# Anomaly detection methods

2023-10-31

증강지능연구실 황승현

# 목차

# Outlier detection 소개

# Anomaly detection

- 이상치 탐지(Outlier detection)
- 이상치(Outlier)
  - 관측된 데이터의 범위에서 많이 벗어난 아주 작은 값이나 큰 값

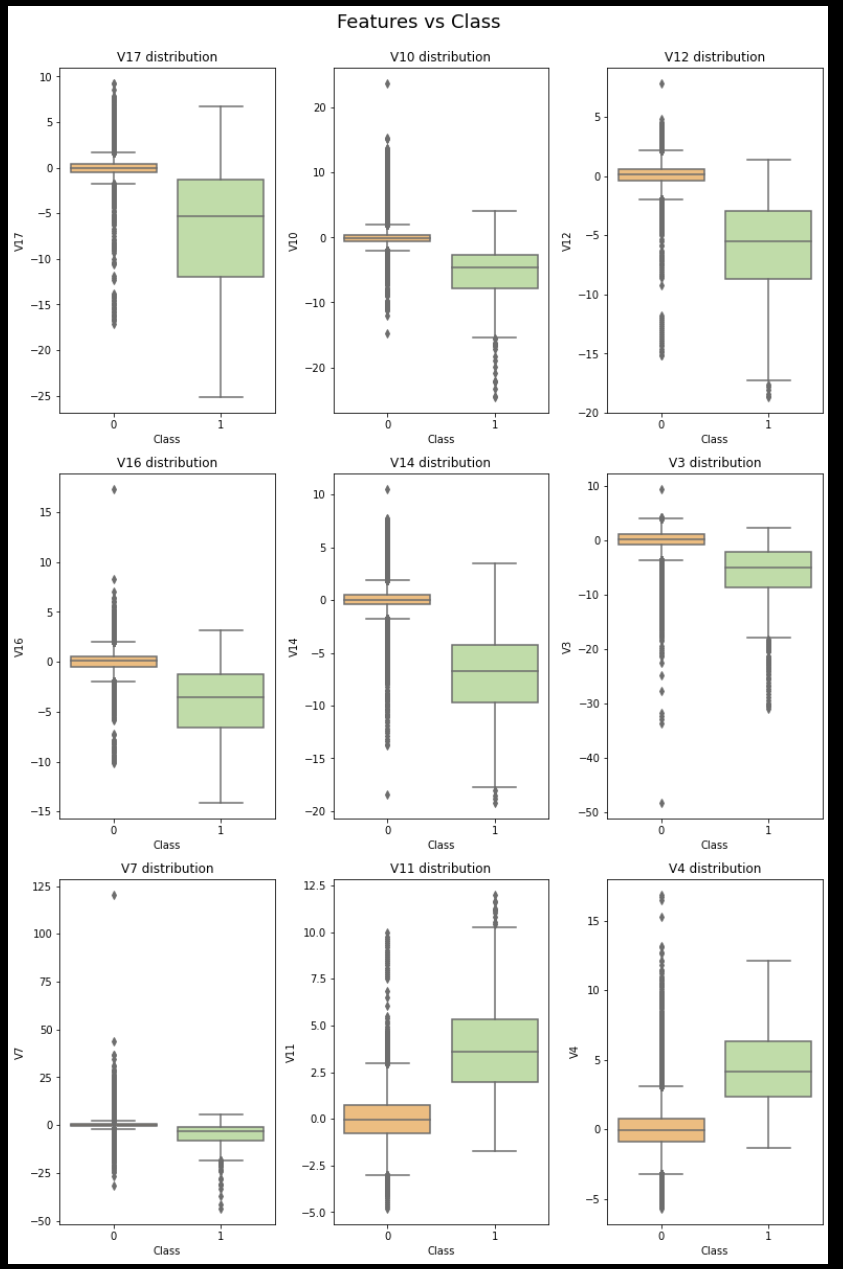
- 이상치 탐지를 하는 이유
- Imbalanced Classification에서 정상과 이상치를 구분하여 분류

# Credit Card Fraud Detection

- 신용카드 사기거래 (고객이 구매하지 않은 상품 구매) 탐지
- 데이터 소개
  - 전체: 284,807
  - 사기: 492
  - 매우 불균형, Positive(사기) : 0.172%
- Features: V1, V2, ... V28

