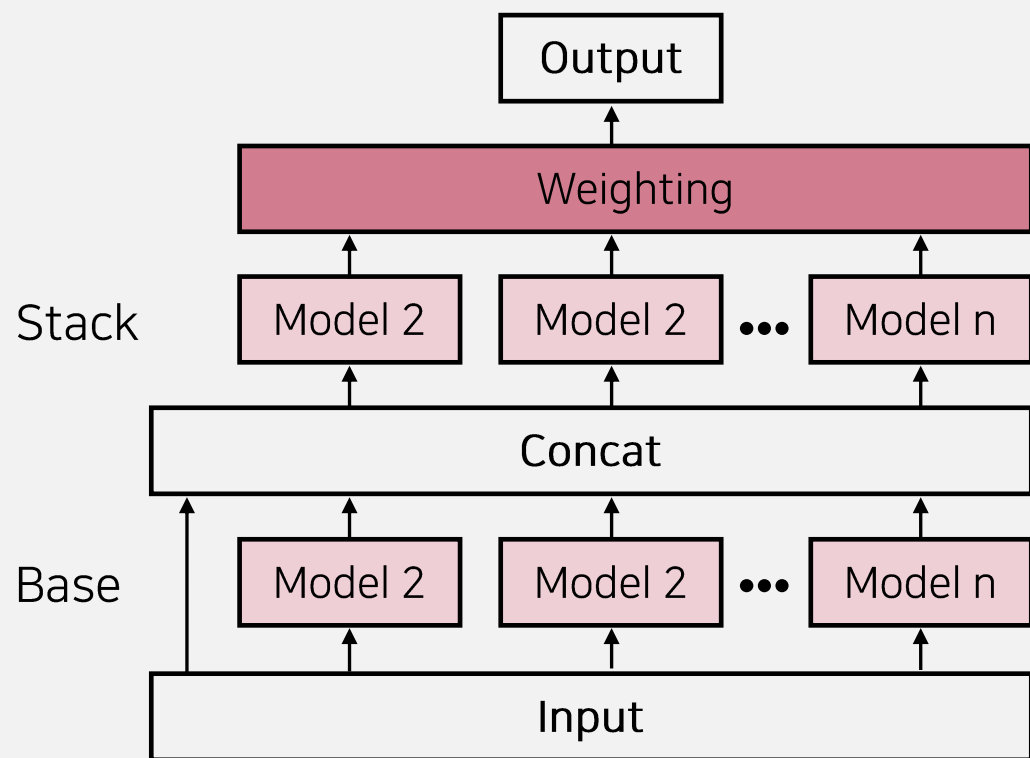
2

Class 값을 기준으로 계층별 K-Fold 교차 검증 실시

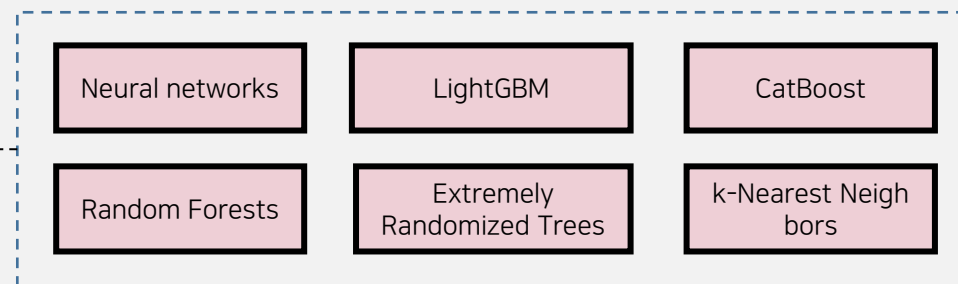
# Contents

- 
- 01 Intro
  - 02 Feature Engineering
  - 03 Validation
  -  04 Modeling
  - 05 Outro
-

## AutoGluon-Tabular



- AutoML 프레임워크
- 보편적인 AutoML이 주로 사용하는 CASH(Combined Algorithm Selection and Hyper-parameter optimization)이 아닌, **ensembling**과 **stacking** 활용
- 각 모델 학습시 Repeated k-fold ensemble bagging 사용  
-> 과학습 방지



## 1 학습시간

GPU - NVIDIA GeForce RTX 3070

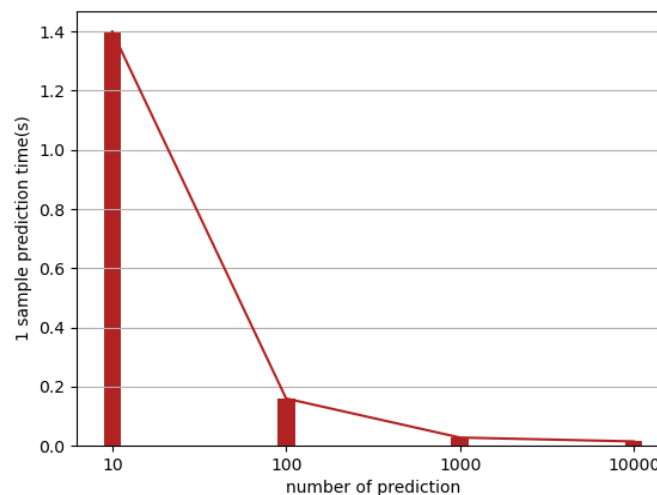
CPU - AMD Ryzen 9 5950X 16-Core Processor

GPU 학습 시 약 2시간 40분 소요

CPU 학습 시 약 4시간 20분 소요

“공정적용을 위한 빠른 학습이가능”

## 2 추론속도



10개 예측시 1개 처리당 1.4s 소요

100개 예측시 1개 처리당 0.16s 소요

1000개 예측시 1개 처리당 0.028s 소요

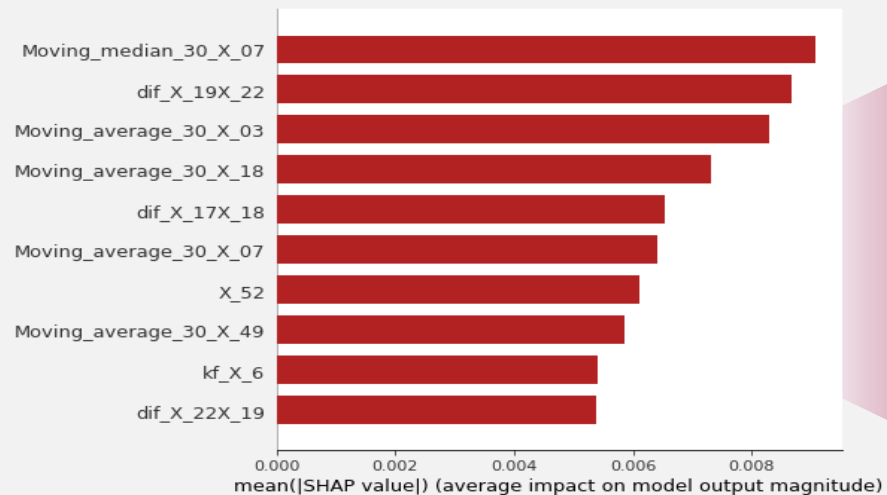
10000개 예측시 1개 처리당 0.015s 소요

“예측의 갯수가 늘어날수록 처리효율 증가”

