KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2022

스마트교통 빅데이터 분석

VDS 데이터 분석 및 기계학습 적용 실습하기

2022.3.28.

이 홍 석 (hsyi@kisti.re.kr)

교통 데이터 다운로드

국가교통데이터 오픈마켓

대전시 VDS 운행 원시 이력 데이터

- VDS(Vehicle Detection System) : 차량검지기
  - ✓도로 상에 약 1km 간격으로 설치되어 실시간으로 교통량, 점유율, 속도, 대기행렬길이, 차량길이 등의 정보를 검지하여 소통 및 돌발상황 등을 감시하는 장치로 도로 환경적 특성에 따라 설치하며 종류는 루프식, 영상식, 레이더식 등이 있다
  - ✓대전시 VDS 운행 원시 이력 데이터 (등록일자, 검지기 ID, VDS ID, VDS 구간 ID, 요일 구분, 교통량 (소/중/대형), 속도, 점유율, 차두 길이, 차두 시간 등)

도로 0 대전광역시

대전시 VDS 운행 원시 이력 데이터

대전시 VDS 운행 원시 이력 데이터 [vds-collect-info.zip] 외 5건(80.74MB)

결제금액 합계 : 0 원

결제완료

상품 다운로드

문의하기

교통 데이터 다운로드

http://tportal.daejeon.go.kr/stats/content01.view?search=1

교통데이터 DW 시스템 (대전시)

판다스 연습

1. 데이터 오브젝트 생성하기

- Series
  - ✓Series는 같이 값들의 리스트를 넘겨주어 만듬
  - ✓인덱스는 자동적으로 기본적으로 지정되는 정수 인덱스를 사용
- 1차원 배열 같은 구조 (예)
  - ✓엑셀로 보면 복수의 행(row)으로 이루어진 1개의 열,
  - ✓복수의 열(column)으로 이루어진 하나 행(row)의 값
  - ✓엑셀처럼, column들이 색인(index)을 가지고 있어 데이터를 지칭하는 이름을 가짐

DataFrame 만들기


```
df = pd.DataFrame(np.random.randn(8,5), index=dates, columns=list('abcde'))
```

```
df
```

	a	b	c	d	e
2020-01-01	-0.802490	-0.670454	0.571352	-1.001153	0.205154
2020-01-02	-0.802341	0.329855	1.889353	-1.574091	-0.259556
2020-01-03	0.272315	-0.678223	-0.328725	-0.066139	0.158881
2020-01-04	-1.103216	-0.831071	2.872117	1.987097	-1.521131
2020-01-05	0.218559	0.072759	0.531204	0.785738	-0.294122
2020-01-06	-1.433724	0.648570	-0.498555	-1.563853	-0.988152
2020-01-07	-0.867036	-1.141157	0.638432	1.491141	-0.433551
2020-01-08	2.700744	-1.400121	1.222277	-0.602262	0.200138

DataFrame 만들기 다른 방법

In [30]:
```
df2 = pd.DataFrame({'a': 1.,
                    'b': pd.Timestamp('20130102'),
                    'c': pd.Series(1, index=list(range(4)), dtype='float32'),
                    'd': np.array([3]*4, dtype='int32'),
                    'e': pd.Categorical(['test', 'train', 'test', 'train']),
                    'f': 'foo'})
```

In [31]:
```
df2
```

Out[31]:

	a	b	c	d	e	f
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

2. 데이터 확인하기

head(), head(5) 위에서 부터 데이터 보여줌

```
df.head()
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801

```
df.head(3)
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681

```
df.tail()
```

	a	b	c	d	e
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

```
df.tail(3)
```

	a	b	c	d	e
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

DataFrame 속성 확인

DataFrame 속성 확인 index, columns, values

```
df.index
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08'],
              dtype='datetime64[ns]', freq='D')
```

```
df.columns
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

```
df.values
```

```
array([[-0.1617311 ,  0.84452809, -0.43382289,  0.38187277,  1.10848399],
       [ 0.97981521, -0.08804282, -0.22897753, -1.39795857, -1.39874591],
       [ 1.27615329, -1.47740467,  0.9570735 , -0.5431942 , -0.39568057],
       [ 0.00644179,  1.63848983,  0.73020528,  0.51741274, -1.09175269],
       [-1.10150497, -0.89583825, -0.27186124, -0.13472203,  0.17980079],
       [ 2.21416968, -1.31659282, -1.89851741, -1.265935  , -1.57235185],
       [-0.6060803 , -2.32160025,  0.18614666, -0.53517175,  1.63562266],
       [ 0.13848857, -1.13541823,  0.62121683,  1.36880715, -2.36526483]])
```

메소드 .describe()는 통계 정보

```
df.describe()
```

	a	b	c	d	e
count	8.000000	8.000000	8.000000	8.000000	8.000000
mean	0.343219	-0.593985	-0.042317	-0.201111	-0.487486
std	1.080660	1.308257	0.908243	0.935041	1.383935
min	-1.101505	-2.321600	-1.898517	-1.397959	-2.365265
25%	-0.272818	-1.356796	-0.312352	-0.723879	-1.442147
50%	0.072465	-1.015628	-0.021415	-0.334947	-0.743717
75%	1.053900	0.145100	0.648464	0.415758	0.411972
max	2.214170	1.638490	0.957074	1.368807	1.635623

컬럼별로 데이터의 개수(count), 데이터의 평균 값(mean), 표준 편차(std), 최솟값(min), 4분위수(25%, 50%, 75%), 그리고 최댓값(max)들의 정보

.sort_index()로 행과 열 정렬

정렬 메소트 .sort_index()

```
df.sort_index(axis=0, ascending=True)
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

```
df.sort_index(axis=1, ascending=True)
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

정렬 할 대상 축을 결정할때에는 axis를 이용하며, axis=0는 인덱스 기준으로 정렬(기본값), axis=1은 칼럼 기준으로 정렬함

정렬의 방향은 ascending을 이용하며, ascending=True는 오름차순 정렬(기본값); ascending=False는 내림차순 정렬

3. 데이터 선택하기 (Selection)

```
df
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

```
df['a']
```

```
2020-01-01   -0.161731
2020-01-02    0.979815
2020-01-03    1.276153
2020-01-04    0.006442
2020-01-05   -1.101505
2020-01-06    2.214170
2020-01-07   -0.606080
2020-01-08    0.138489
Freq: D, Name: a, dtype: float64
```

```
df[0:4]
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753

```
df['2020-01-02':'2020-01-06']
```

	a	b	c	d	e
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352

선택 df[0:4]

데이터프레임 자체가 갖고 있는 슬라이싱은
df[컬럼명], df[시작인덱스:끝인덱스+1], df[시작인덱스명:끝인덱스명]의 형태로 사용

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

.iloc는 위치를 이용하여 선택하기


```
df
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265

```
df.iloc[3]
```

```
a     0.006442
b     1.638490
c     0.730205
d     0.517413
e    -1.091753
Name: 2020-01-04 00:00:00, dtype: float64
```

다음과 같이 위치를 나타내는 인덱스 번호를 이용하여 데이터를 선택

여기서 인덱스 번호는 python에서 사용하는 인덱스와 같은 개념

인덱스 번호는 0 부터 시작하므로,
첫 번째 데이터는 인덱스 번호가 0 이고,
두 번째 데이터는 인덱스 번호가 1 임.
인덱스 번호 3 (네 번째 행)을 선택하는 예제

조건을 이용하여 선택하기

조건을 이용한 선택 ¶

```
df[df.b > 0]
```

	a	b	c	d	e
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753

```
df[df > 0]
```

	a	b	c	d	e
2020-01-01	NaN	0.844528	NaN	0.381873	1.108484
2020-01-02	0.979815	NaN	NaN	NaN	NaN
2020-01-03	1.276153	NaN	0.957074	NaN	NaN
2020-01-04	0.006442	1.638490	0.730205	0.517413	NaN
2020-01-05	NaN	NaN	NaN	NaN	0.179801
2020-01-06	2.214170	NaN	NaN	NaN	NaN
2020-01-07	NaN	NaN	0.186147	NaN	1.635623
2020-01-08	0.138489	NaN	0.621217	1.368807	NaN

필터링을 위한 isin()


```
df4 = df.copy()
df4['f'] = ['one', 'one','two','three','four','three','six','seven']
```

```
df4
```

	a	b	c	d	e	f
2020-01-01	-0.161731	0.844528	-0.433823	0.381873	1.108484	one
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746	one
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681	two
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753	three
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801	four
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352	three
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623	six
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265	seven

```
df4[df4['f'].isin(['two', 'four'])]
```

	a	b	c	d	e	f
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681	two
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801	four

데이터 변경하기

데이터 변경하기

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range('20200102', periods=6))
s1
```

```
2020-01-02    1
2020-01-03    2
2020-01-04    3
2020-01-05    4
2020-01-06    5
2020-01-07    6
Freq: D, dtype: int64
```

