KiSTi 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2023

인공지능특론 (Advanced Artificial Intelligence)

12주 인공신경망 분류(Classification)

Hongsuk Yi KISTI

교통 데이터 소개

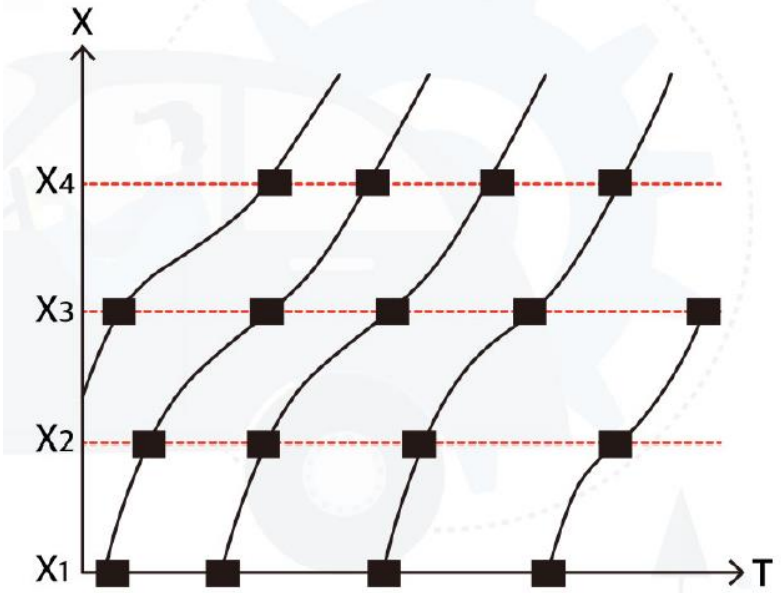
교통 데이터 관측

❖ 고정위치 관측

- ✓도로상의 고정된 위치에서 지나가는 차량들을 관측
- ✓고정된 시각에 일정 구간의 교통 상태를 관측
- ✓항공 사진 촬영

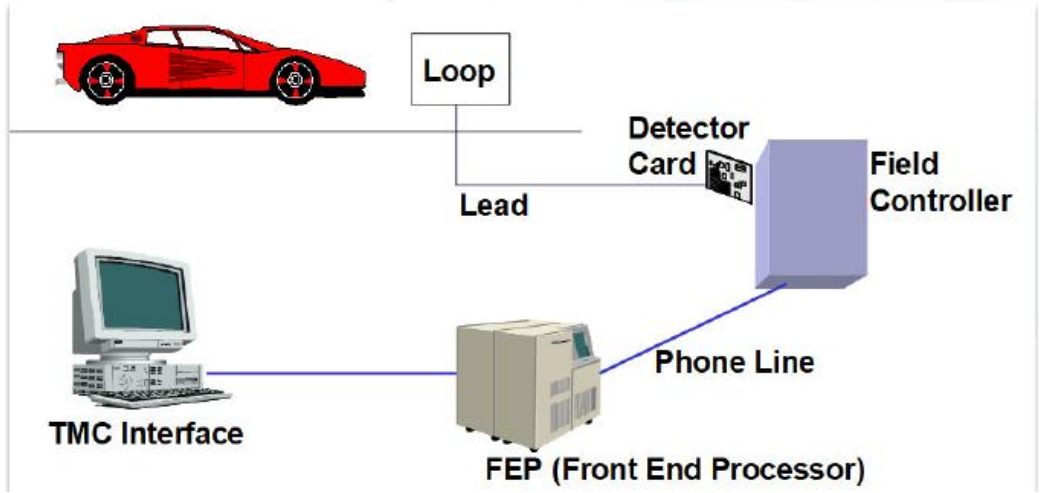
❖ 관측가능한 데이터는

- ✓통과 차량수, 차량 사이의 시간 간격, 차량 속도



루프 검지기

- 교통량 산정
- 도로 용량 분석
- 차량 통과 속도 감시



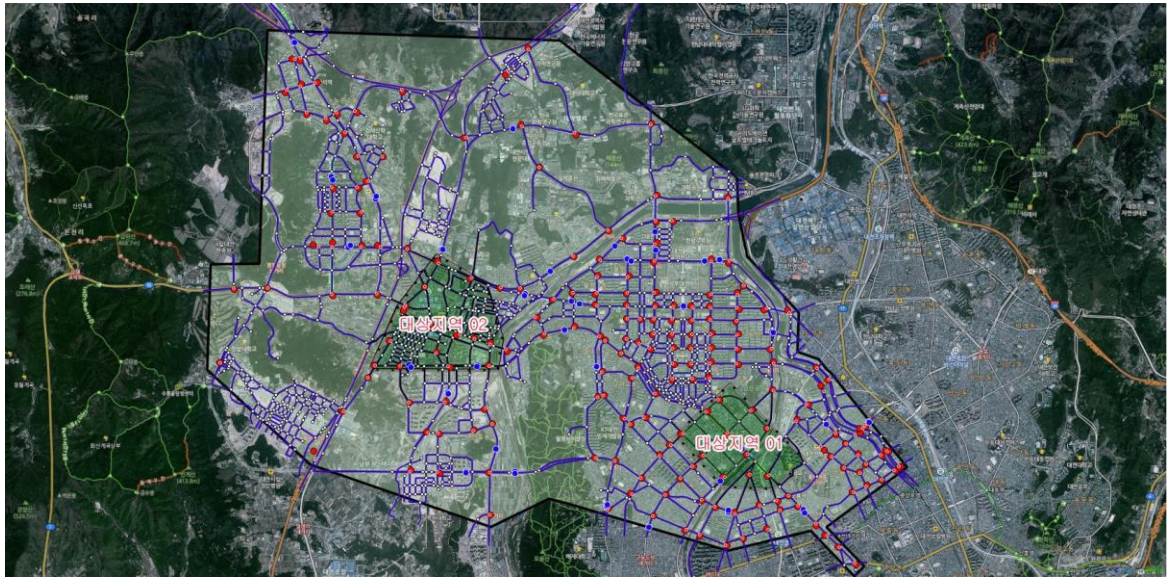
교통 관련 데이터 다운로드

- 국가교통데이터 오프마켓
  - ✓ https://www.bigdata-transportation.kr/

- 대전교통정보센터