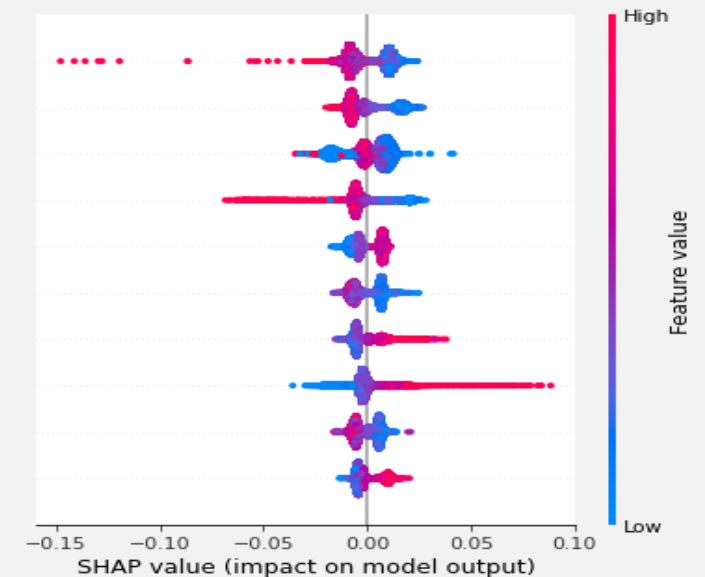
## Shapley Value

- 전체 모델 관점 : 변수들이 예측모델에 얼마나 기여하는가
- 개별 관측치 관점 : 개별 관측치에 따라 예측 결과가 어떻게 달라지는가

<Y\_01 예측 모델에 대한 shap value>




Moving\_median\_30\_X\_07  
dif\_X\_19X\_22  
Moving\_average\_30\_X\_03  
Moving\_average\_30\_X\_18  
dif\_X\_17X\_18  
Moving\_average\_30\_X\_07  
X\_52  
Moving\_average\_30\_X\_49  
kf\_X\_6  
dif\_X\_22X\_19



- ✓ 상단에 위치한 변수일수록 예측모형에 큰 기여를 함  
Moving\_median\_30\_X\_07, dif\_X\_18X\_22 순서
- ✓ 높은 X\_07값의 이동평균값을 가질 수록 Y\_01값이 낮아지는 경향을 보임



# Contents

- 
- 01 Intro
  - 02 Feature Engineering
  - 03 Validation
  - 04 Modeling
  -  05 Outro
-