```
df['f']=s1
df
```

	a	b	c	d	e	f
2020-01-01	0.000000	0.844528	-0.433823	0.381873	1.108484	NaN
2020-01-02	0.979815	-0.088043	-0.228978	-1.397959	-1.398746	1.0
2020-01-03	1.276153	-1.477405	0.957074	-0.543194	-0.395681	2.0
2020-01-04	0.006442	1.638490	0.730205	0.517413	-1.091753	3.0
2020-01-05	-1.101505	-0.895838	-0.271861	-0.134722	0.179801	4.0
2020-01-06	2.214170	-1.316593	-1.898517	-1.265935	-1.572352	5.0
2020-01-07	-0.606080	-2.321600	0.186147	-0.535172	1.635623	6.0
2020-01-08	0.138489	-1.135418	0.621217	1.368807	-2.365265	NaN

4. 결손 데이터(Missing Data) 처리

- 결손 데이터(Missing Data)는 : 여러가지 이유로 데이터를 전부 다 측정하지 못하는 경우가 종종 발생하며, Pandas는 편리한 API를 제공한다. 결손데이터는 컬럼에 값이 없는 NULL을 의미하며, 넘파이의 NaN으로 표시, 즉, np.nan
- pandas는 결손 데이터를 NaN을 처리하지 않으므로 이 값을 다른 값으로 대체해야 함. 또한 NaN 값은 평균, 총합 등의 함수 연사 시 제외 됨.
- NaN 여부를 확인하는 API는 isna()이며, NaN 값을 다른 값으로 대체하는 API는 fillna().
- 재인덱싱은 해당 축에 대하여 인덱스를 변경/추가/삭제를 함.

결측치: dropna(), fillna(), isna()

fillna()로 결손데이터 대체하기

```
df1.fillna(value=999)
```

	a	b	c	d	e	f	g
2020-01-01	0.000000	1.737818	0.449463	0.700941	-0.350025	999.0	1.0
2020-01-02	-0.699451	-0.241029	0.591983	-0.625677	-0.141708	1.0	1.0
2020-01-03	-0.863738	1.314997	-1.633504	-1.316650	-1.132148	2.0	999.0
2020-01-04	-1.619386	2.101894	-1.303257	-0.771174	-0.865577	3.0	999.0

```
df1
```

	a	b	c	d	e	f	g
2020-01-01	0.000000	1.737818	0.449463	0.700941	-0.350025	NaN	1.0
2020-01-02	-0.699451	-0.241029	0.591983	-0.625677	-0.141708	1.0	1.0
2020-01-03	-0.863738	1.314997	-1.633504	-1.316650	-1.132148	2.0	NaN
2020-01-04	-1.619386	2.101894	-1.303257	-0.771174	-0.865577	3.0	NaN

apply lambda 함수 이용해보기

```
def get_square(a):
    return a**2

print('4의 제곱은:', get_square(4))
```

4의 제곱은: 16

위는 함수의 정의이고, lambda는 보다 쉽게 적용

```
lambda_square = lambda x: x ** 2
print('4의 제곱은:', lambda_square(4))
```

4의 제곱은: 16

```
df=df[0:3]; df
```

	a	b	c	d	e	f
2020-01-01	0.000000	1.737818	0.449463	0.700941	-0.350025	NaN
2020-01-02	-0.699451	-0.241029	0.591983	-0.625677	-0.141708	1.0
2020-01-03	-0.863738	1.314997	-1.633504	-1.316650	-1.132148	2.0

```
df.apply(np.cumsum)
```

	a	b	c	d	e	f
2020-01-01	0.000000	1.737818	0.449463	0.700941	-0.350025	NaN
2020-01-02	-0.699451	1.496789	1.041446	0.075265	-0.491733	1.0
2020-01-03	-1.563189	2.811785	-0.592058	-1.241385	-1.623881	3.0

판다스는 apply 함수에 lambda 식을 결합해 DataFrame이나 Series의 레코드별로 데이터를 가공 기능 제공

복잡한 데이터 가공의 경우 어쩔 수 없이 apply lambda를 적용한다.

```
df.apply(lambda x: x.max() - x.min())
```

```
a    0.863738
b    1.978847
c    2.225487
d    2.017591
e    0.990441
f    1.000000
dtype: float64
```

```
df.mean()
```

```
a   -0.521063
b    0.937262
c   -0.197353
d   -0.413795
e   -0.541294
f    1.500000
dtype: float64
```

- 같은 형태의 자료를 이어주는 concat
- 다른 형태의 자료를 한 컬럼으로 합치는 merge
- 기존 데이터 프레임에 하나의 행을 추가하는 append

6. 합치기

```
df = pd.DataFrame(np.random.randn(10, 4))
df
```

	0	1	2	3
0	-0.962940	-0.104394	-0.074647	1.533884
1	-0.600499	0.213632	-0.040409	0.642713
2	-0.465268	-1.266487	-1.023249	1.022946
3	-0.286531	2.373228	-0.775739	0.583997
4	0.361653	0.225475	-1.397136	-0.346902
5	-2.350007	-0.667676	-0.299217	0.278333
6	1.799277	0.516733	1.873353	1.150520
7	-0.407686	0.463574	-1.591343	0.109620
8	1.188445	0.644770	0.427377	-0.325065
9	1.067360	1.295874	1.302318	-0.602613

```
pieces = [df[:3], df[3:7], df[7:]]
pieces
```

```
[          0         1         2         3
 0 -0.962940 -0.104394 -0.074647  1.533884
 1 -0.600499  0.213632 -0.040409  0.642713
 2 -0.465268 -1.266487 -1.023249  1.022946,
           0         1         2         3
 3 -0.286531  2.373228 -0.775739  0.583997
 4  0.361653  0.225475 -1.397136 -0.346902
 5 -2.350007 -0.667676 -0.299217  0.278333
 6  1.799277  0.516733  1.873353  1.150520,
           0         1         2         3
 7 -0.407686  0.463574 -1.591343  0.109620
 8  1.188445  0.644770  0.427377 -0.325065
 9  1.067360  1.295874  1.302318 -0.602613]
```

```
pd.concat(pieces)
```

	0	1	2	3
0	-0.962940	-0.104394	-0.074647	1.533884
1	-0.600499	0.213632	-0.040409	0.642713
2	-0.465268	-1.266487	-1.023249	1.022946
3	-0.286531	2.373228	-0.775739	0.583997
4	0.361653	0.225475	-1.397136	-0.346902
5	-2.350007	-0.667676	-0.299217	0.278333
6	1.799277	0.516733	1.873353	1.150520
7	-0.407686	0.463574	-1.591343	0.109620
8	1.188445	0.644770	0.427377	-0.325065
9	1.067360	1.295874	1.302318	-0.602613

8. Time Series 데이터 처리

- 시계열 데이터에서 1초 마다 측정된 데이터를 5분 마다 데이터의 형태 변환가능
  - ✓주기(frequency)를 다시 샘플링 할 수 있는 단순하고, 강력하며, 효과적인 기능
  - ✓응용으로 금융, 교통, 기상 데이터를 다룰 때 매우 흔히 하는 연산

```
rng = pd.date_range('1/1/2012', periods=8, freq='S');rng
```

```
DatetimeIndex(['2012-01-01 00:00:00', '2012-01-01 00:00:01',
               '2012-01-01 00:00:02', '2012-01-01 00:00:03',
               '2012-01-01 00:00:04', '2012-01-01 00:00:05',
               '2012-01-01 00:00:06', '2012-01-01 00:00:07'],
              dtype='datetime64[ns]', freq='S')
```

```
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng);ts
```

```
2012-01-01 00:00:00     37
2012-01-01 00:00:01    410
```

판다스에서 파일 다루기

- 다양한 포멧을 파일을 DataFrame로 로딩하는 API제공
  - ✓ 대표적으로 read_csv(), read_table(), read_fwf()
- CSV는 컬럼을 ,로 구분이고 read_table은 '₩t' 탭으로 구분됨
  - ✓예제, file name은 titanic.csv 파일이 있다면,
    - 판다스에서, df=pd.read_csv(r'C:₩Users₩hsyi₩work₩titanic.csv')
    - df.head(4)로 확인함

VDS 데이터 머신러닝

차량검지기 VDS 데이터 분석

plot_ml_curve 라이브러리 설정하기


```
import numpy as np
def plot_ml_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 20)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

    plt.legend(loc='lower right')
    return plt
```

데이터 가져오기

```
[10] import pandas as pd
```

```
[11] from pandas import datetime
```

```
[12] def parser(x):
        return datetime.strptime(x, '%Y-%m-%d %H:%M')
```

```
[13] df = pd.read_csv('./daejeon_vds16.csv',  date_parser=parser)
```

```
[14] df.head()
```

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
0	2017-04-02 0:00	43	34	9	0	50.3	1.90
1	2017-04-02 0:05	45	32	13	0	58.9	1.84
2	2017-04-02 0:10	46	34	12	0	50.6	1.87
3	2017-04-02 0:15	45	36	9	0	50.9	1.72
4	2017-04-02 0:20	27	13	13	1	62.2	1.12

df.info()


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8064 entries, 0 to 8063
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      8064 non-null   object
 1   ToVol     8064 non-null   int64
 2   SmVol     8064 non-null   int64
 3   MeVol     8064 non-null   int64
 4   LaVol     8064 non-null   int64
 5   Speed     8064 non-null   float64
 6   Occ.Rate  8064 non-null   float64
dtypes: float64(2), int64(4), object(1)
memory usage: 441.1+ KB
```

df.describe()


```
[16] df.describe()
```

초

	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
count	8064.000000	8064.000000	8064.000000	8064.000000	8064.000000	8064.000000
mean	110.459945	79.353299	29.948537	1.158110	49.327431	6.166941
std	63.954451	46.802106	19.081136	1.530192	7.921856	6.739946
min	6.000000	2.000000	0.000000	0.000000	9.100000	0.230000
25%	50.000000	35.000000	13.000000	0.000000	44.900000	2.140000
50%	122.000000	87.000000	29.000000	1.000000	48.500000	5.550000
75%	155.000000	111.000000	44.000000	2.000000	54.200000	7.290000
max	338.000000	250.000000	145.000000	16.000000	87.800000	82.100000

maxs = df.max()


```
maxs = df.max()
print(maxs)
```

```
Date        2017-04-29 9:55
ToVol                  338
SmVol                  250
MeVol                  145
LaVol                   16
Speed                 87.8
Occ.Rate              82.1
dtype: object
```