  - ✓ http://traffic.daejeon.go.kr/mainFront/atmsMain.do

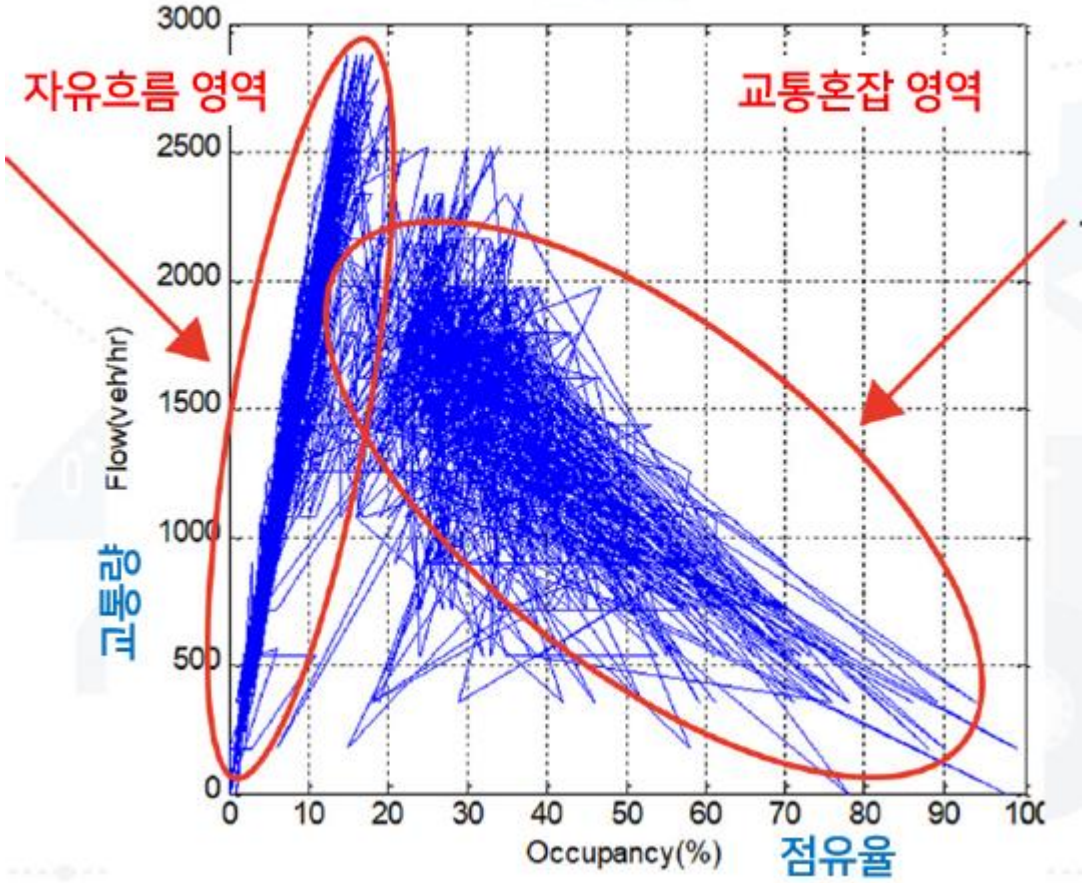
VDS 운행 원시 이력 데이터

- 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



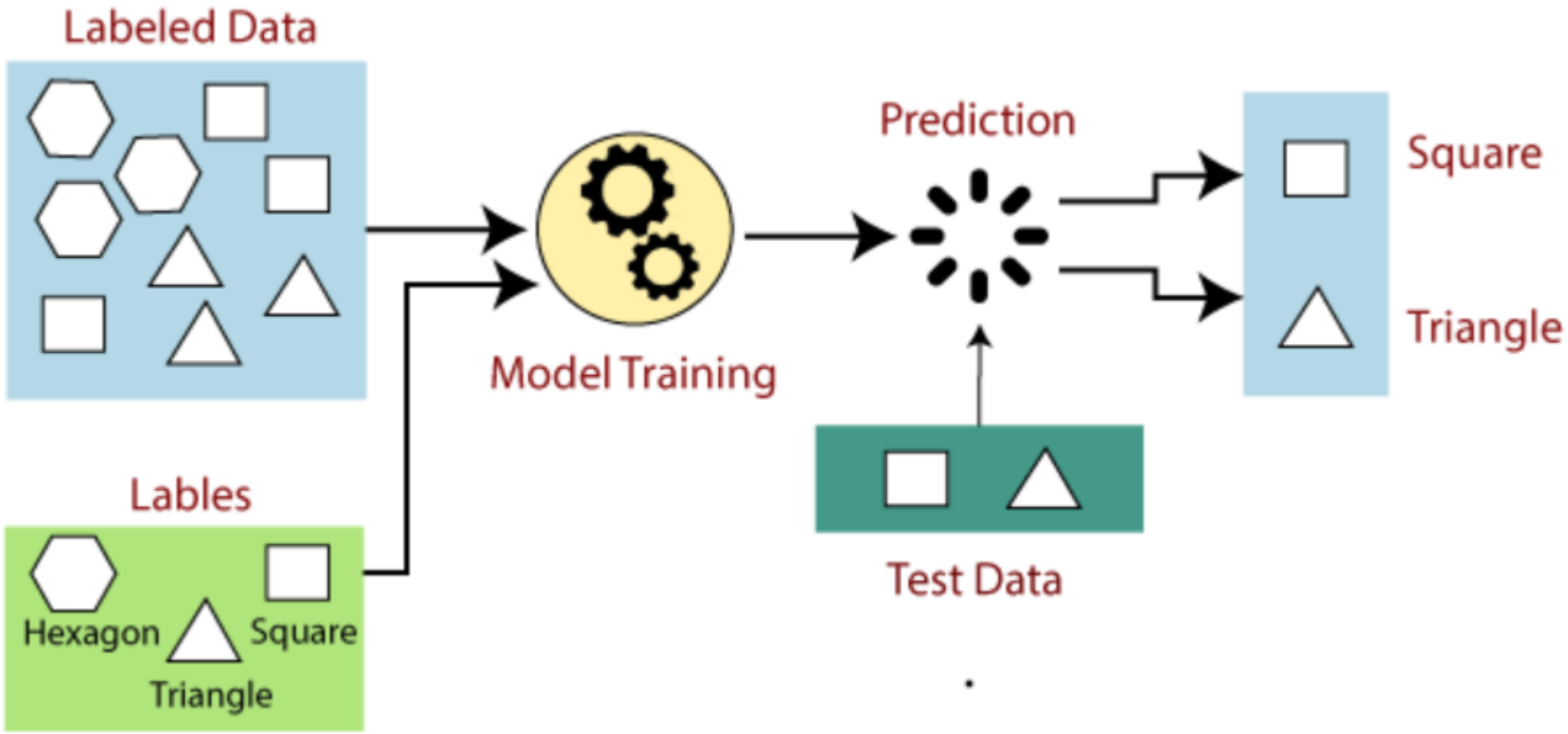
A	B	C	D	E	F	G
Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate
2017-04-02 0:00	43	34	9	0	50.3	1.9
2017-04-02 0:05	45	32	13	0	58.9	1.84
2017-04-02 0:10	46	34	12	0	50.6	1.87
2017-04-02 0:15	45	36	9	0	50.9	1.72
2017-04-02 0:20	27	13	13	1	62.2	1.12
2017-04-02 0:25	30	15	15	0	63.5	1.17
2017-04-02 0:30	27	14	12	1	73.6	0.99
2017-04-02 0:35	40	25	15	0	52.7	1.76