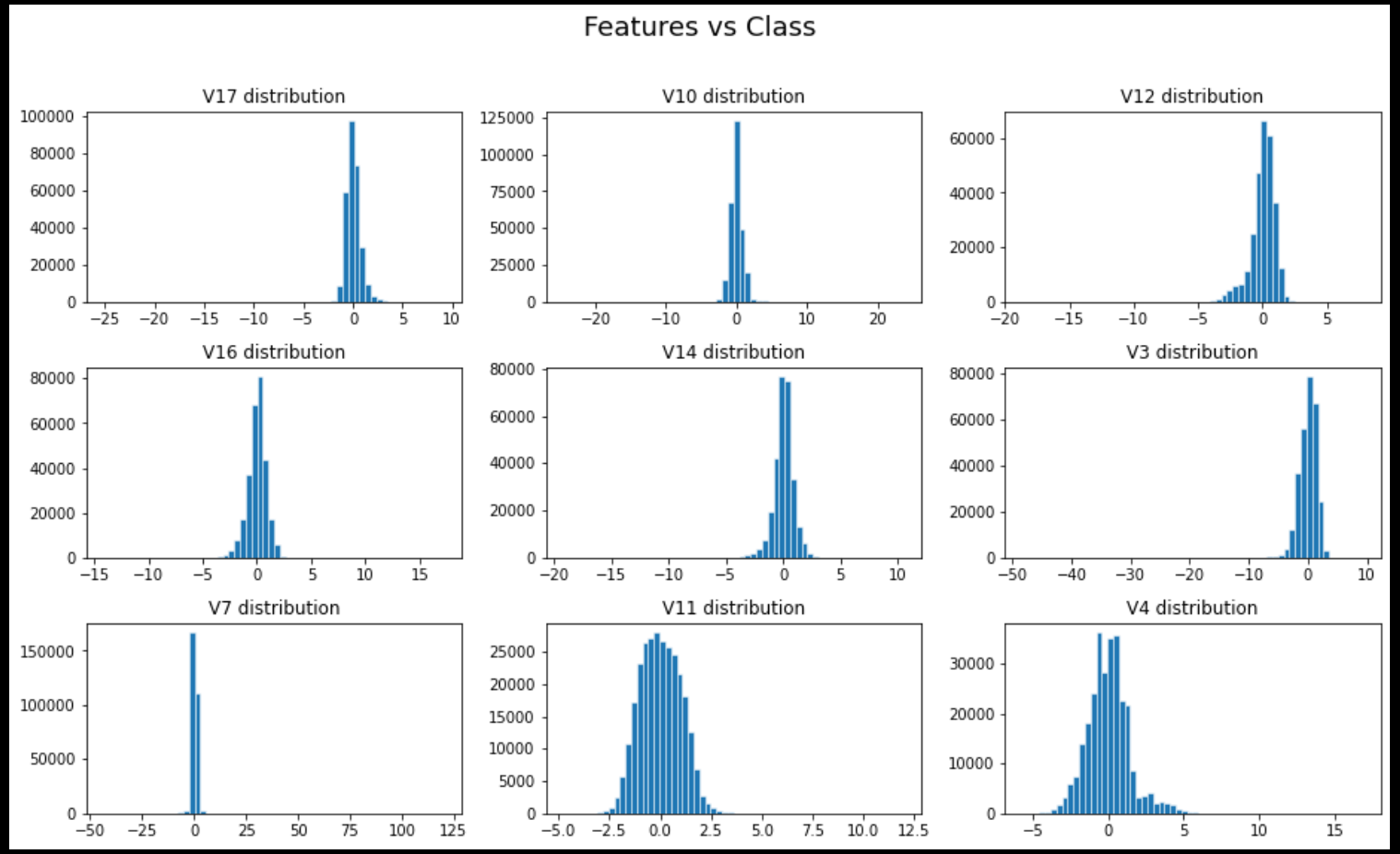
# Data distribution

Box plot



# Data distribution

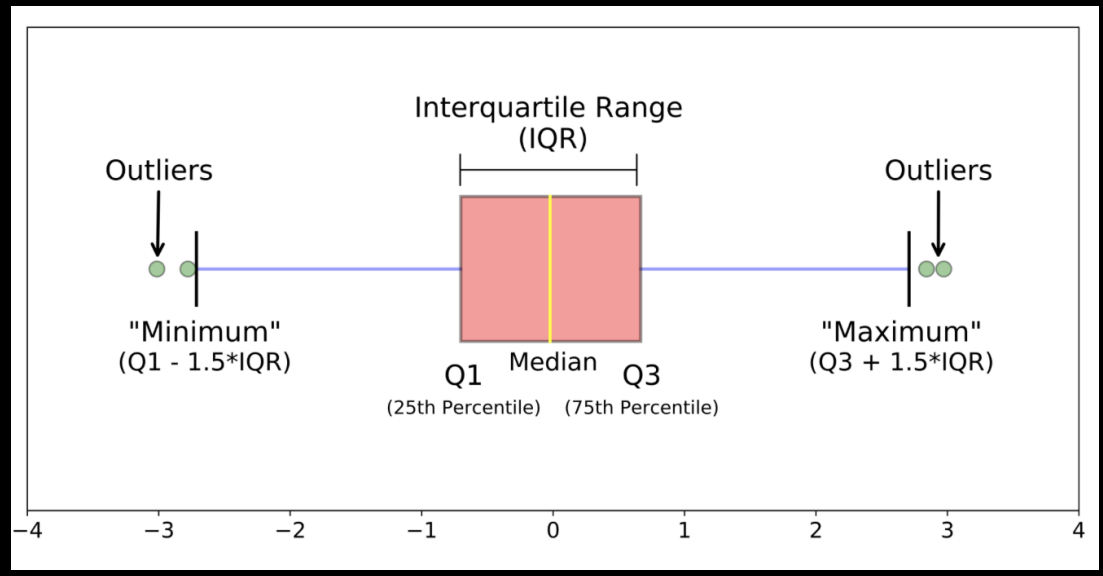
Histogram



# Tukey's IQR method

# Tukey's IQR method

- Tukey Fences
- 사분위수를 이용한 이상치 탐지
- IQR(사분범위): Q3 – Q1
- (Q1 - 1.5 IQR) and (Q3 + 1.5 IQR) 외는 모두 이상치
- 데이터 수 많아야 함