라벨을 정하기

속도 Speed를 라벨로 정하자

```
def get_score(v):
    if v < 20:
        score = 'Jam'
    elif v < 40:
        score = 'Slow'
    else :
        score = 'Normal'
    return score
```

```
[20] df["label"] = df["Speed"].apply(lambda v: get_score(v))
     df
```

	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label
0	2017-04-02 0:00	43	34	9	0	50.3	1.90	Normal
1	2017-04-02 0:05	45	32	13	0	58.9	1.84	Normal

df.head()

```
df.head()
```

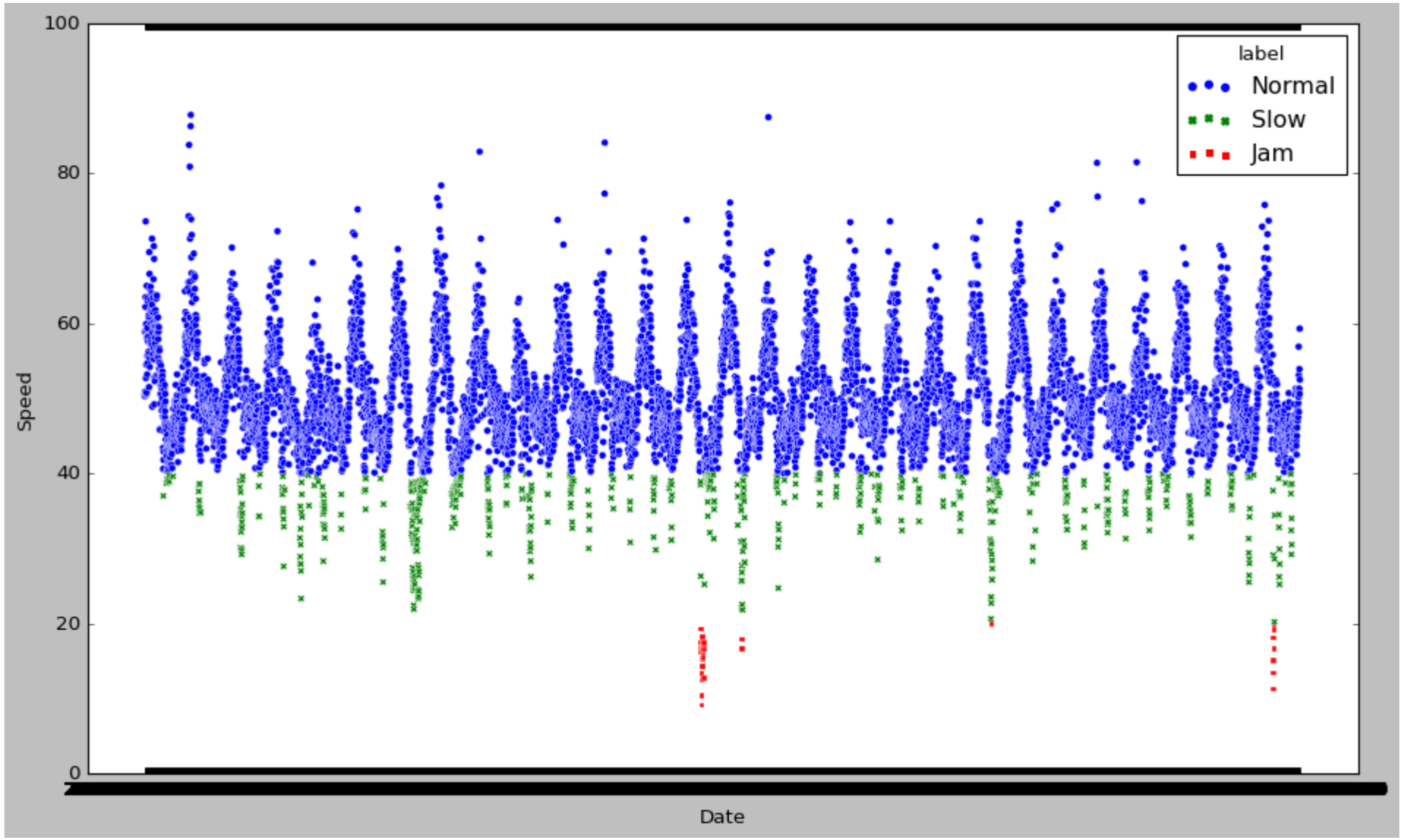
	Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label
0	2017-04-02 0:00	43	34	9	0	50.3	1.90	Normal
1	2017-04-02 0:05	45	32	13	0	58.9	1.84	Normal
2	2017-04-02 0:10	46	34	12	0	50.6	1.87	Normal
3	2017-04-02 0:15	45	36	9	0	50.9	1.72	Normal
4	2017-04-02 0:20	27	13	13	1	62.2	1.12	Normal