점유률과 교통량 관계


자유흐름 영역
교통혼잡 영역
3000
2500
2000
1500
1000
500
0
Flow(veh/hr)
교통량
0
10
20
30
40
50
60
70
80
90
100
Occupancy(%)
점유율

딥러닝 기반 교통흐름 분류

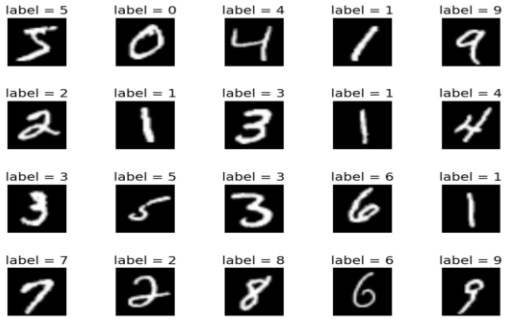
How Supervised Learning Works?



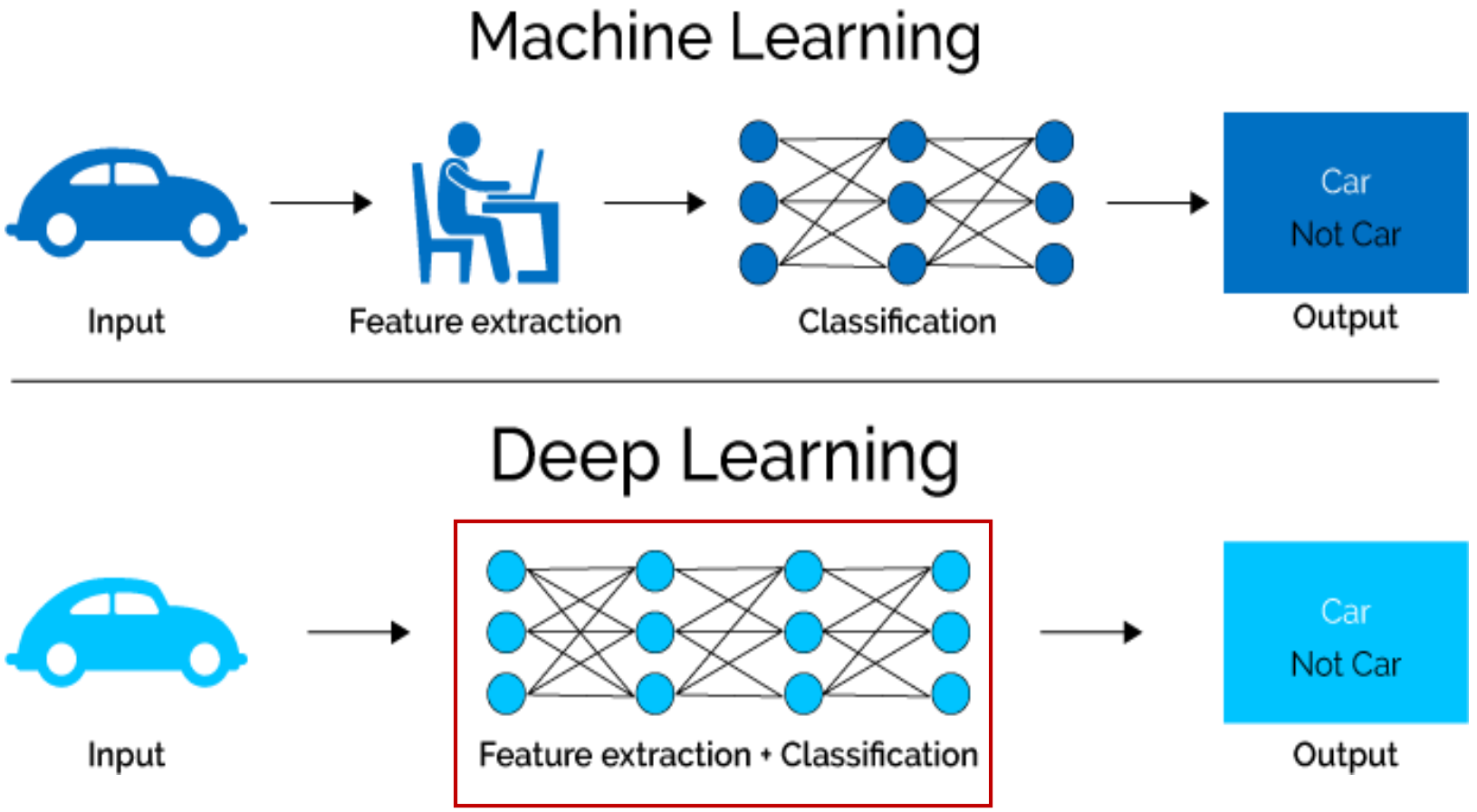
(source) https://www.javatpoint.com/supervised-machine-learning

지도학습과 레이블(Label)