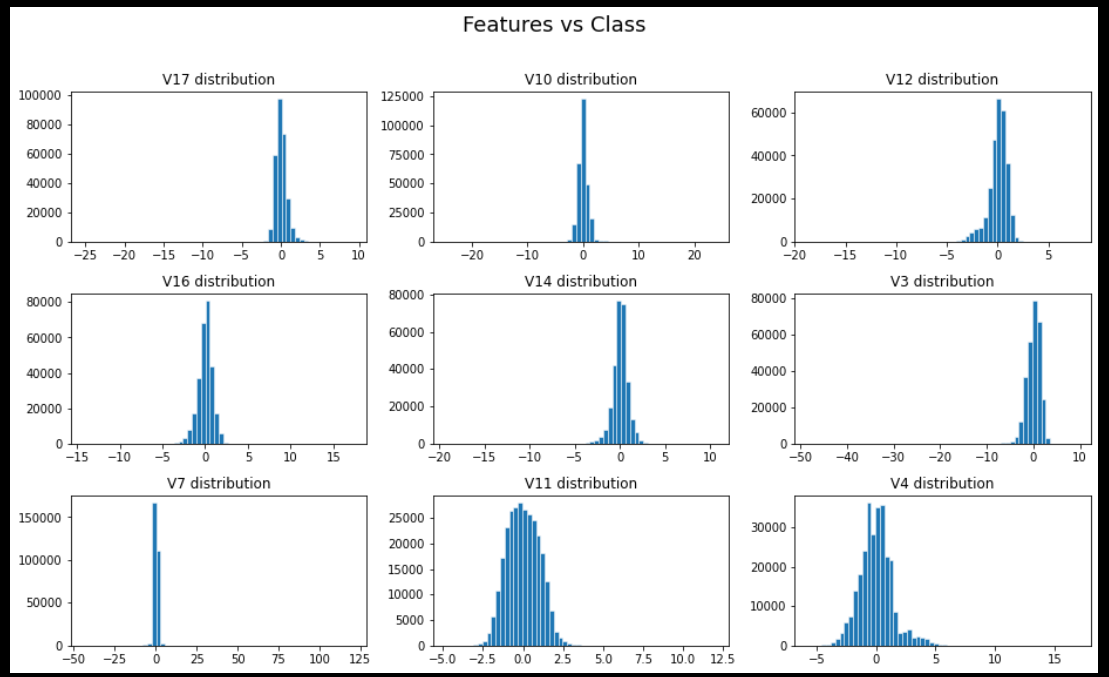


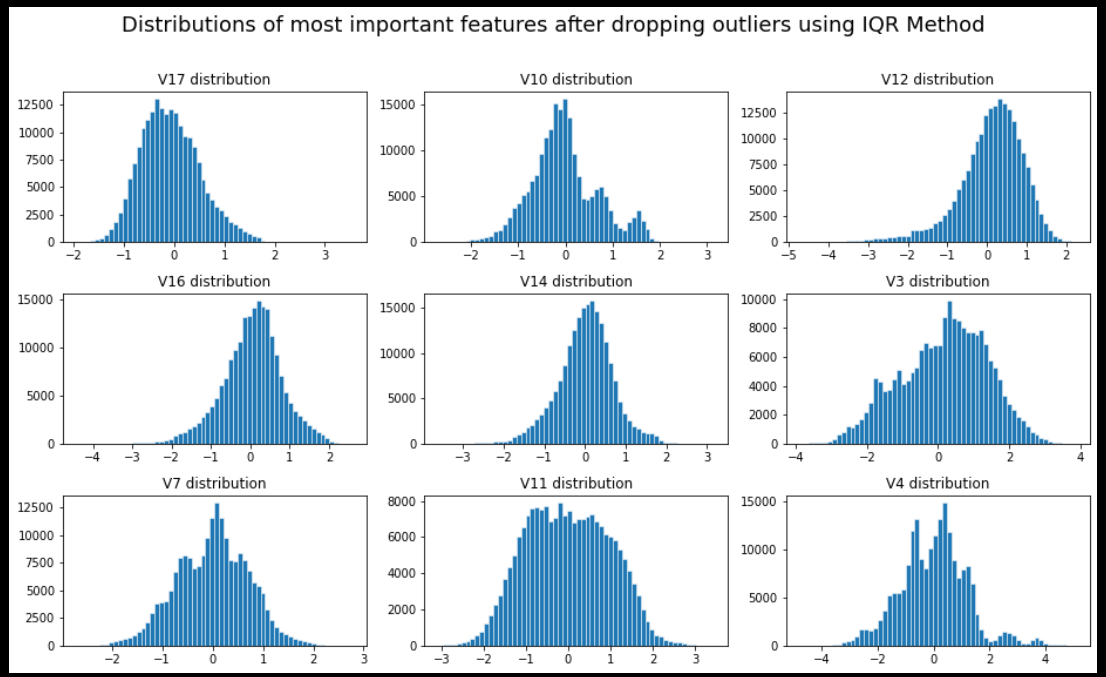
# Tukey's IQR method - code

- DataFrame의 각 feature의 범위 계산
- 범위 넘는 곳은 outlier

```python
def IQR_method (df,n,features):
    """
    Takes a dataframe and returns an index list corresponding to the observations
    containing more than n outliers according to the Tukey IQR method.
    """
    outlier_list = []

    for column in features:

        # 1st quartile (25%)
        Q1 = np.percentile(df[column], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[column],75)

        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determining a list of indices of outliers
        outlier_list_column = df[(df[column] < Q1 - outlier_step) | (df[column] > Q3 + outlier_step )].index

        # appending the list of outliers
        outlier_list.extend(outlier_list_column)

    # selecting observations containing more than x outliers
    outlier_list = Counter(outlier_list)
    multiple_outliers = list( k for k, v in outlier_list.items() if v > n )

    # Calculate the number of records below and above lower and above bound value respectively
    df1 = df[df[column] < Q1 - outlier_step]
    df2 = df[df[column] > Q3 + outlier_step]

    print('Total number of outliers is:', df1.shape[0]+df2.shape[0])

    return multiple_outliers
```