# 감사합니다

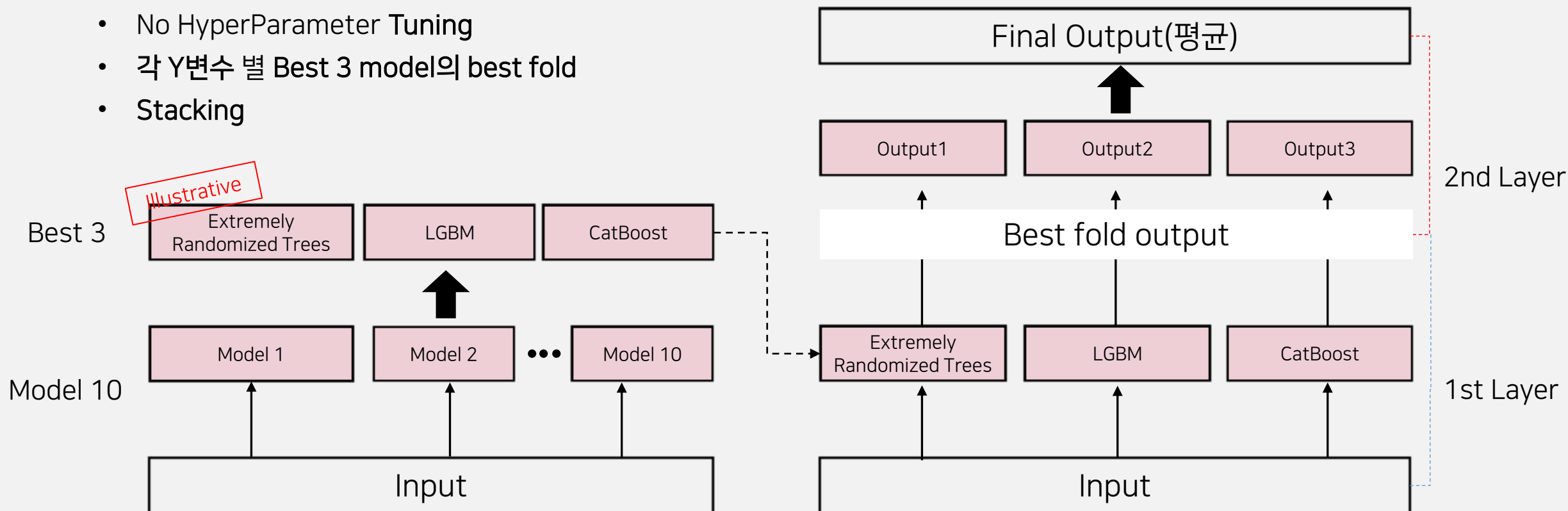
팀 주혁이  
전주혁 강동인 민재식 최새한 권수지

# Appendix

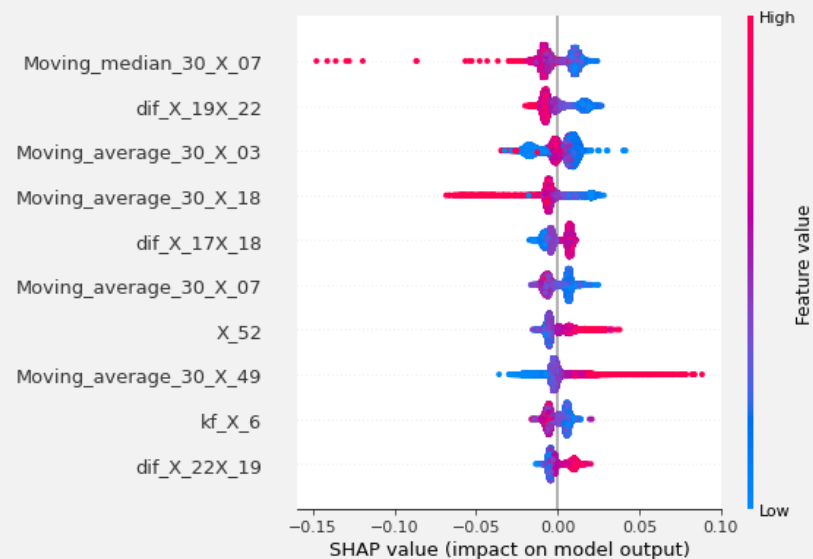
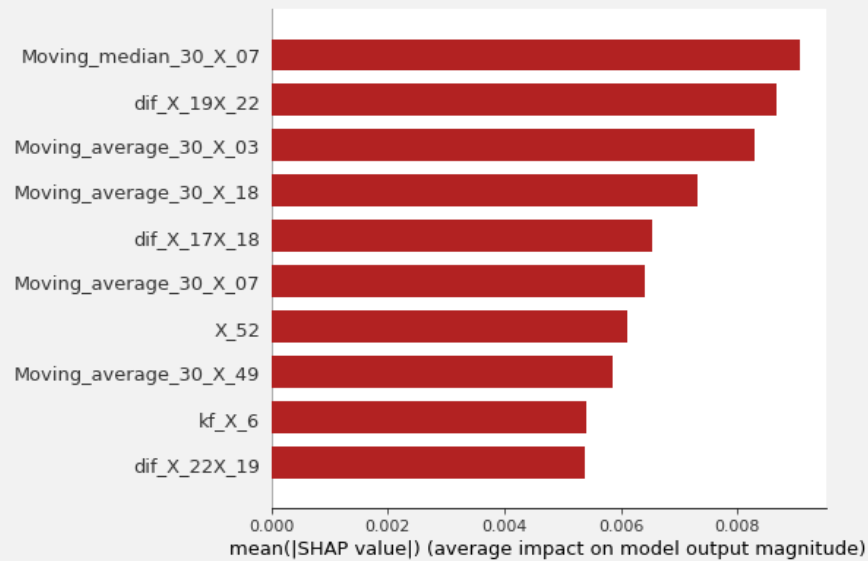
## [참고\_추가적인 모델]

## Ensemble Model(private score : 1.929 2등)

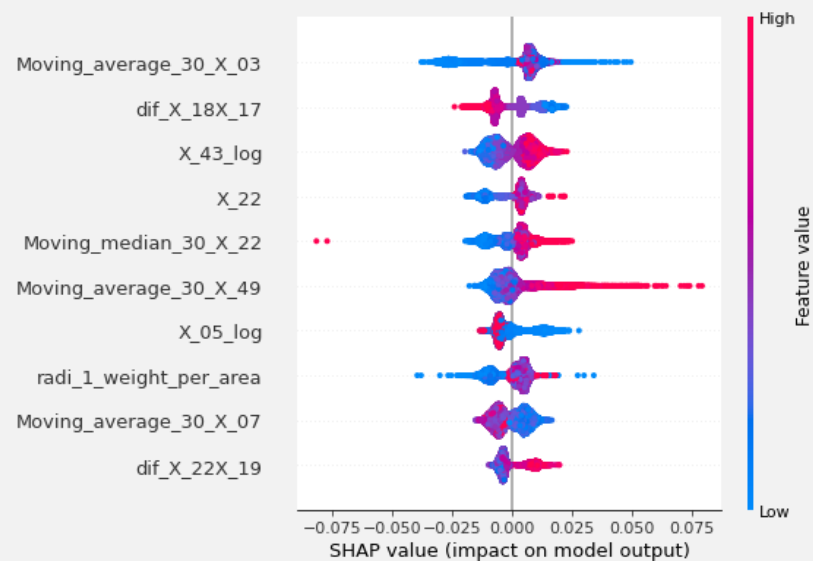
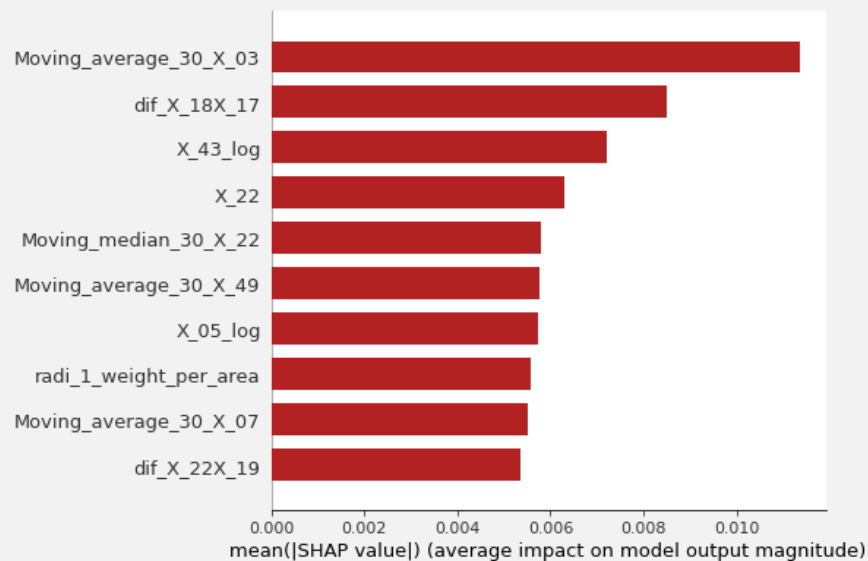
- No HyperParameter Tuning
- 각 Y변수 별 Best 3 model의 best fold
- Stacking