MNIST



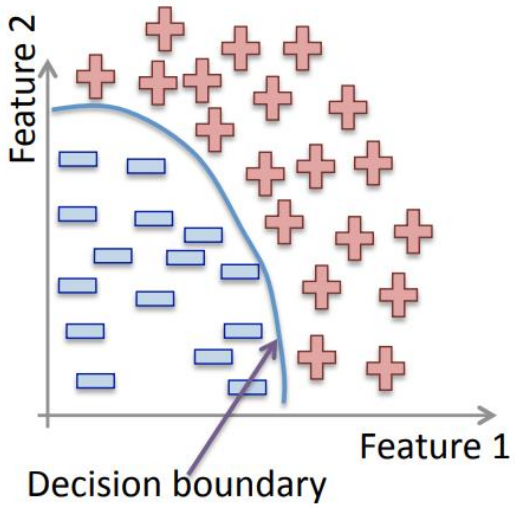
인공지능의 분류

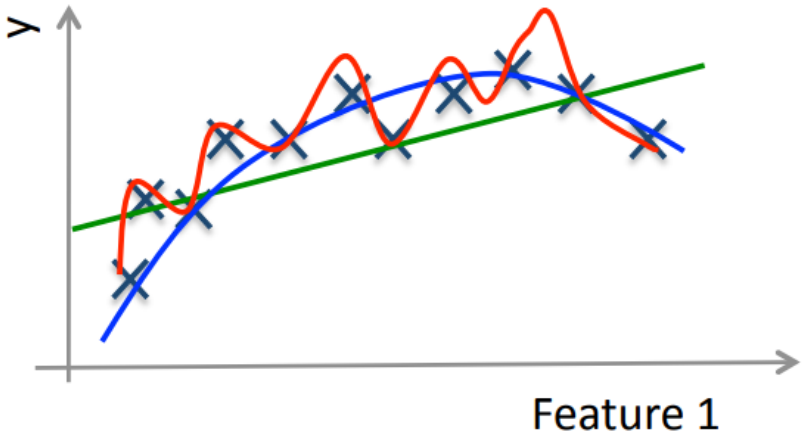


자료: https://www.xenonstack.com/blog/data-science/log-analytics-deep-machine-learning-ai/

지도학습(Supervised Learning)

- 지도학습 알고리즘을 적용하기위해서는 레이블이 있어야 함
  - ✓ 누가 레이블을 만드나?
- 분류 문제 다루기
  - ✓분류의 정확도
- 회귀 문제 다루기
  - ✓실측치와 출력치의 오차를 최소화





학습할 내용

- Deep Neural Network을 이용한 분류
  - ✓VDS 데이터 분류 정확도
  - ✓VDS 라벨 최적화를 통한 분류 정확도 향상
- 회귀를 통한 예측 문제 다루기
  - ✓선형 함수 회귀
  - ✓비선형 함수 회귀
  - ✓활성함수 종류 다루기
- 시계열 데이터를 이용한 예측 문제 해결
  - ✓RNN 이해
  - ✓LSTM 이해
  - ✓VDS 데이터를 이용한 SimpleRNN 적용 및 예측하기
- 교통 데이터를 활용한 Many-to-One, Many-to-Many 응용문제

도로의 교통 흐름을 레이블링

KISTI SINCE 1962

❖ VDS 데이터의 속도로 레이블링 해보자

✓2개 클래스
- 혼잡(Jam) : 0~30km
- 서행(Slow): 30km 이상

✓3개 클래스
- 혼잡(Jam) : 0~25km
- 서행(Slow): 25~50km
- 원할(Fast): 50~

✓4개 클래스

Traffic-jam Level	Speed Performance Index	Traffic State	Traffic Context
1	[0, 0.25]	Traffic jam	The average speed is the lowest; the road traffic state is very poor.
2	(0.25, 0.50]	Slow	The average speed is low; the road traffic state is poor.
3	(0.5, 0.75]	Moderate	The average speed is moderate; the road traffic state is a little congested.
4	(0.75, 0.1]	Fast	The average speed is high; the road traffic state is good.

(Source): https://www.researchgate.net/figure/The-criterion-of-traffic-jam-level-and-the-speed-performance-index_tbl1_322834098

2개의 클래스로 레이블링

KISTI SINCE 1962

2개의 클래스로 레이블링

In [55]:
```
num_classes = 2
class_labels= ['Jam', 'Normal']
```

In [17]:
```
def get_score(speed):
    if speed < 45:
        label = 'Jam'
    else :
        label = 'Normal'
    return label
```

3개의 클래스로 레이블링

```
def get_score(speed):
    if speed < 45:
        label = 'Jam'
    elif speed < 50:
        label = 'Slow'
    else :
        label = 'Normal'
    return label
```

2개의 클래스로 레이블링

KISTI SINCE 1962

In [20]:
```
df["label_speed"] = df["Speed"].apply(lambda spd: get_score(spd))
df.head()
```

Out[20]:

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

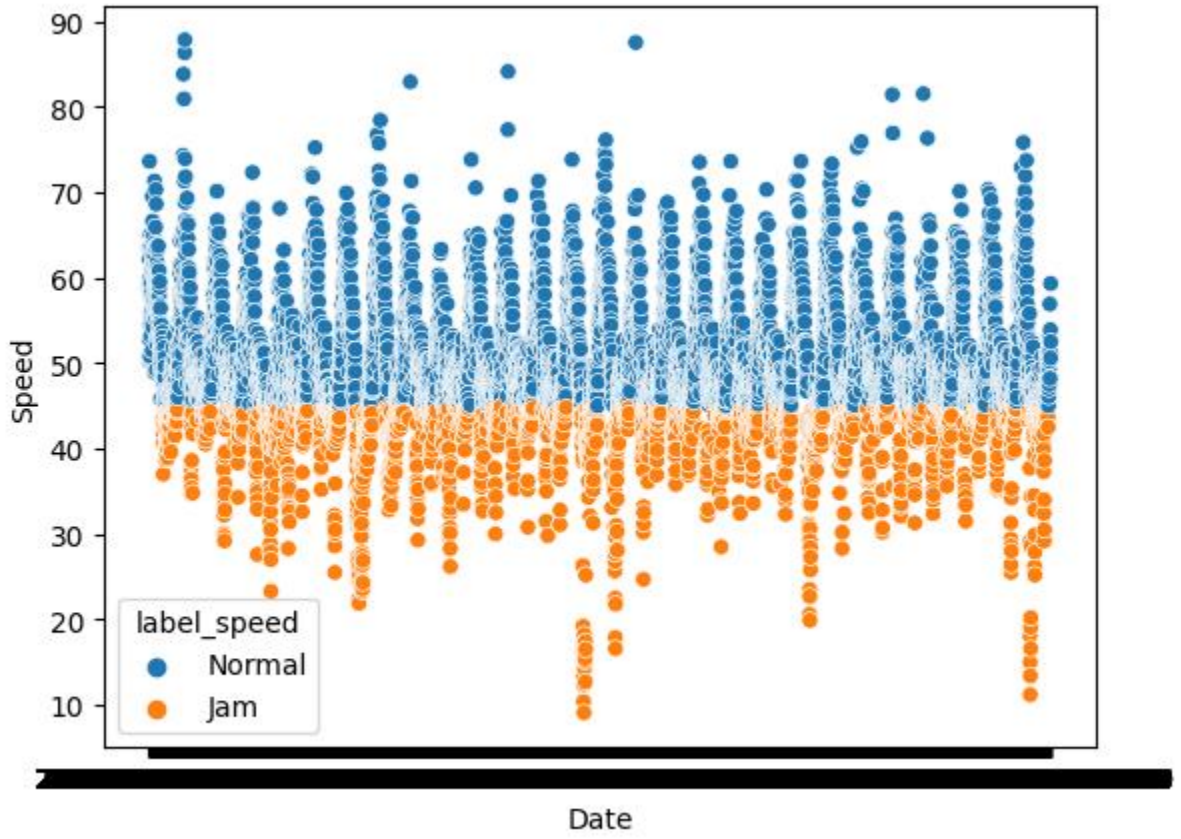
In [21]:
```
df["label_speed"].unique()
```