# Tukey's IQR method - 비교



# Standard deviation

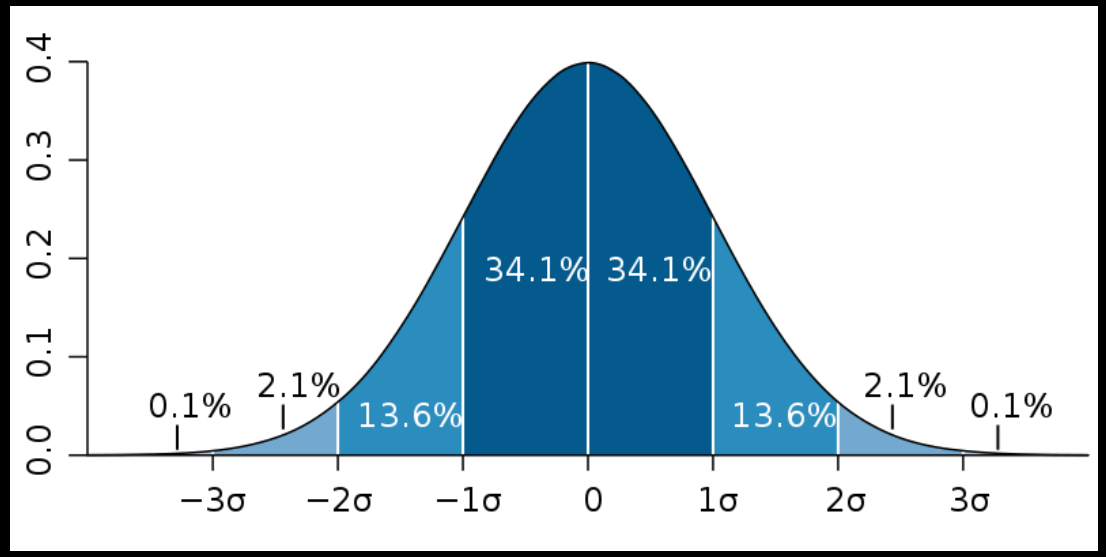
표준편차

# Standard deviation method

- 데이터셋이 가우시안 분포일 때
- 표준편차에 따라 이상치 분류할 수 있음
  - 표준편차: 평균에서 얼마나 떨어져있나

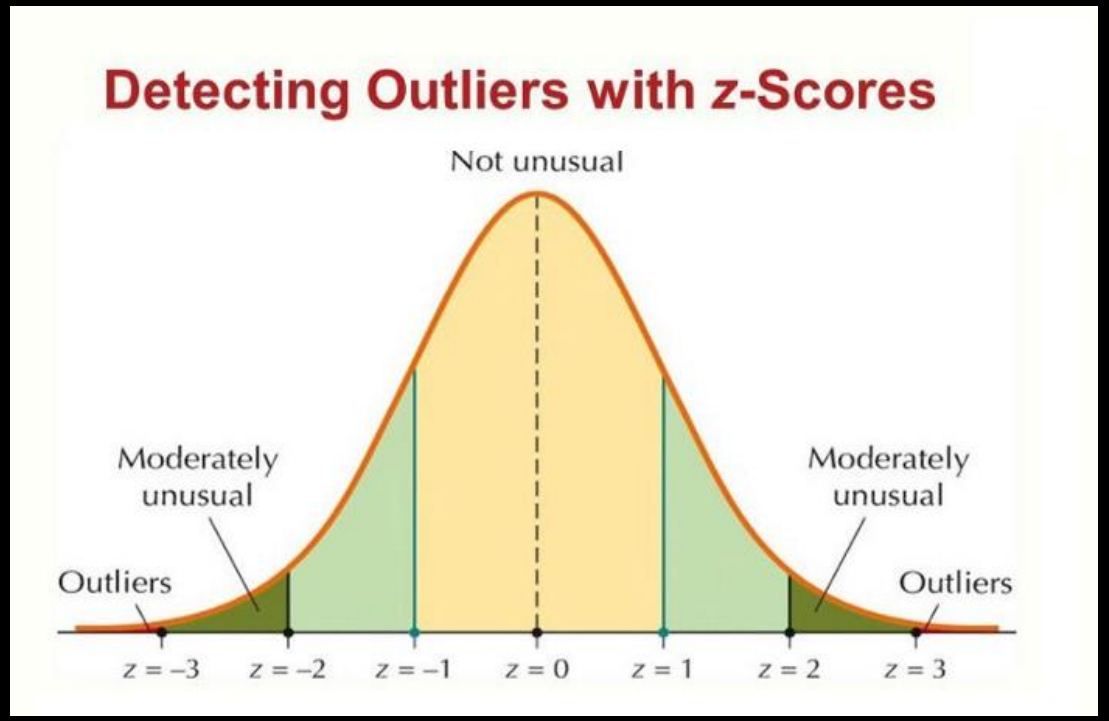
$$
\begin{aligned}
\sigma &= \sqrt{\mathrm{E}\left[(X-\mu)^2\right]} \\
&= \sqrt{\mathrm{E}\left[X^2\right] + \mathrm{E}[-2\mu X] + \mathrm{E}\left[\mu^2\right]} \\