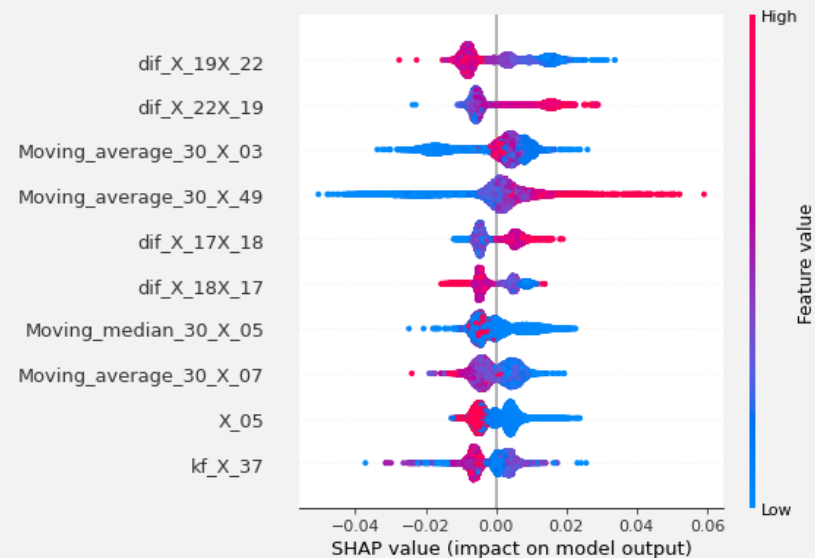
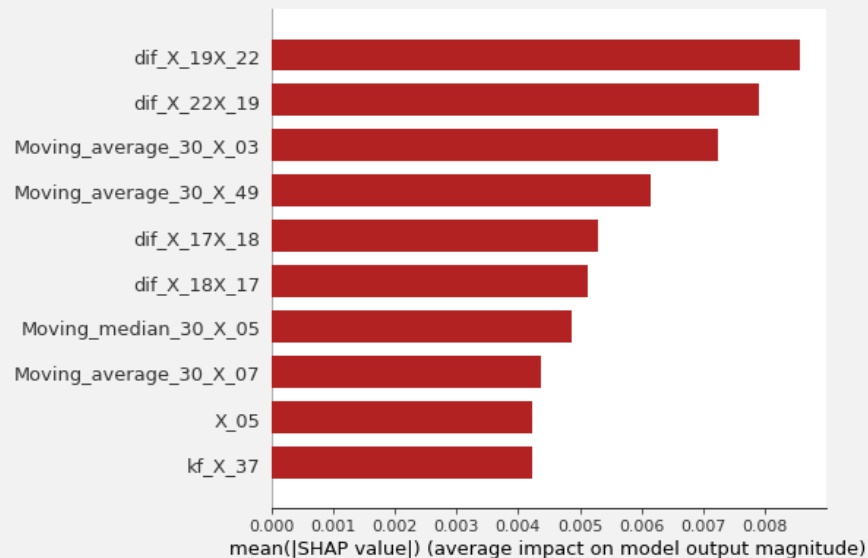
## Y\_01 Model



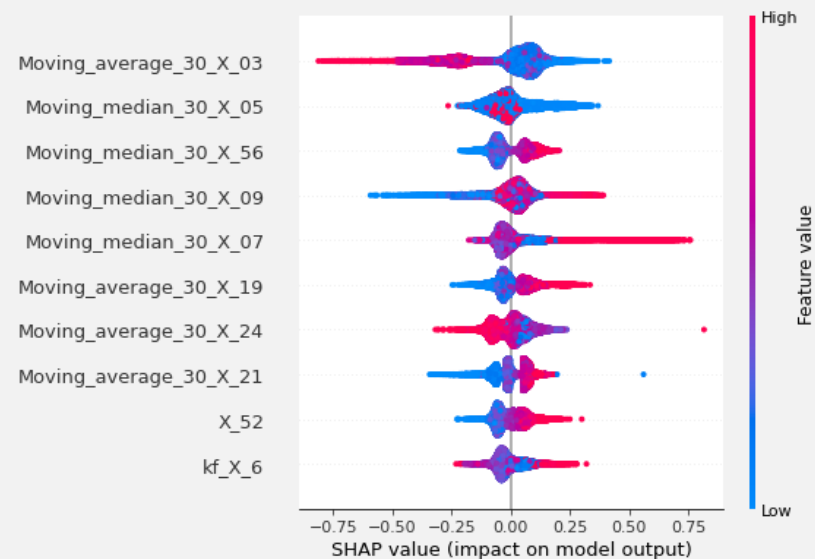
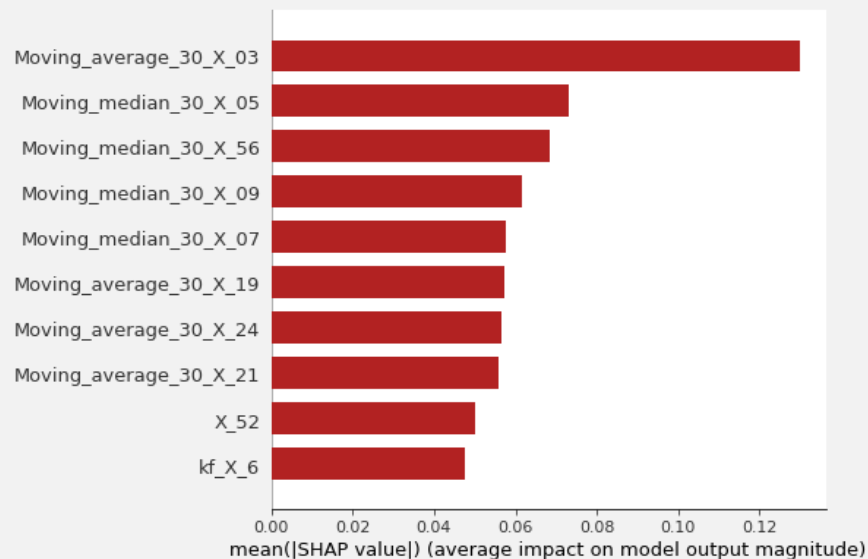
## Y\_02 Model



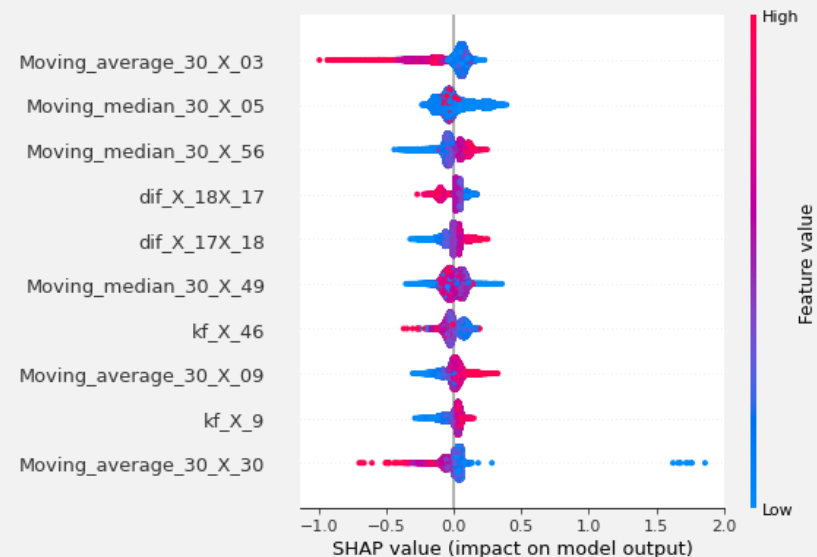
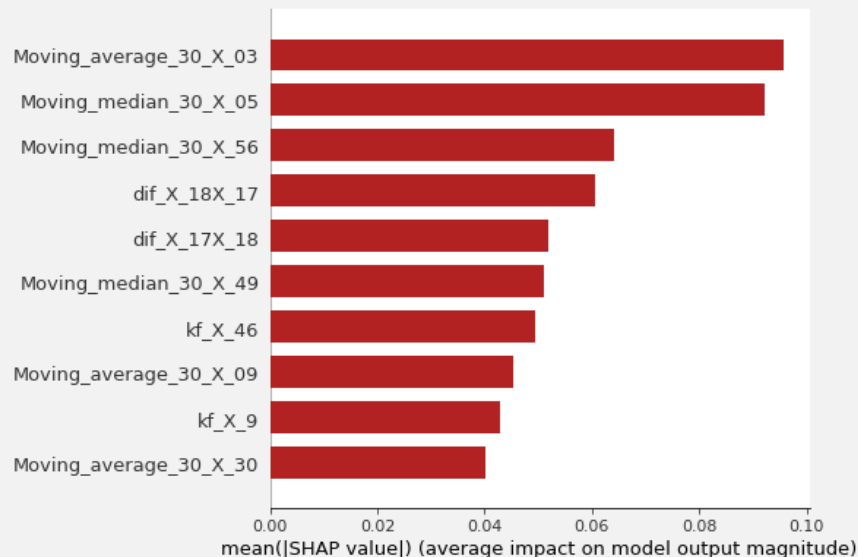
## Y\_03 Model