Out[21]:
```
array(['Normal', 'Jam'], dtype=object)
```

Seaborn으로 통계 분석 및 가시화

In [22]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

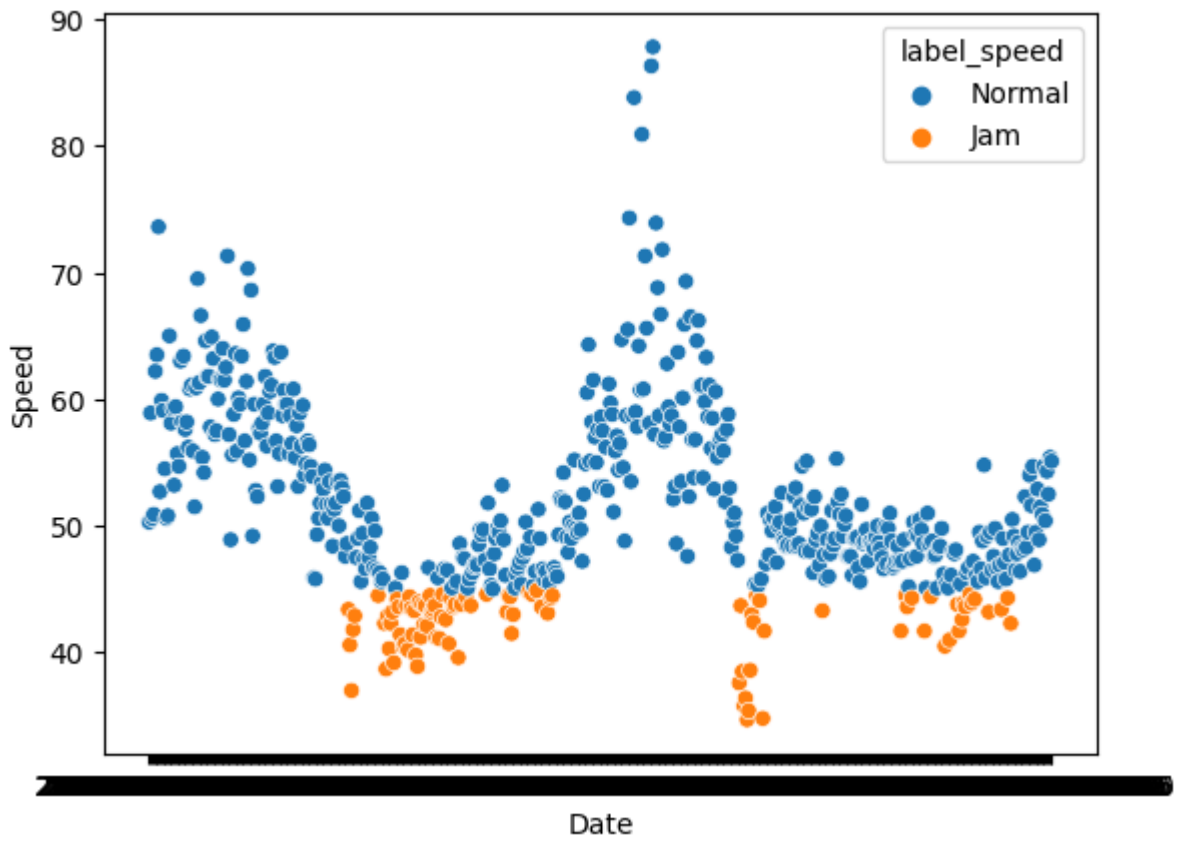
```
sns.scatterplot(data=df, x = 'Date', y = 'Speed', hue='label_speed')
```