&= \sqrt{\mathrm{E}\left[X^2\right] - 2\mu\,\mathrm{E}[X] + \mu^2} \\
&= \sqrt{\mathrm{E}\left[X^2\right] - 2\mu^2 + \mu^2} \\
&= \sqrt{\mathrm{E}\left[X^2\right] - \mu^2} \\
&= \sqrt{\mathrm{E}\left[X^2\right] - (\mathrm{E}[X])^2}
\end{aligned}
$$

# Three-sigma rule




- 표준편차 * 3 범위
- 68% : 1 ± σ
- 95% : 2 ± σ
- 99.7% : 3 ± σ

# Z-score

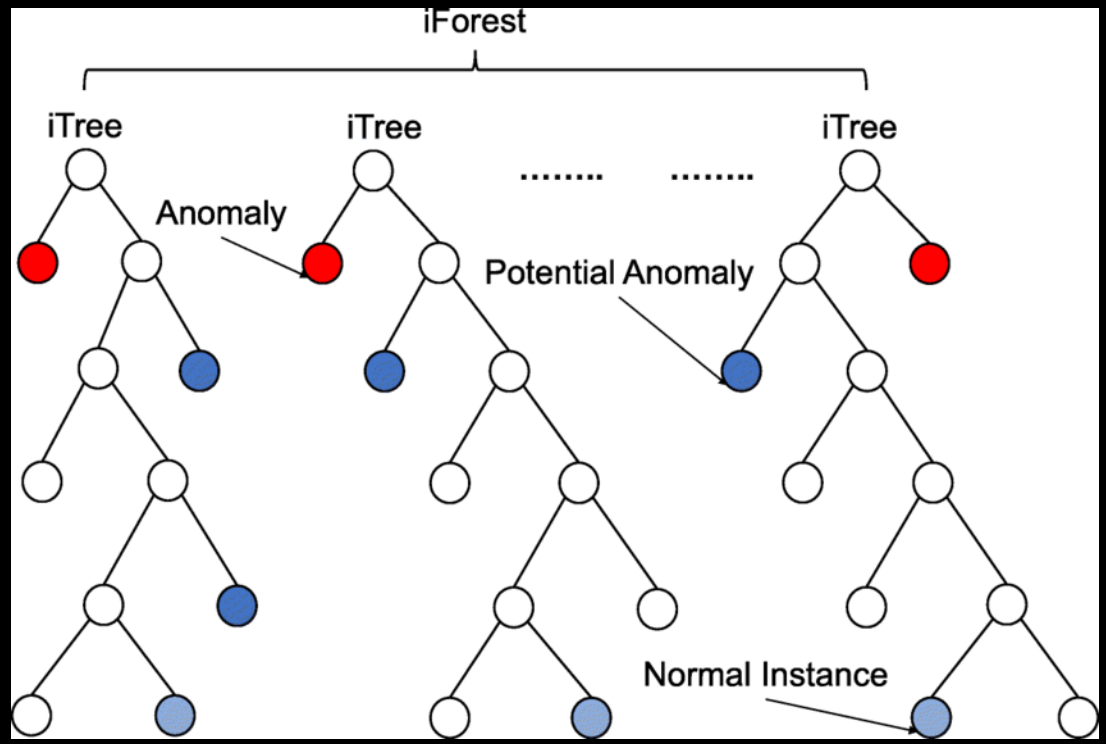



- 표준 점수
- 평균과의 거리
- abs( (df[column] – 평균)/표준편차 )