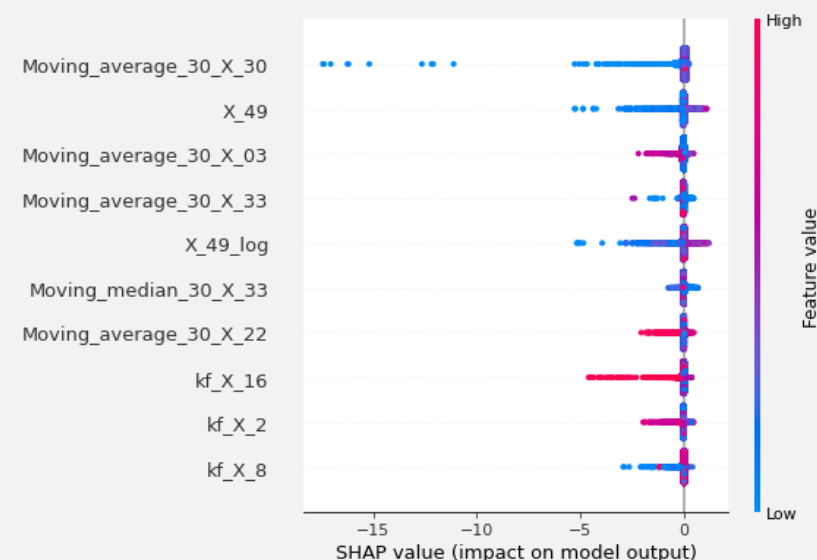
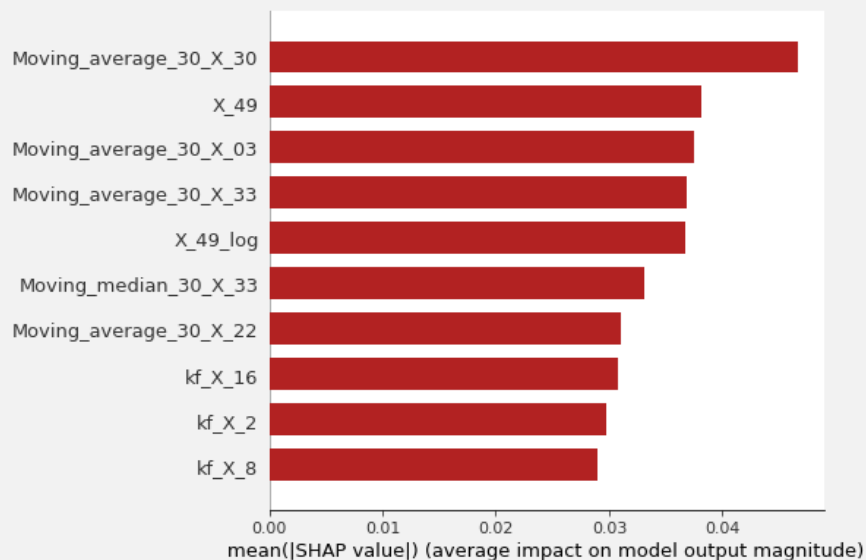
## Y\_04 Model



## Y\_05 Model

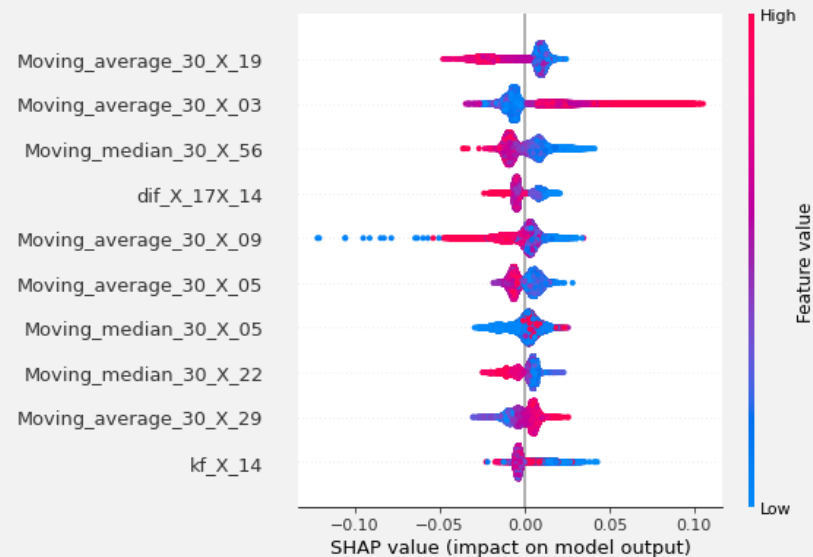
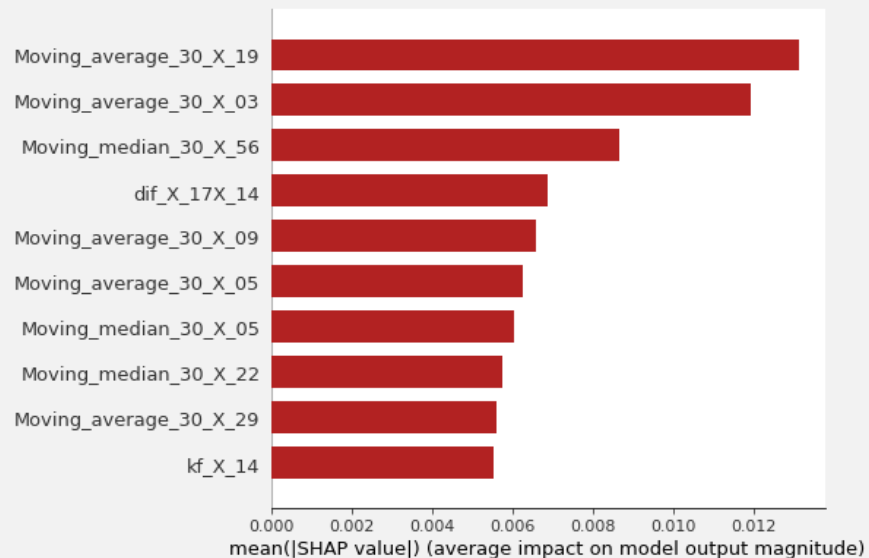