3. 데이터 가시화

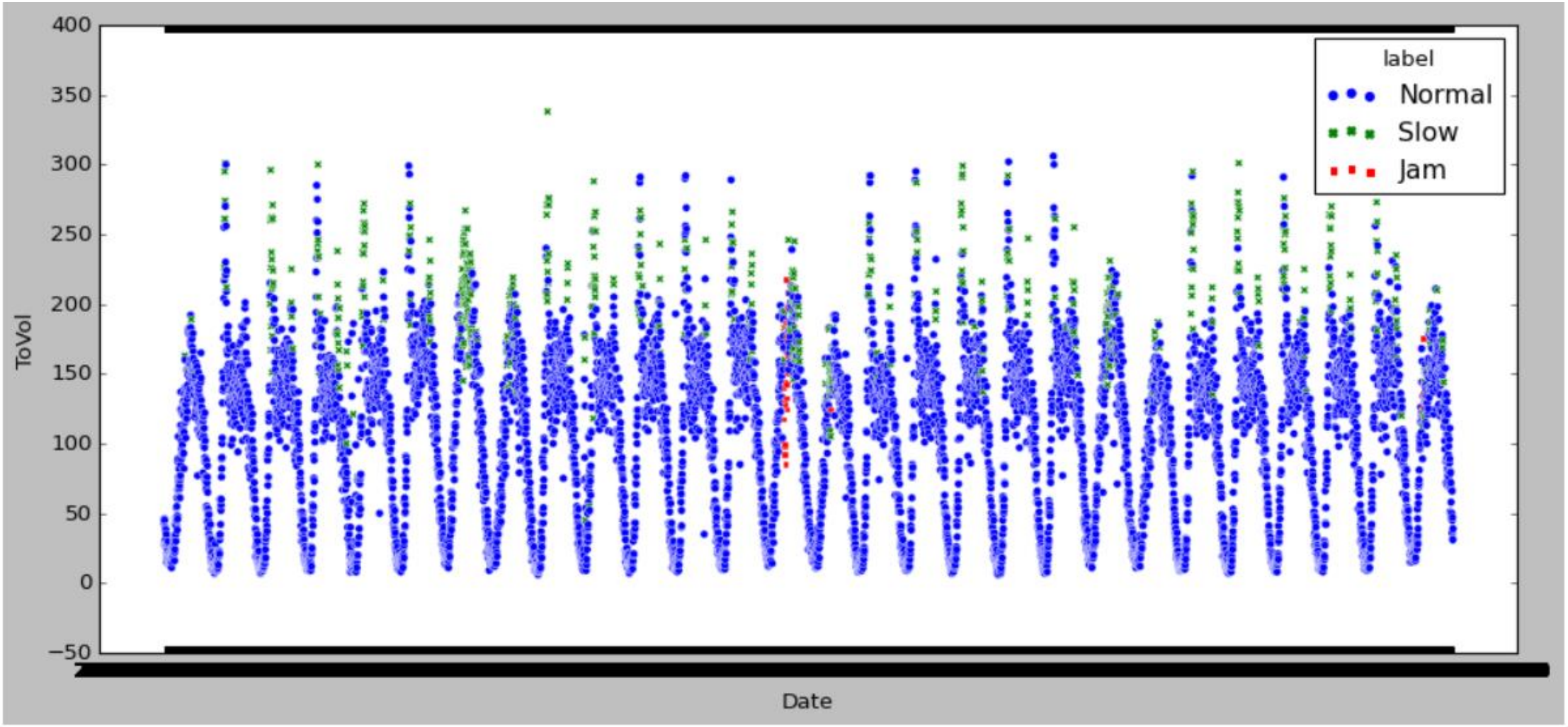
[23]
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use("classic")
```

[24]
```
import seaborn as sns
plt.figure(figsize=(14,8))
sns.scatterplot(data=df, x = 'Date', y = 'Speed', hue='label', style='label')
```

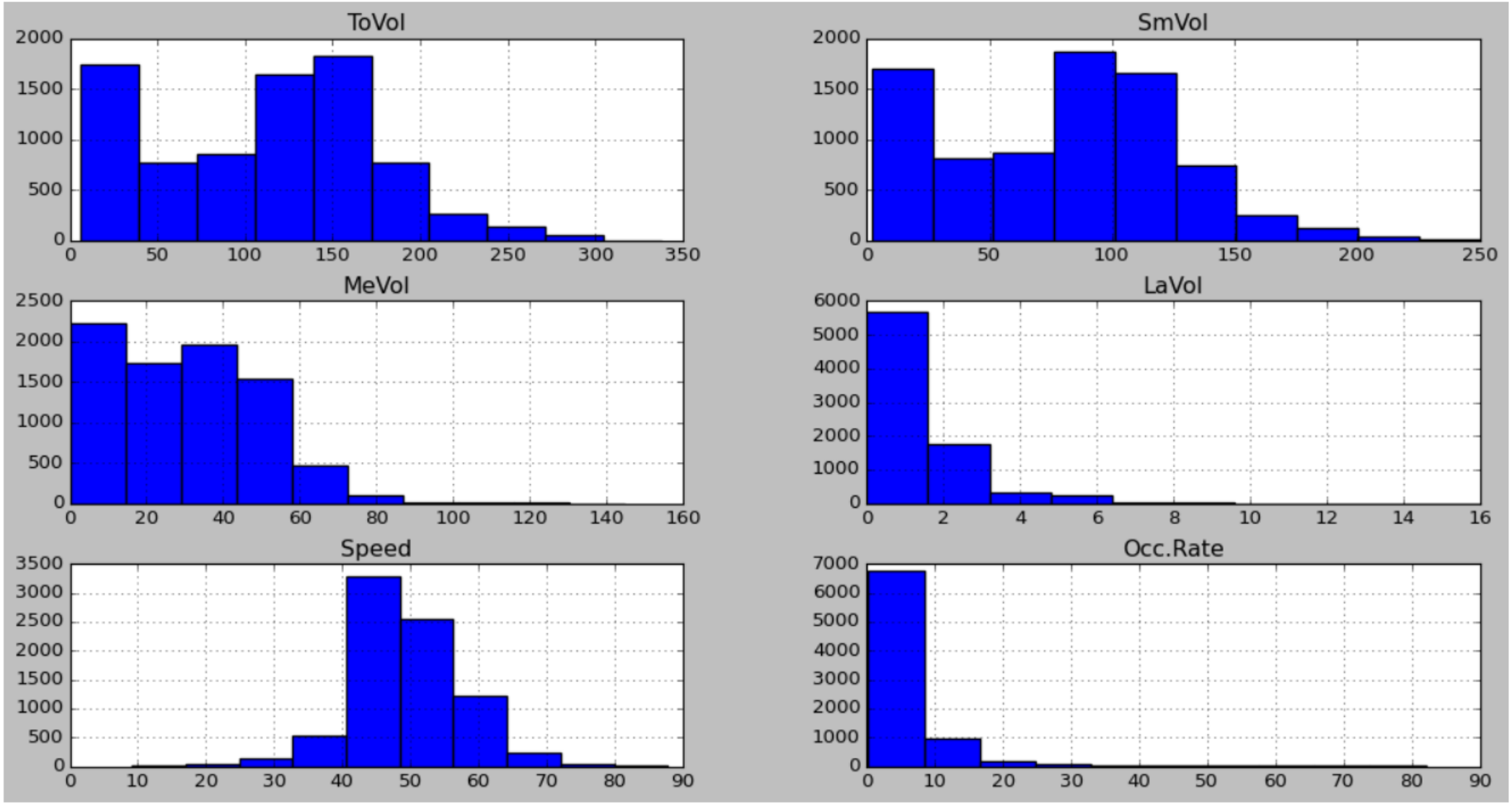
데이터 가시화



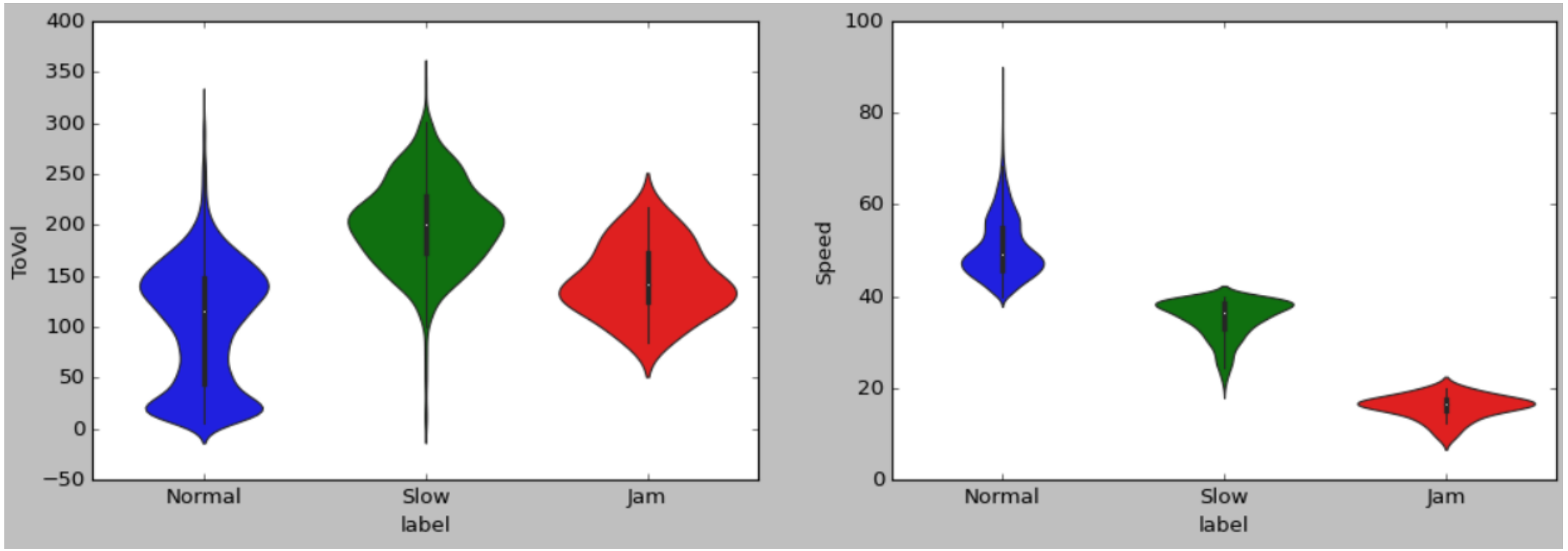
sns.scatterplot(data=df, x = 'Date', y = 'ToVol', hue='label')



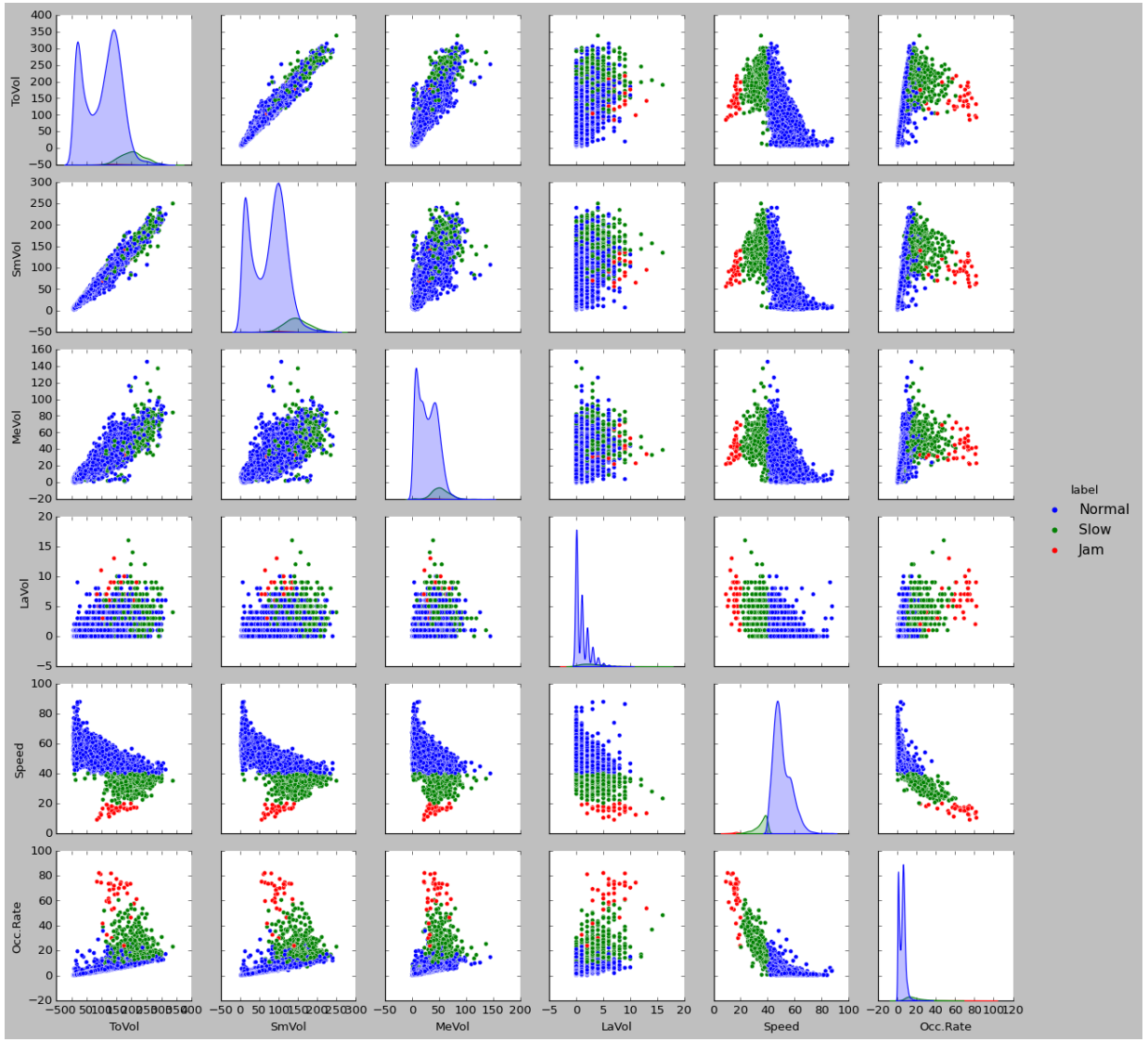
df.hist(edgecolor='black', linewidth=1.2)

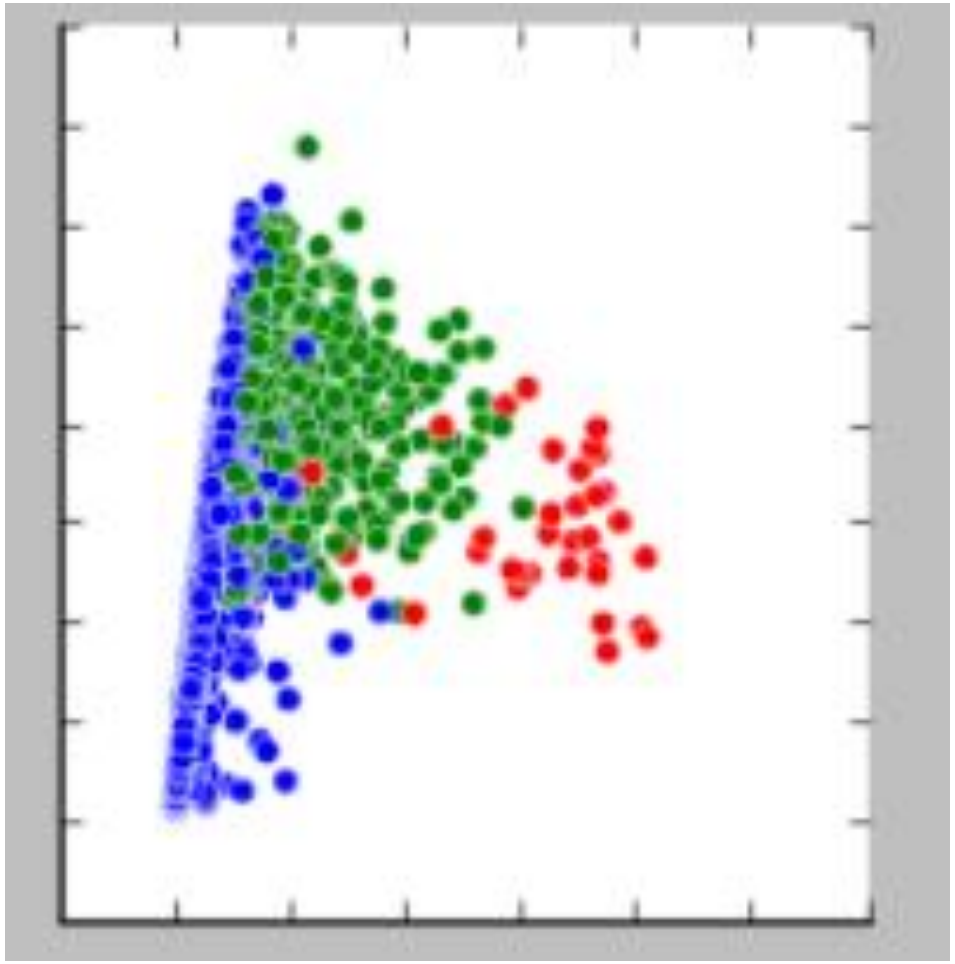


sns.violinplot



sns.pairplot(df,hue='label')





sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')

머신러닝을 위한 입력 데이터 만들자

1) 입력 X와 출력 y의 값을 정하기