# Isolation Forest

# Isolation Forest

- Decision Tree 기반 알고리즘
- 랜덤 포레스트의 변형
- 이상치의 특성을 이용
  - 이상치는 소수임
  - 이상치는 정상치와 다른 속성



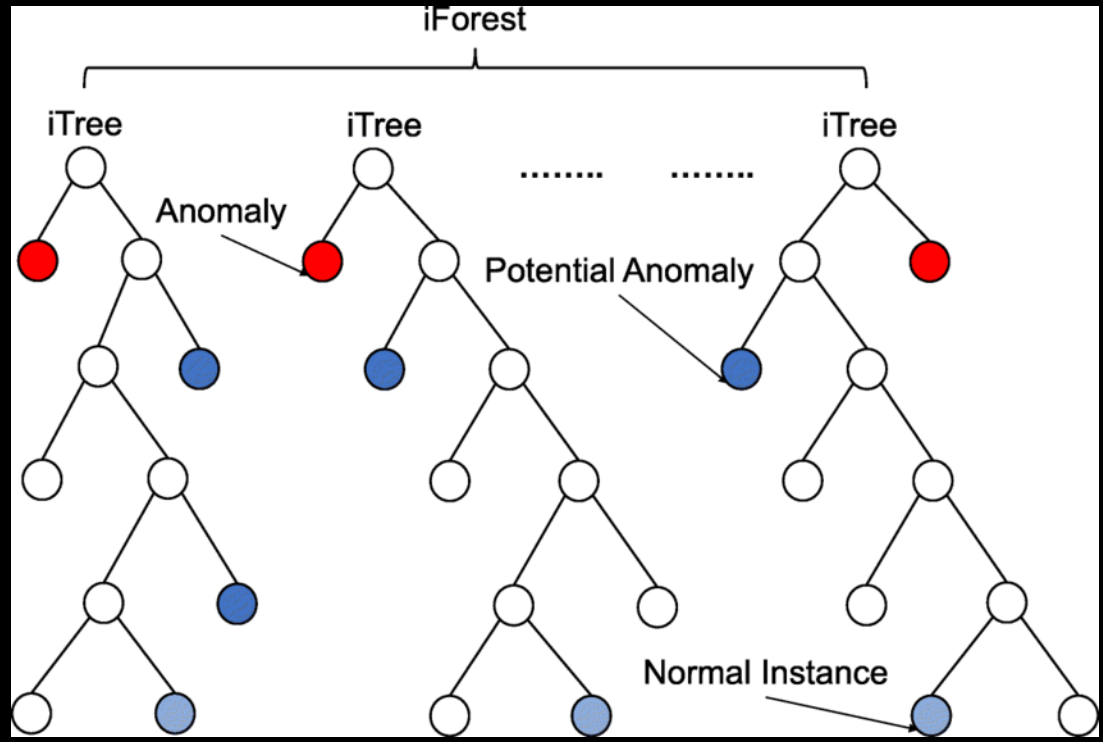
# Isolation Forest

- 정상 데이터

  분리하기 위해서 트리 깊어짐
- 이상치

  트리의 상단에서
- 데이터에서 이상치를 분리하기

  더 쉽다.



# Isolation Forest

```
from sklearn.ensemble import IsolationForest

df5 = df.copy()
df5 = df5.drop(['Class'], axis=1)

model=IsolationForest(n_estimators=150, max_samples='auto', contamination=float(0.1), max_features=1.0)
model.fit(df5)
```

- n_estimators : 트리의 수
- max_samples : 추출할 샘플의 수
- contamination : 예상되는 이상치의 비율 (0, 0.5]
- max_features : 훈련할 특징

# Isolation Forest

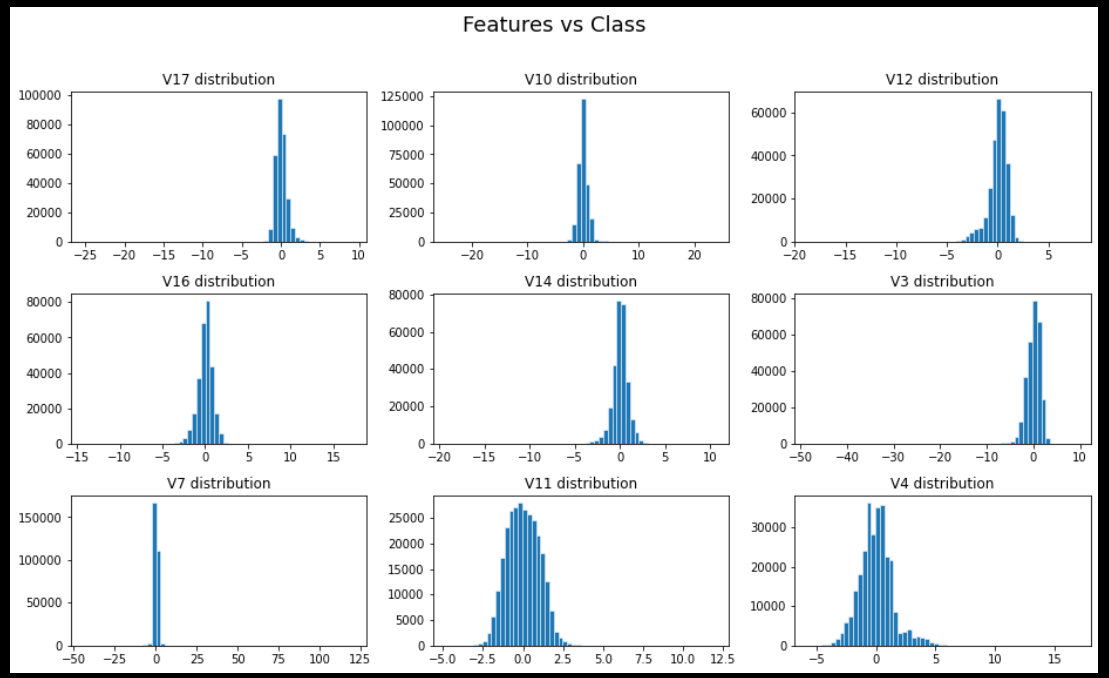
```
anomaly = df5.loc[df5['anomaly']==-1]
anomaly_index = list(anomaly.index)
print('Total number of outliers is:', len(anomaly))
```