## Y\_06 Model

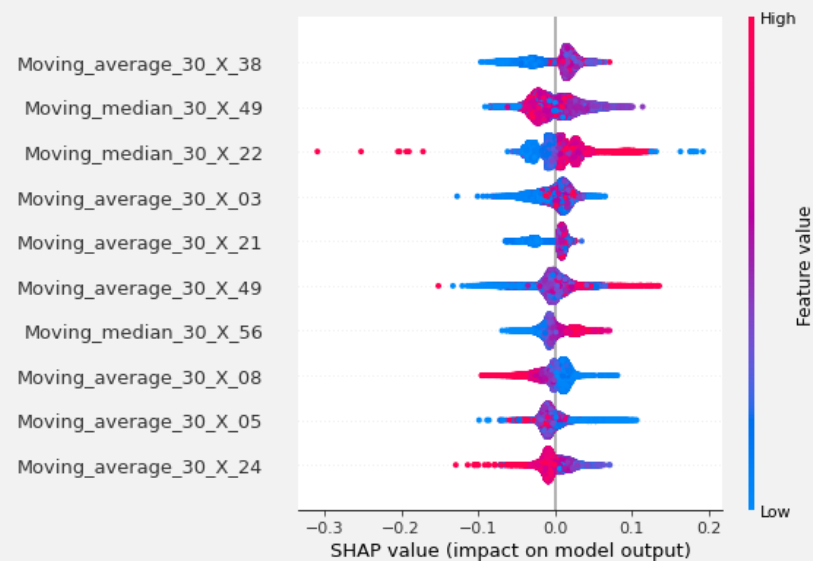
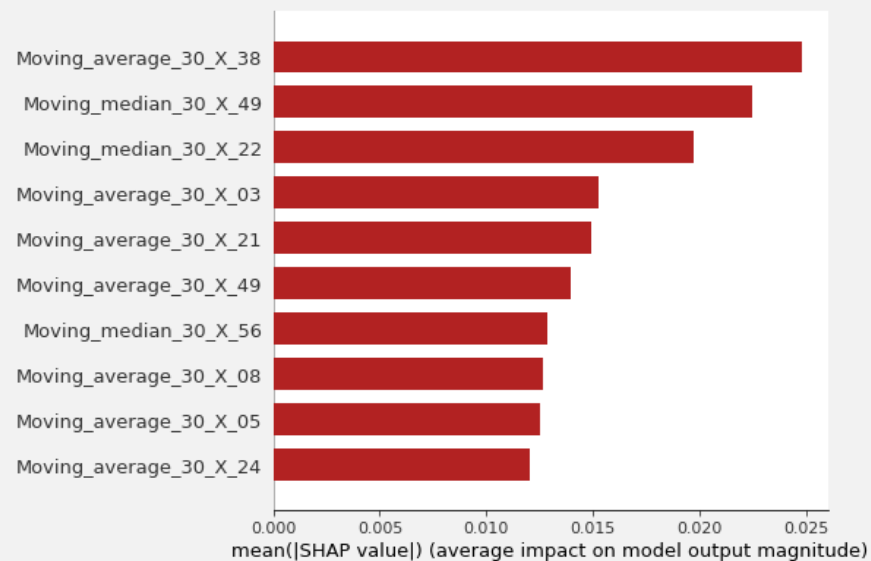




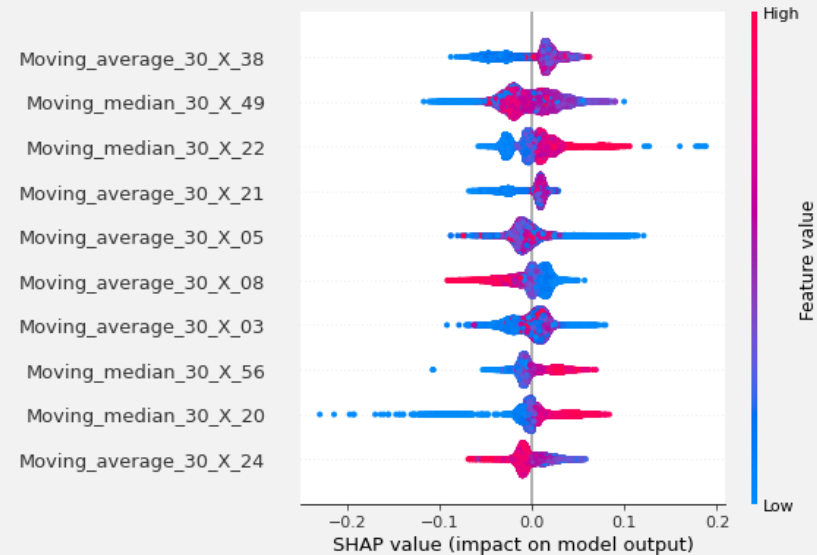
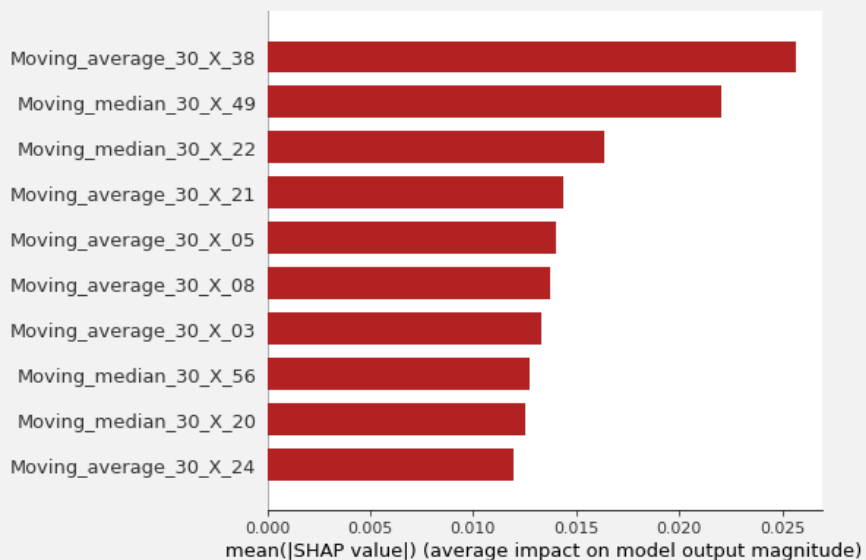
## Y\_07 Model