```
df['label'].unique()
```

```
array(['Normal', 'Slow', 'Jam'], dtype=object)
```

```
#feature_cols = ['ToVol', 'SmVol', 'Speed', 'Occ.Rate']
#feature_cols = ['ToVol', 'SmVol', 'LaVol', 'MeVol']
feature_cols = ['ToVol','Occ.Rate']
target_col = 'label'
X = df[feature_cols]
y = df[target_col]
```

```
X.head()
```

	ToVol	Occ.Rate
0	43	1.90
1	45	1.84
2	46	1.87
3	45	1.72
4	27	1.12

```
y.head()
```

```
0    Normal
1    Normal
2    Normal
3    Normal
4    Normal
Name: label, dtype: object
```

2) 출력용 라벨을 머신러닝

텍스트를 숫자로 바꾸자

In [34]:
```
class_dic = {'Jam':0, 'Slow':1, 'Normal':2}
y_ohc = y.apply(lambda z: class_dic[z])
```

In [35]:
```
y_ohc.head()
```

Out [35] :
```
0    2
1    2
2    2
3    2
4    2
Name: label, dtype: int64
```

데이터를 훈련과 테스트로 나누자

3) 데이터를 훈련과 테스트로 나누자

- (실전) 데이터를 validation을 포함해서 나눌수 있다.
- (해보기) 전체 데이터를 train : validation : test = 0.6: 0.2: 0.2 로 나누어라

```
from sklearn.model_selection import train_test_split,  ShuffleSplit, learning_curve
#from sklearn.model_selection import learning_curve, train_test_split, KFold,  ShuffleSplit

X_train, X_test, y_train, y_test = train_test_split(X, y_ohc, test_size=0.20, random_state=30)
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape,  y_test.shape)
```

```
(6451, 2) (6451,)
(1613, 2) (1613,)
```

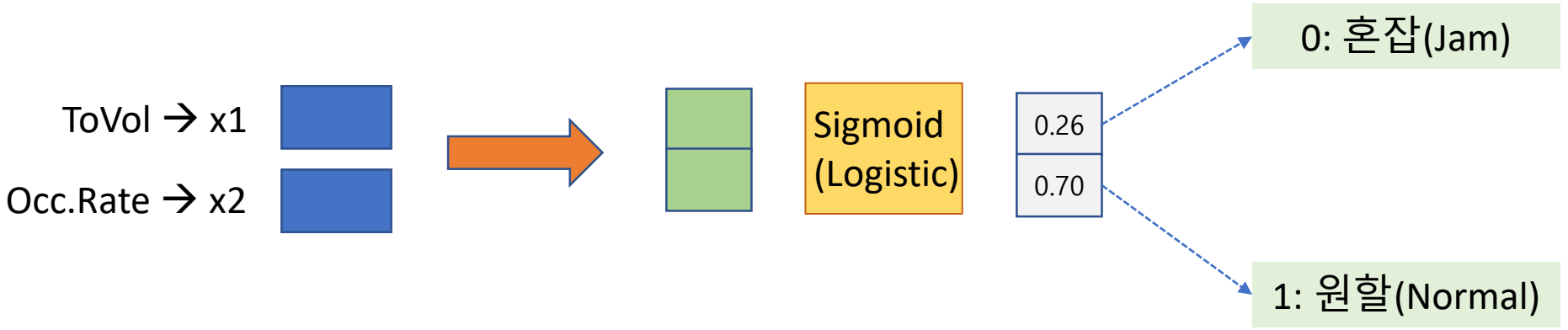
데이터 정규화


```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5. sklearn을 이용하여 머신러닝 모델을 만들자