속도 가시화

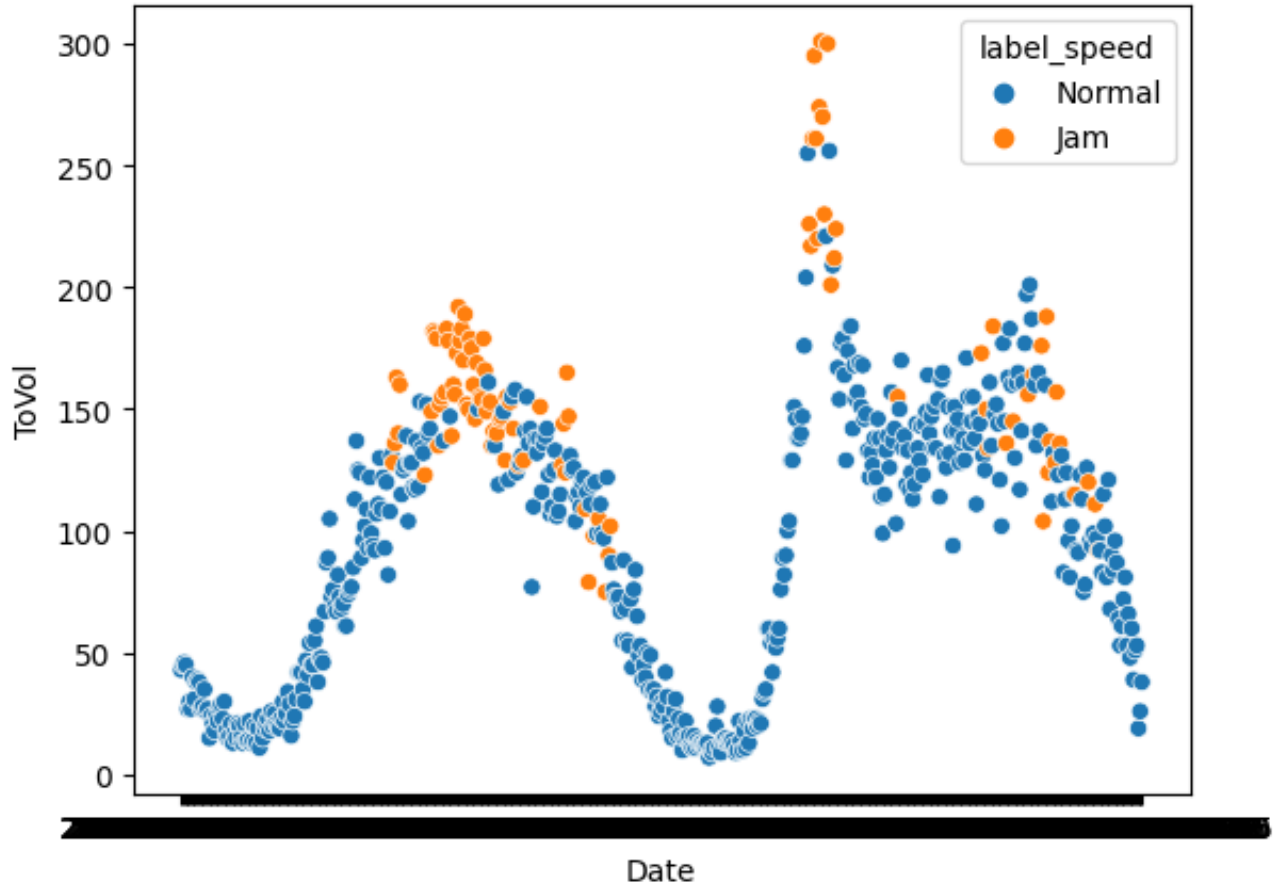
KISTI SINCE 1962

In [28]:
```
sns.scatterplot(data=df[:576], x = 'Date', y = 'Speed', hue='label_speed')
```



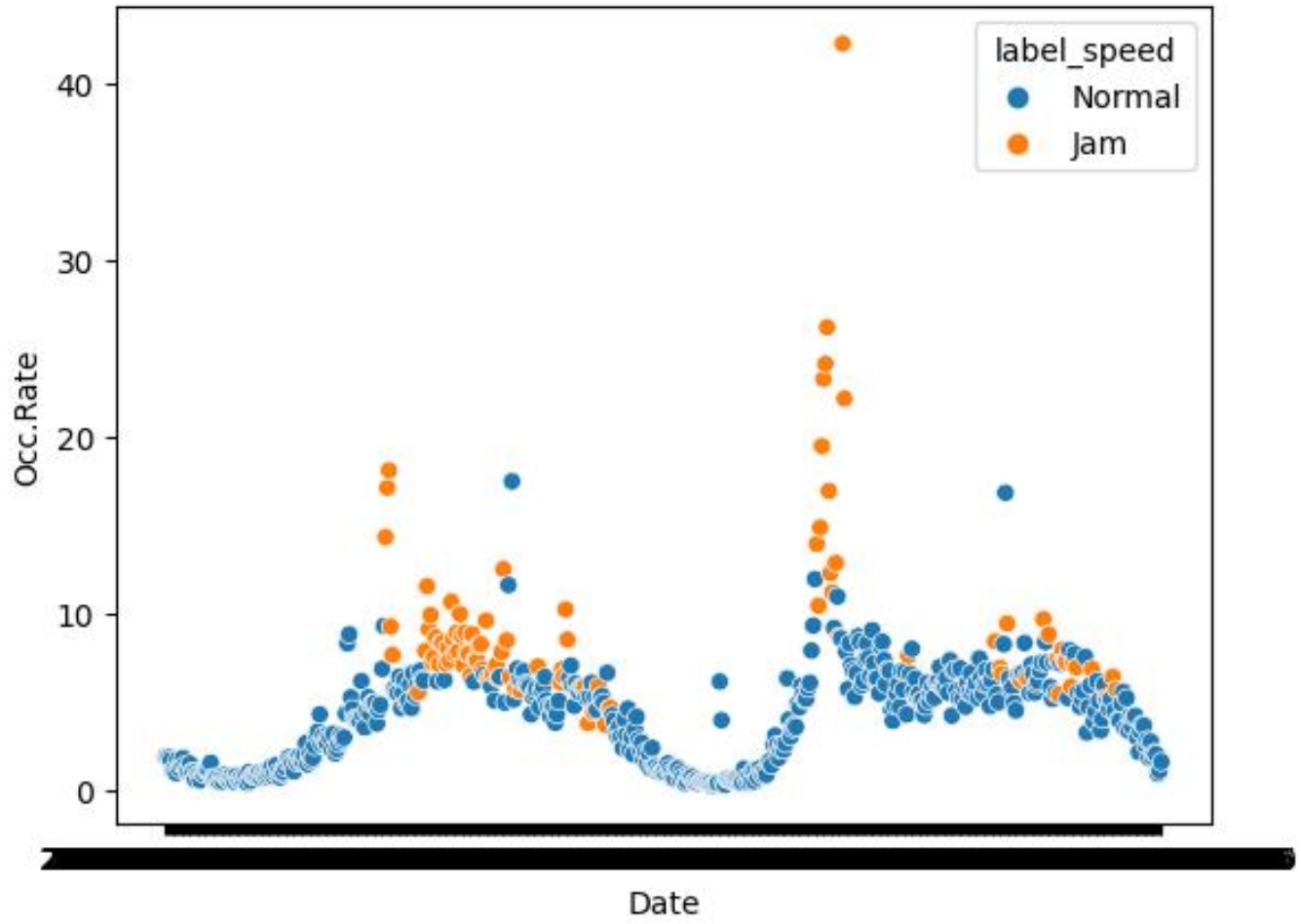
교통량 가시화

```
sns.scatterplot(data=df[:576], x = 'Date', y = 'ToVol', hue='label_speed')
```