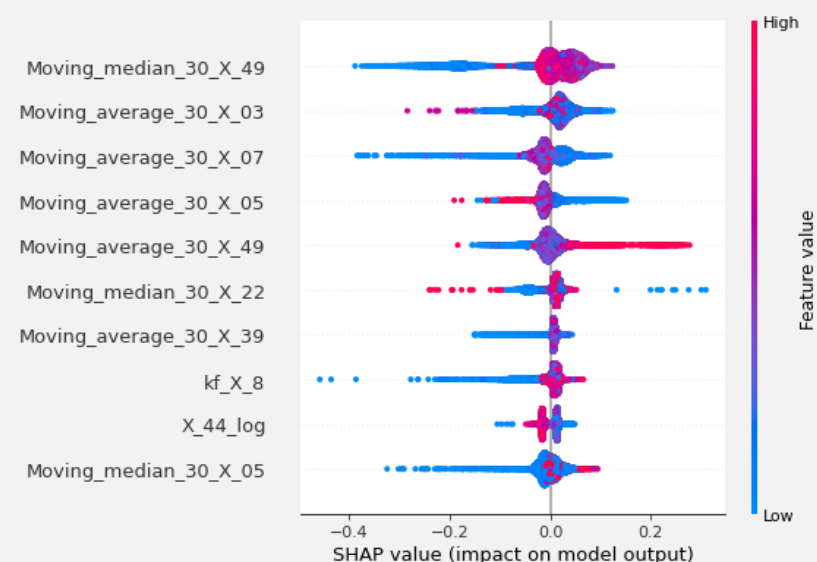
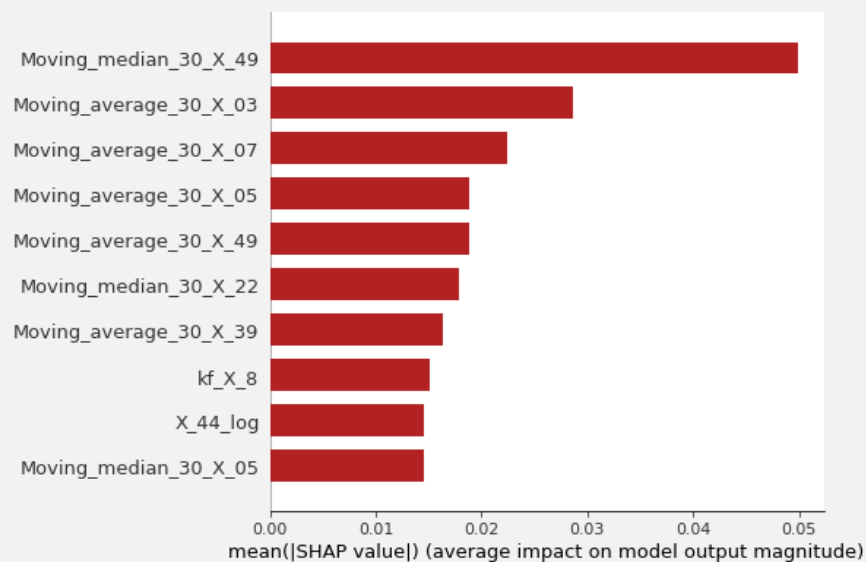
## Y\_08 Model



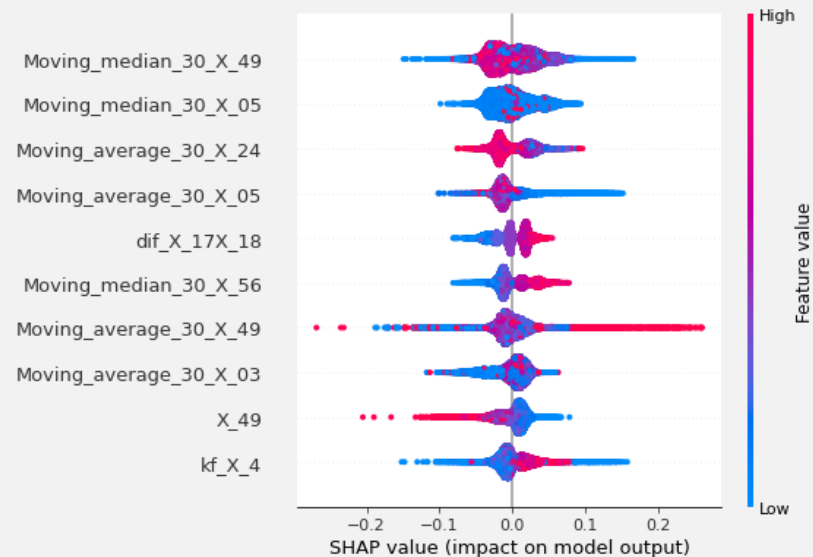
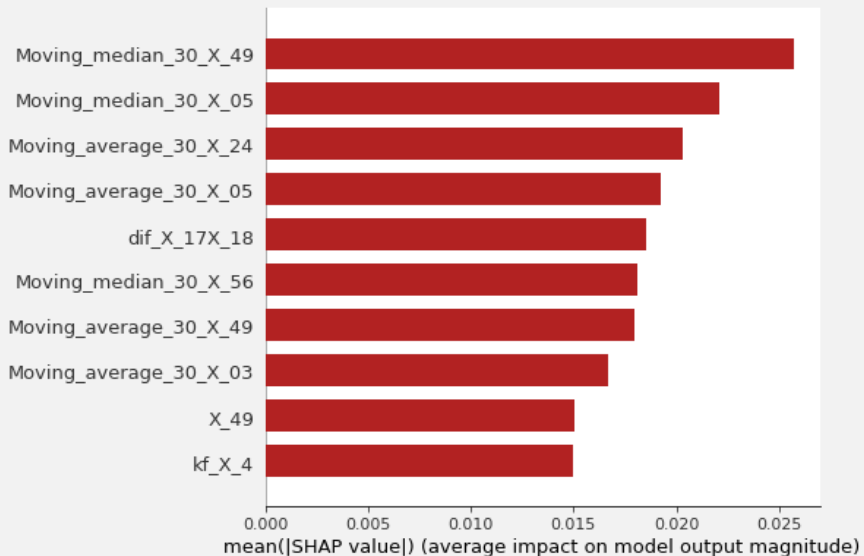
## Y\_09 Model