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
```

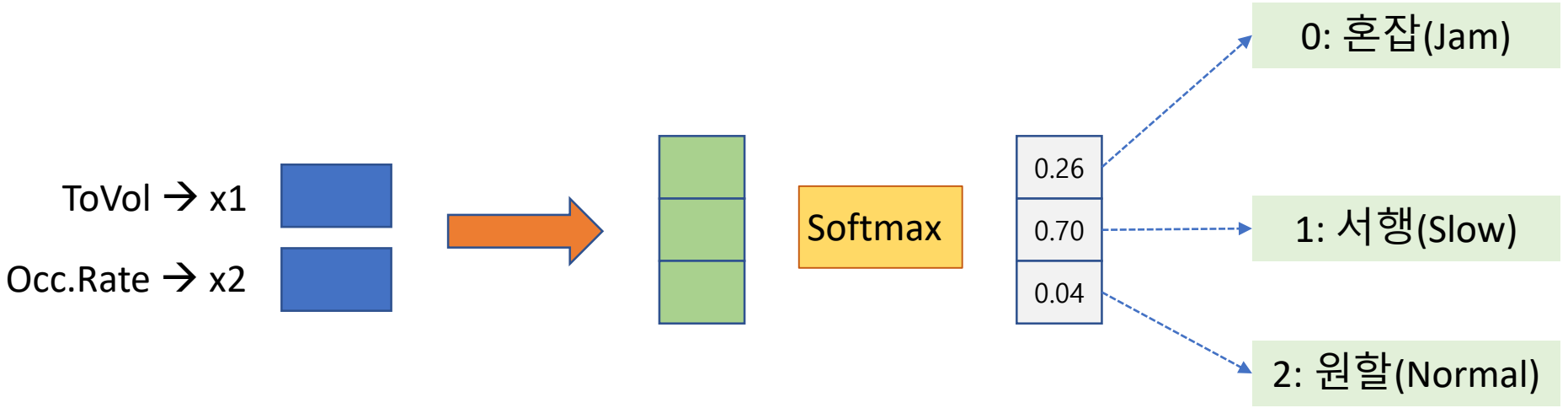
라벨이 2개 일경우

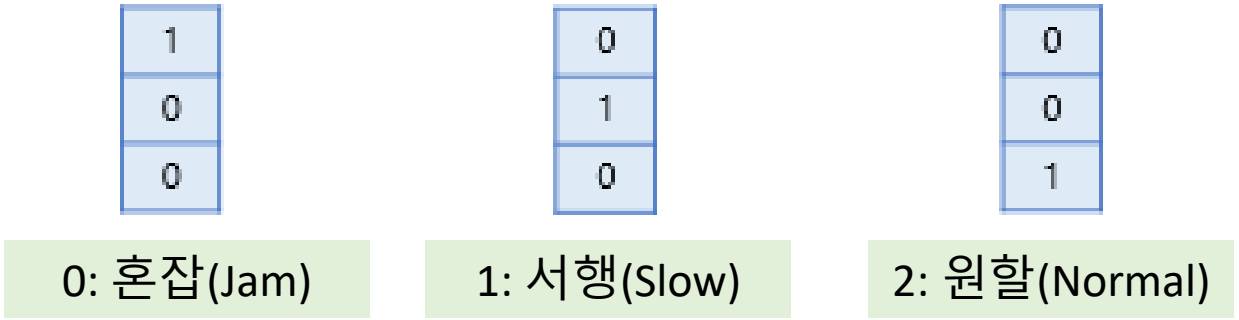

ToVol → x1
Occ.Rate → x2
Sigmoid
(Logistic)
0.26
0.70
0: 혼잡(Jam)
1: 원할(Normal)

Logistic Regression

$$cost(W) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}log(p^{(i)}) + (1-y^{(i)})log(1-p^{(i)})]$$

라벨이 3개 일경우

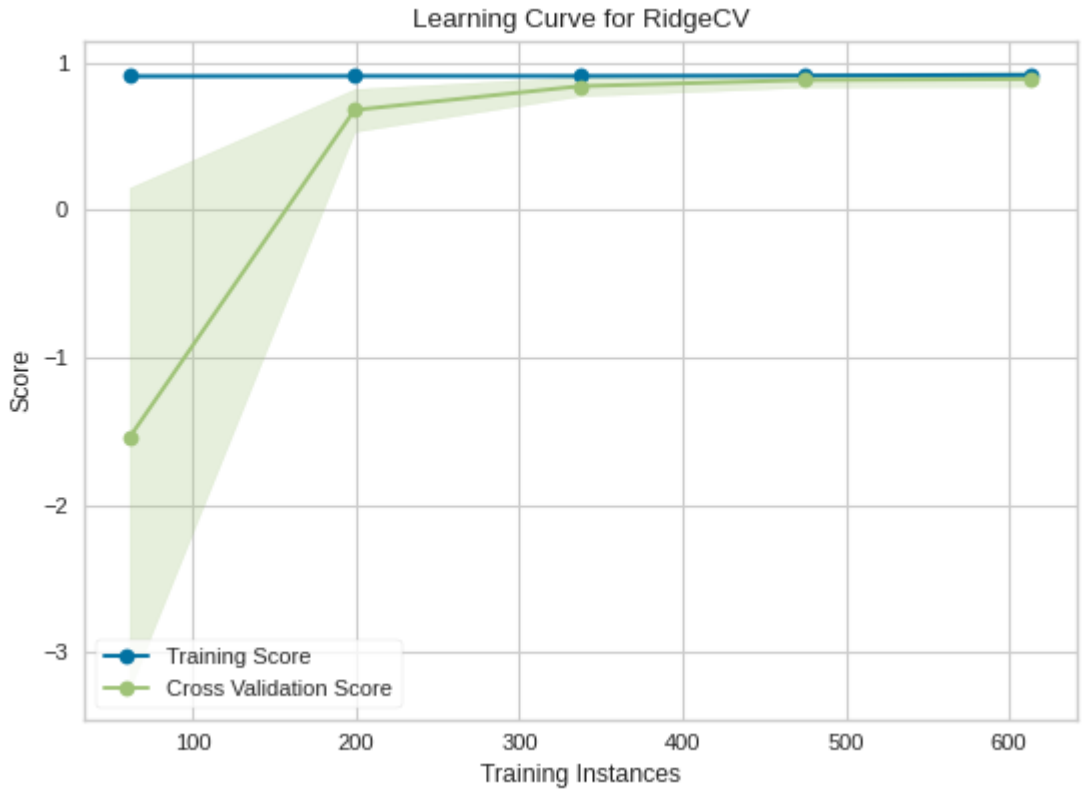

ToVol → x1
Occ.Rate → x2
Softmax
0.26
0.70
0.04
0: 혼잡(Jam)
1: 서행(Slow)
2: 원할(Normal)


1
0
0
0: 혼잡(Jam)
0
1
0
1: 서행(Slow)
0
0
1
2: 원할(Normal)

Softmax

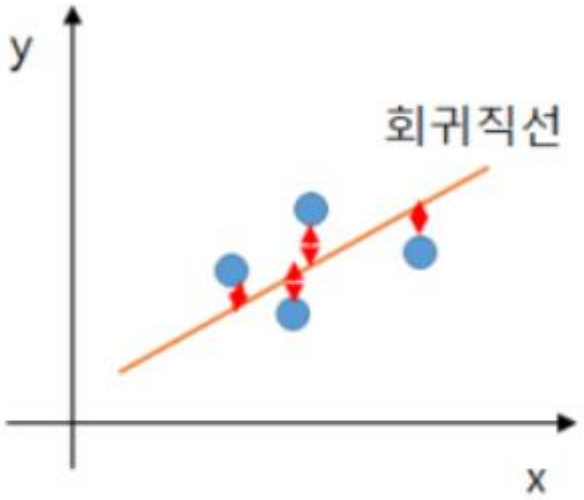
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

sklearn learning curve : Regression에서 R2 스코어 사용

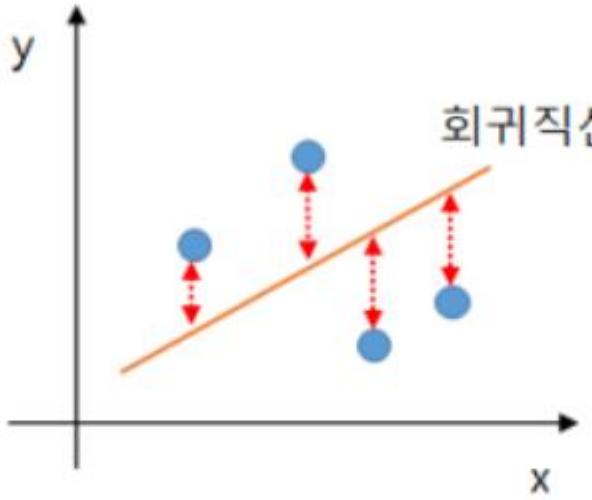


회귀의 평가지표(default): 설명력(결정계수, R2)

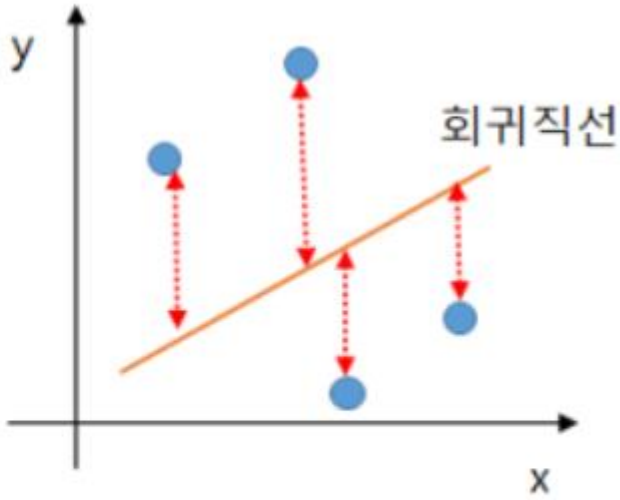
$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



R^2가 1에 가까운 경우



R^2가 0.5에 가까운 경우



R^2가 0에 가까운 경우

1) 로지스틱 회귀

In [41]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [42]:
```
m_lr = LogisticRegression()
m_lr.fit(X_train,y_train)
```

Out [42] : LogisticRegression()

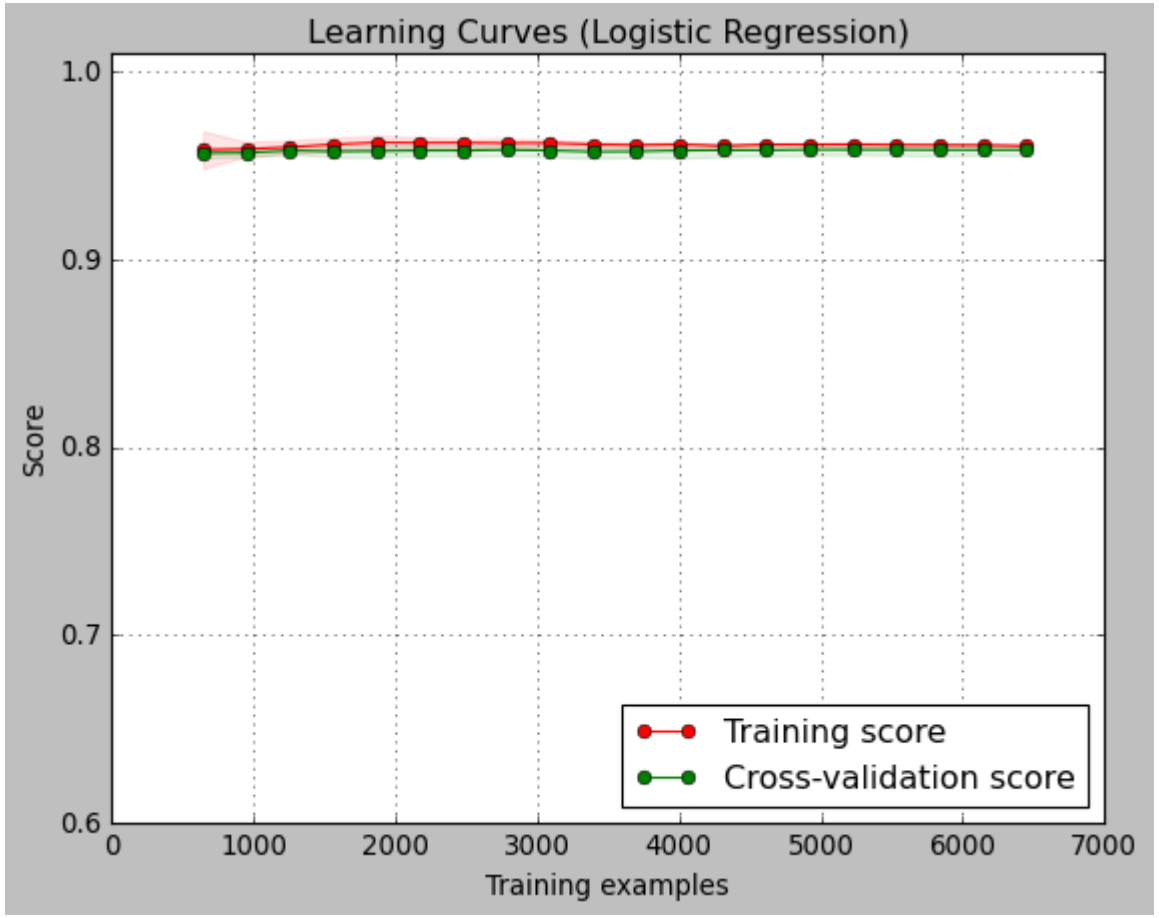
In [43]:
```
pred = m_lr.predict(X_test)