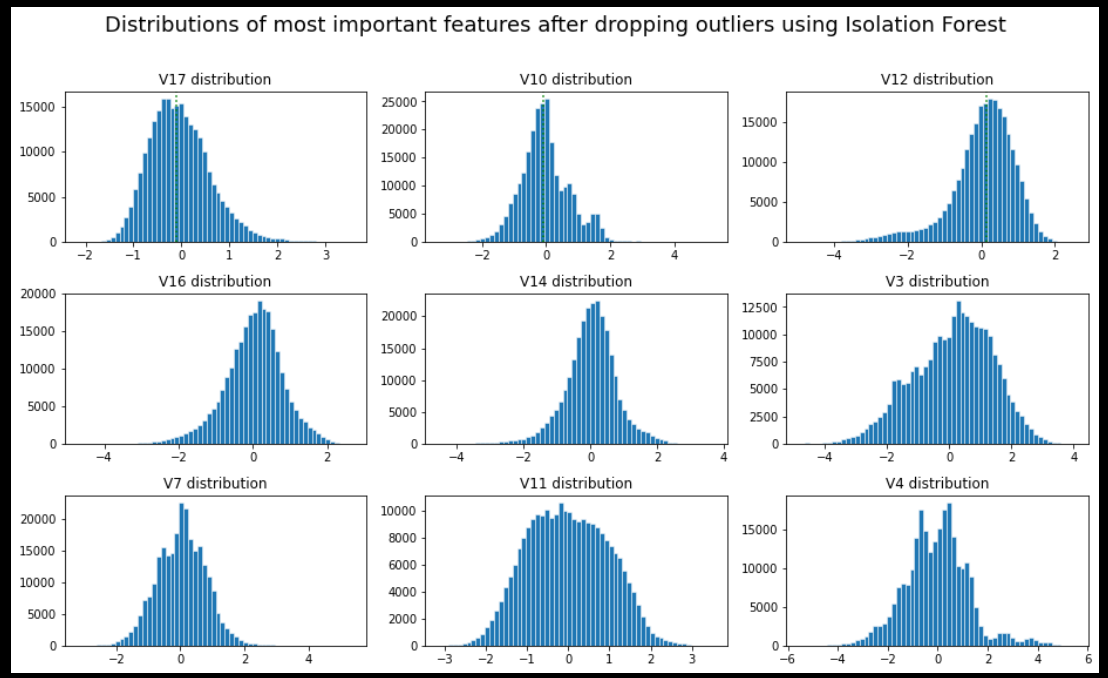
```
Total number of outliers is: 28481
```

```
df5[df5['anomaly']==-1].head(10)
```