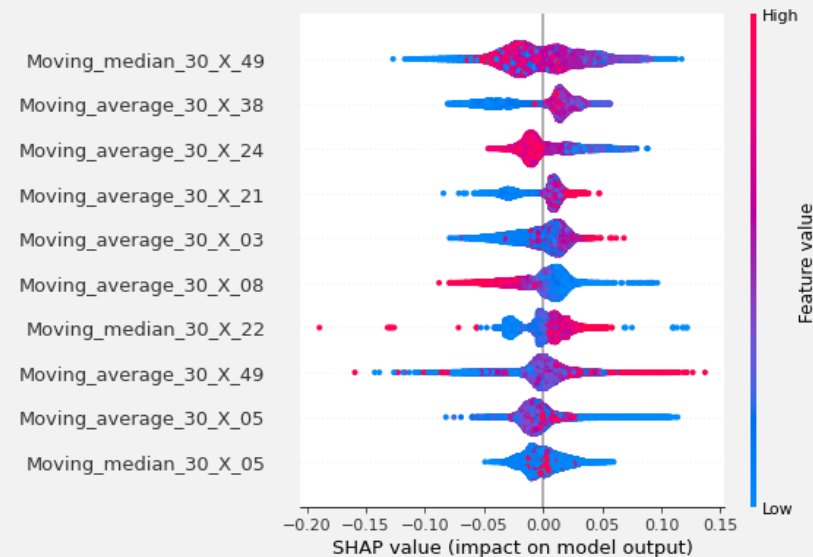
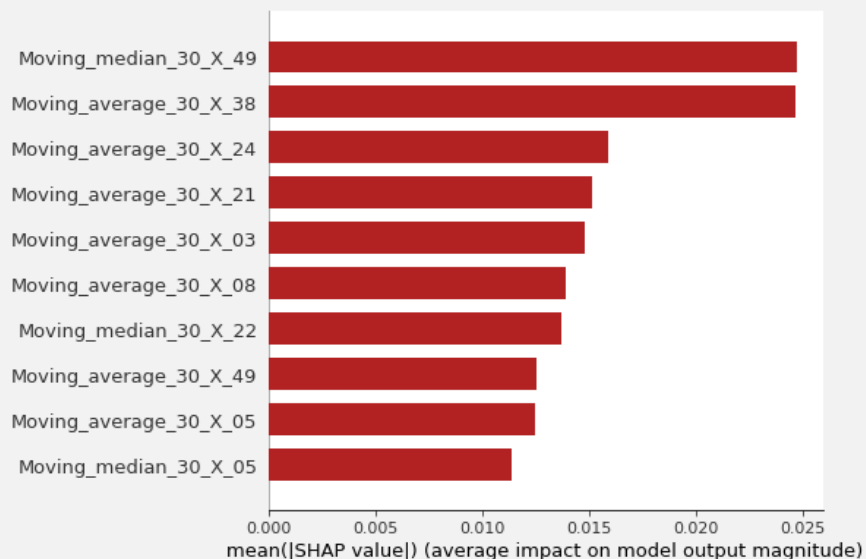
## Y\_10 Model



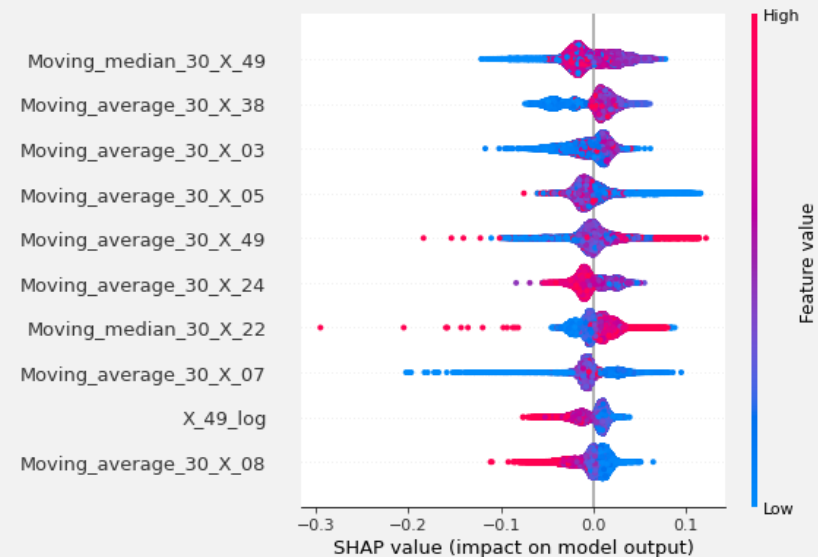
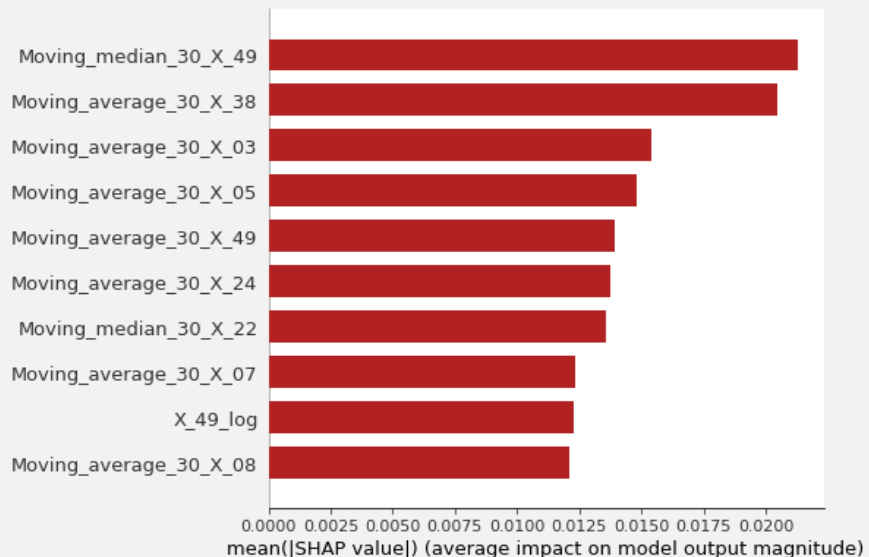
## Y\_11 Model



## Y\_12 Model



## Y\_13 Model



## Y\_14 Model