acc_lr = metrics.accuracy_score(pred,y_test)
print('The accuracy of the Logistic Regression is', acc_lr)
```

```
The accuracy of the Logistic Regression is 0.9603223806571606
```

In [44]:
```
title = "Learning Curves (Logistic Regression)"
cv = ShuffleSplit(n_splits=4, test_size=0.2, random_state=0)
```

1) 로지스틱 회귀


```
#import myUtil as myutil
plot_ml_curve(m_lr, title, X, y, ylim=(0.6, 1.01), cv=cv)
```



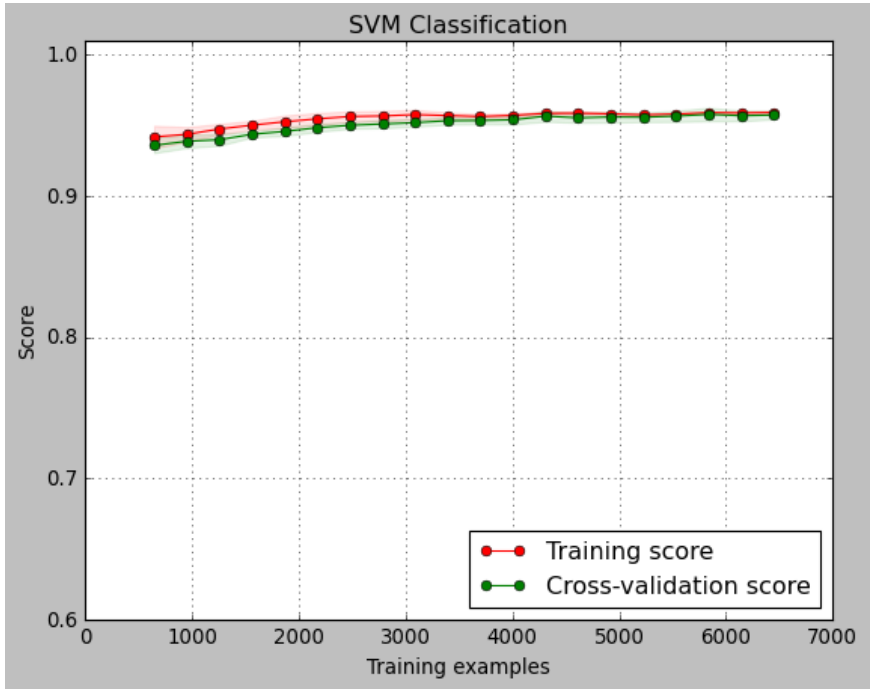
2) Support Vector Machine (SVM)

```
sv = svm.SVC()
sv.fit(X_train,y_train)
pred = sv.predict(X_test)
acc_svm = metrics.accuracy_score(pred,y_test)
print('The accuracy of the SVM is:', acc_svm)
```

The accuracy of the SVM is: 0.9671419714817111

```
%%time
title = "SVM Classification"
cv = ShuffleSplit(n_splits=4, test_size=0.2, random_state=0)
plot_ml_curve(sv, title, X, y, ylim=(0.6, 1.01), cv=cv )
```

Wall time: 34.6 s

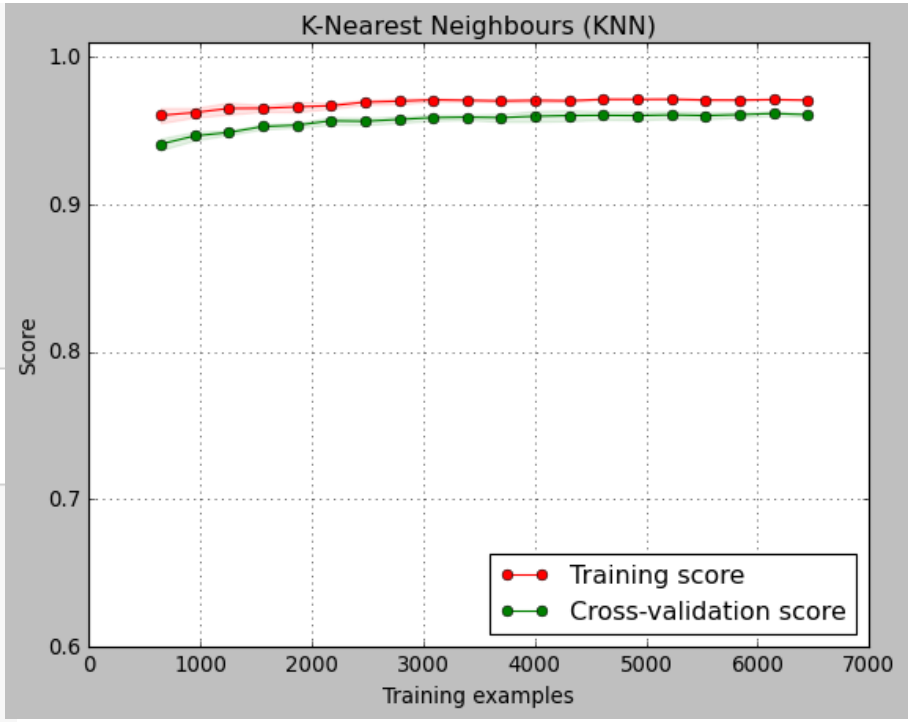


3) K-Nearest Neighbours