| V4 | V5 | V6 | V7 | V8 | V9 | V10 | ... | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | scores | anomaly |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | 0.207643 | ... | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | -0.014630 | -1 |
| -0.492199 | 0.948934 | 0.428118 | 1.120631 | -3.807864 | 0.615375 | 1.249376 | ... | -1.015455 | 0.057504 | -0.649709 | -0.415267 | -0.051634 | -1.206921 | -1.085339 | 40.80 | -0.002724 | -1 |
| 1.736239 | 3.049106 | -1.763406 | -1.559738 | 0.160842 | 1.233090 | 0.345173 | ... | 0.984460 | 2.458589 | 0.042119 | -0.481631 | -0.621272 | 0.392053 | 0.949594 | 46.80 | -0.039431 | -1 |
| 3.710061 | -6.631951 | 5.122103 | 4.371691 | -2.006868 | -0.278736 | -0.230873 | ... | -0.381671 | 0.969719 | 0.019445 | 0.570923 | 0.333278 | 0.857373 | -0.075538 | 1402.95 | -0.105667 | -1 |
| -2.418473 | 0.306326 | -0.824575 | 2.065426 | -1.829347 | 4.009259 | 6.051521 | ... | -0.181268 | -0.163747 | 0.515821 | 0.136318 | 0.460054 | -0.251259 | -1.105751 | 1.46 | -0.093198 | -1 |
| -4.515824 | 0.327567 | -0.174469 | 0.959726 | -1.026456 | 1.700435 | -0.078942 | ... | 0.334614 | -0.364541 | -0.310186 | -0.302599 | -1.243924 | -1.123457 | -0.734351 | 89.17 | -0.020309 | -1 |
| 0.903915 | 3.002224 | -0.491078 | -2.705393 | 0.666451 | 1.922216 | -0.614312 | ... | 0.853360 | -0.971600 | -0.114862 | 0.408300 | -0.304576 | 0.547785 | -0.456297 | 200.01 | -0.050806 | -1 |
| 0.027094 | -1.698307 | 0.460188 | 0.737344 | -0.314216 | -0.842673 | 0.017276 | ... | 0.026123 | -1.134769 | -0.654958 | 0.098386 | -0.209150 | -0.171709 | 0.208057 | 1142.02 | -0.013705 | -1 |
| -0.363736 | 1.460953 | -0.204833 | 0.905819 | -3.384123 | 0.388546 | 0.791069 | ... | -0.883218 | -0.247698 | -0.758606 | 0.086450 | 0.202790 | -0.898858 | -0.944337 | 0.76 | -0.011087 | -1 |
| -1.748371 | 0.578943 | -0.832531 | 1.901440 | -1.913986 | 2.112375 | 3.014355 | ... | 0.073323 | -0.061693 | 0.547204 | -0.466798 | 0.408030 | -2.377933 | -1.255549 | 7.69 | -0.016784 | -1 |

# Isolation Forest



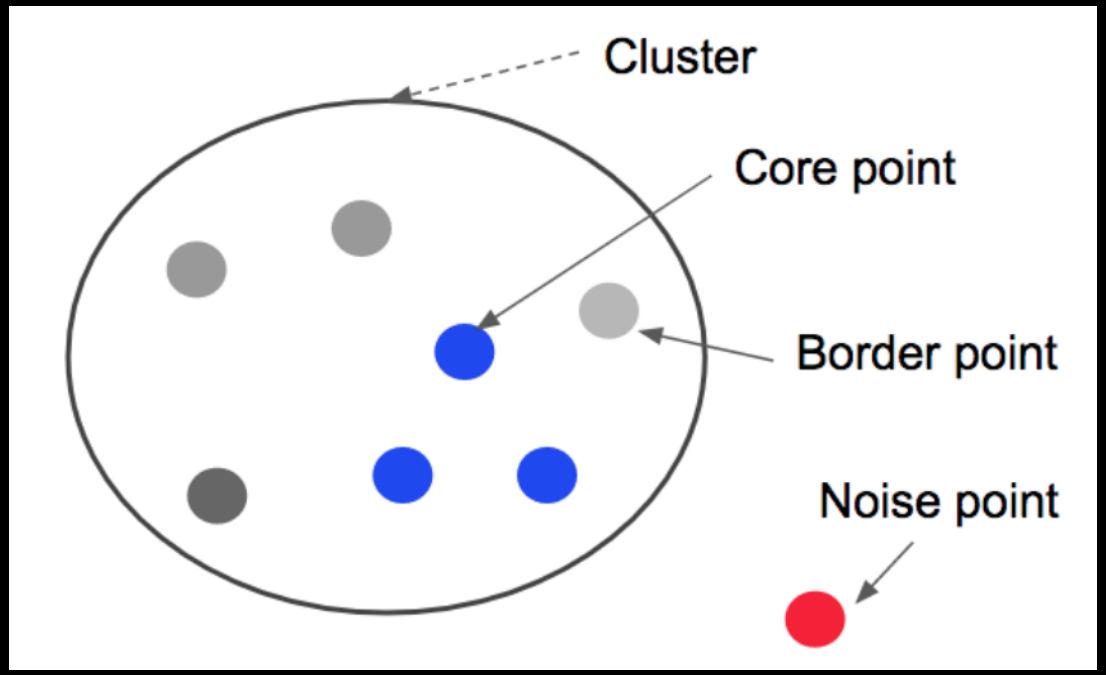
# DBSCAN

Density-Based Spatial Clustering of Applications with Noise

밀도 기반 데이터 클러스터링

# DBSCAN

- 밀도 기반 클러스터링
- 점이 세밀하게 모여 있는 부분을 클러스터링
- 이상치 : 클러스터에 속하지 않음



# DBSCAN

```python
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# scale data first
X = StandardScaler().fit_transform(df6.values)

db = DBSCAN(eps=3.0, min_samples=10).fit(X)
labels = db.labels_
```

```python
pd.Series(labels).value_counts()
```

```
 0     197235
 1      32473
-1      20243
 9      16078
 2      12290
 5       2047
 7       1416
 10       828
 3        333
 24       287
 12       214
 16       213
 13       206
 17       181
 11       170
 18        81
 4         80
 30        48
 37        36
 20        32
 22        32
 23        25
 31        24
 14        21
 29        20
 15        19
 25        18
 28        17
 26        16
 33        16
 21        15
 32        14
 34        12
 35        12
 38        10
 6         10
 8         10
 36        10
 19         8
 27         7
dtype: int64
```