```
knc = KNeighborsClassifier(n_neighbors=6)
knc.fit(X_train,y_train)
pred = knc.predict(X_test)
acc_knn = metrics.accuracy_score(pred,y_test)
print('The accuracy of the KNN is', acc_knn)
```

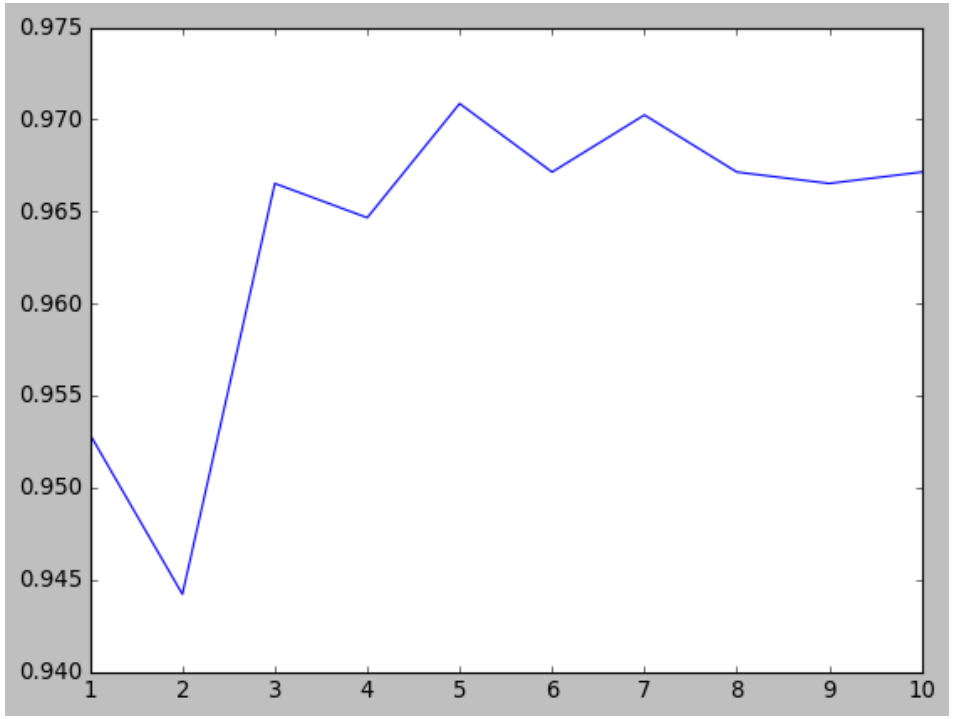
```
The accuracy of the KNN is 0.9671419714817111
```

```
title = "K-Nearest Neighbours (KNN)"
cv = ShuffleSplit(n_splits=4, test_size=0.2, random_state=0)

plot_ml_curve(knc, title, X, y, ylim=(0.6, 1.01), cv=cv, n_jobs=4)
```



3.1 최적의 k-NN의 정확도 확인


```
a_index = list(range(1,11))
a = pd.Series()
x = [1,2,3,4,5,6,7,8,9,10]
for i in list(range(1,11)):
    kcs = KNeighborsClassifier(n_neighbors=i)
    kcs.fit(X_train,y_train)
    y_pred = kcs.predict(X_test)
    a=a.append(pd.Series(
        metrics.accuracy_score(y_pred,y_test)))

plt.plot(a_index, a)
plt.xticks(x)
```



4) 랜덤포레스트 (RandomForestClassifier)

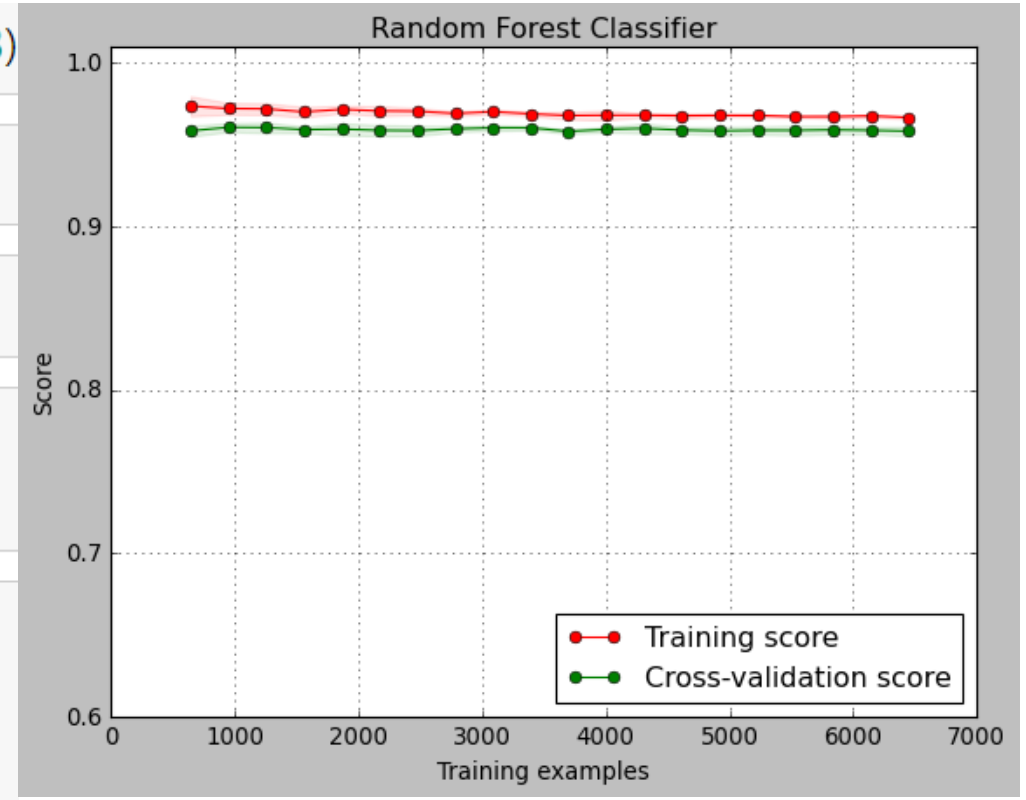

```
m_rf = RandomForestClassifier(n_estimators=100, max_depth = 3)
```

```
m_rf.fit(X_train, y_train)
```

```
pred = m_rf.predict(X_test)
```

```
acc_rf = metrics.accuracy_score(pred,y_test)
print('The accuracy of the RFC is:', acc_rf)
```

```
title = "Random Forest Classifier"
cv = ShuffleSplit(n_splits=4, test_size=0.2, random_state=0)
plot_ml_curve(m_rf, title, X, y, ylim=(0.6, 1.01), cv=cv)
```



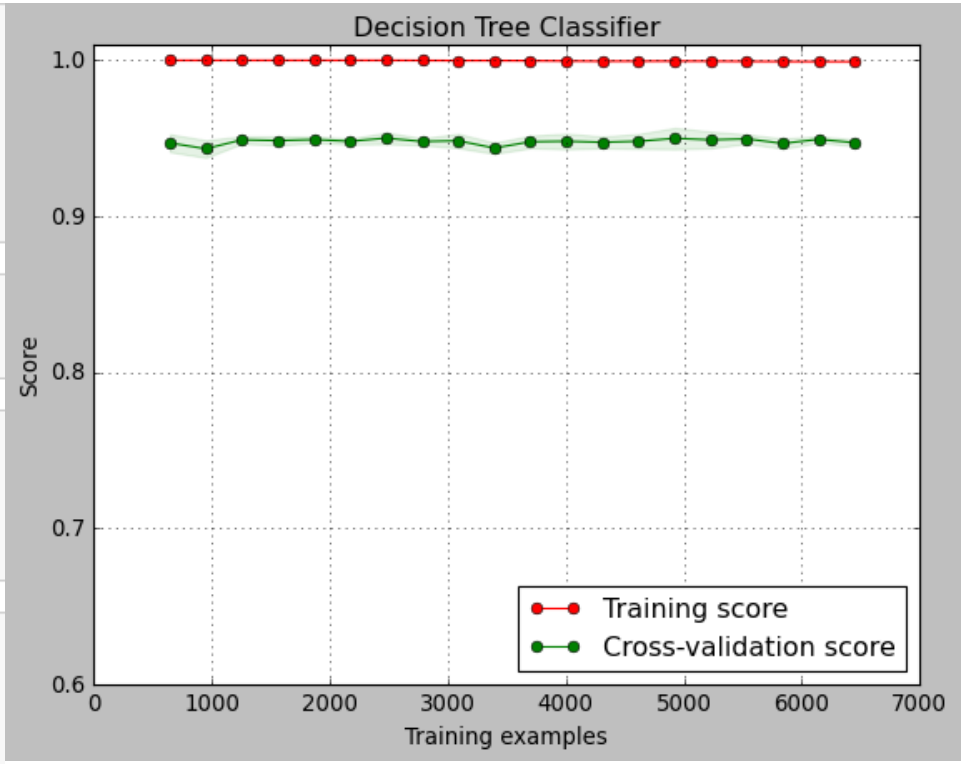
5) 의사결정나무 (DecisionTreeClassifier)


```
m_tree = DecisionTreeClassifier()

m_tree.fit(X_train, y_train)
```

```
prd = m_tree.predict(X_test)
```

```
acc_dt = metrics.accuracy_score(pred,y_test)
print('The accuracy of the Decision Tree is:', acc_dt)
```

```
title = "Decision Tree Classifier"
cv = ShuffleSplit(n_splits=4, test_size=0.2, random_state=0)

plot_ml_curve(m_tree, title, X, y, ylim=(0.6, 1.01), cv=cv)
```



기계학습 성능 비교


```
models = pd.DataFrame({
    'Model': ['Logistic Regression',  'Support Vector Machines','RandomForest',
              'K-Nearest Neighbours', 'Decision Tree'],
    'Score': [acc_lr, acc_svm, acc_rf, acc_knn, acc_dt]})
models.sort_values(by='Score', ascending=False)
```

	Model	Score
1	Support Vector Machines	0.967142
3	K-Nearest Neighbours	0.967142
0	Logistic Regression	0.960322
2	RandomForest	0.960322
4	Decision Tree	0.001860

2022 TRUST KISTI

Korea Institute of Science
and Technology Information