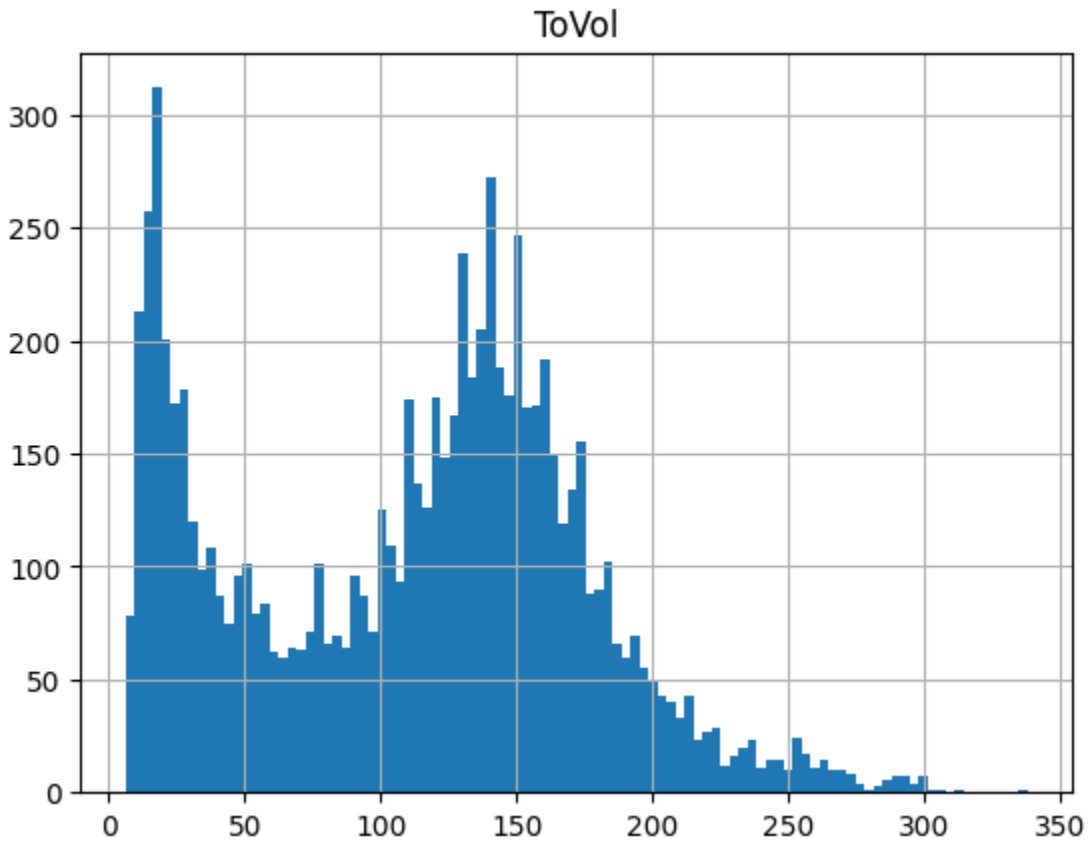
점유률 가시화

```
sns.scatterplot(data=df[:576], x = 'Date', y = 'Occ.Rate', hue='label_speed')
```



히스토그램 : 교통량

```
In [36]: df.hist('ToVol',bins=100)
```



상관관계

In [53]:
```
df2=df.drop(['LaVol','MeVol','SmVol'],axis=1)
```

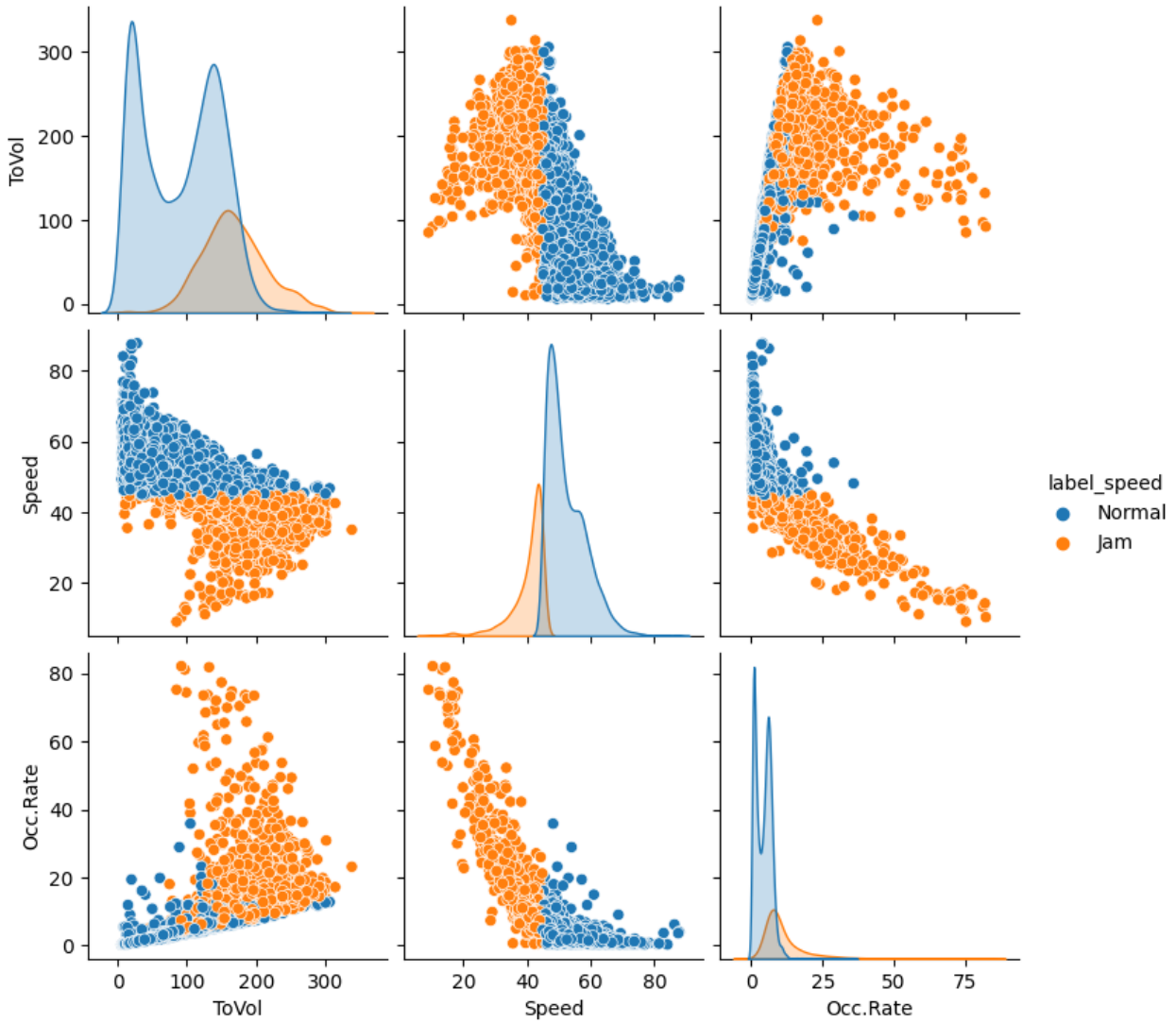
In [55]:
```
df2.head()
```

Out[55]:

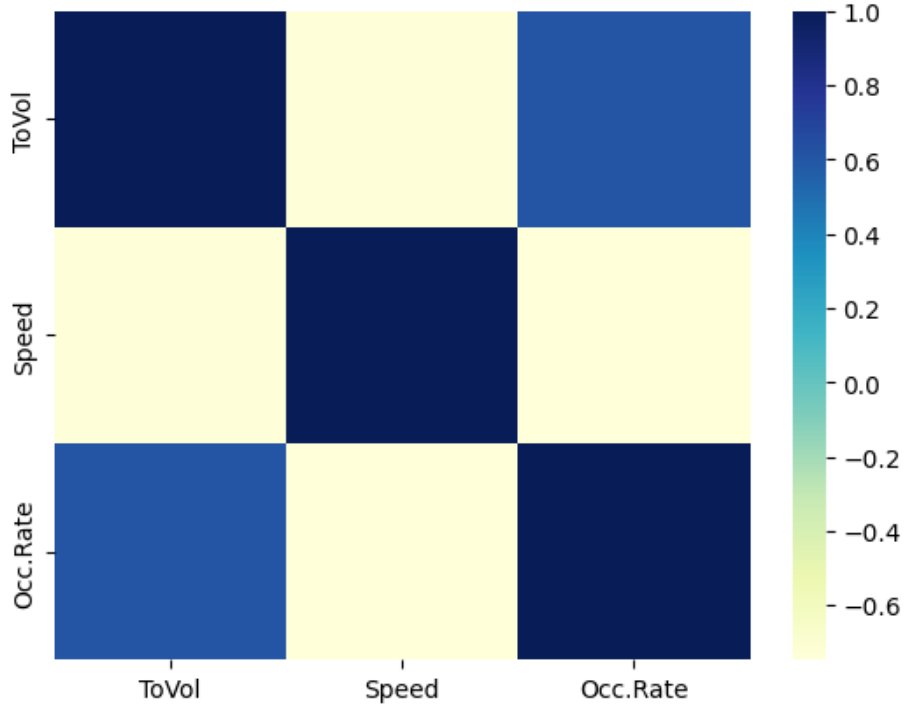
Date	ToVol	Speed	Occ.Rate	label_speed
2017-04-02 0:00	43	50.3	1.90	Normal
2017-04-02 0:05	45	58.9	1.84	Normal
2017-04-02 0:10	46	50.6	1.87	Normal
2017-04-02 0:15	45	50.9	1.72	Normal
2017-04-02 0:20	27	62.2	1.12	Normal

Pairplot와 heatmap

```
sns.pairplot(df2, hue='label_speed')
```

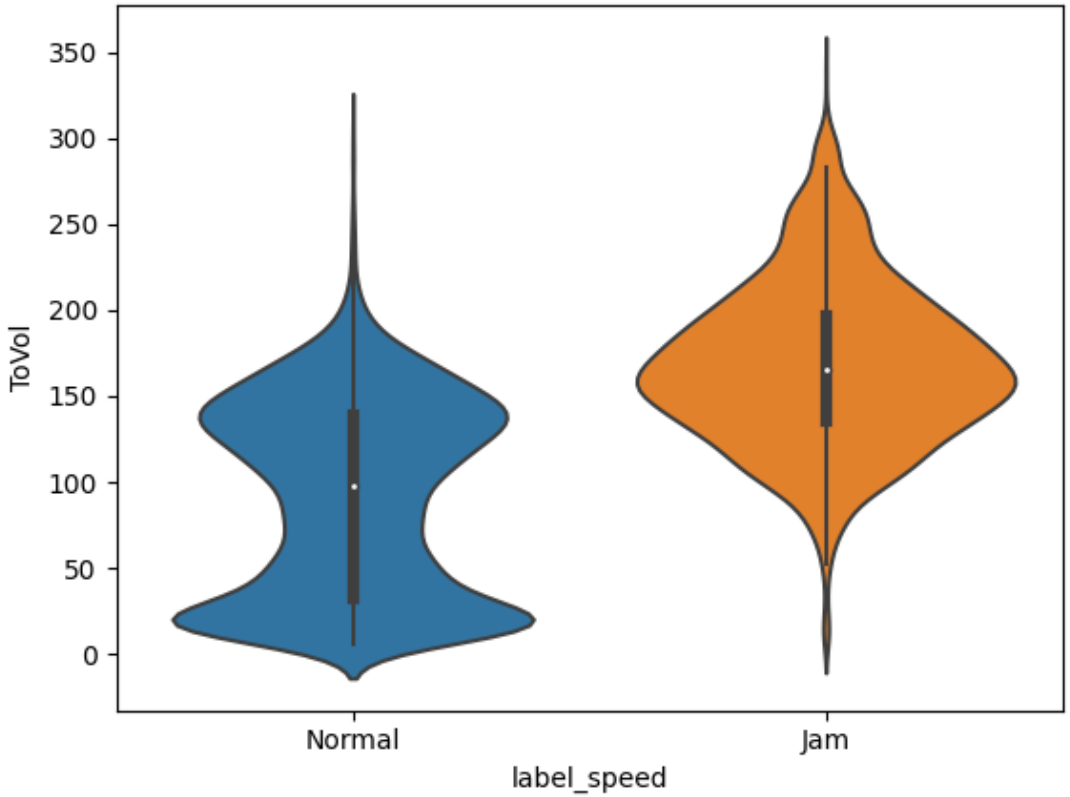



```
sns.heatmap(df2.corr(), cmap='YlGnBu')
```

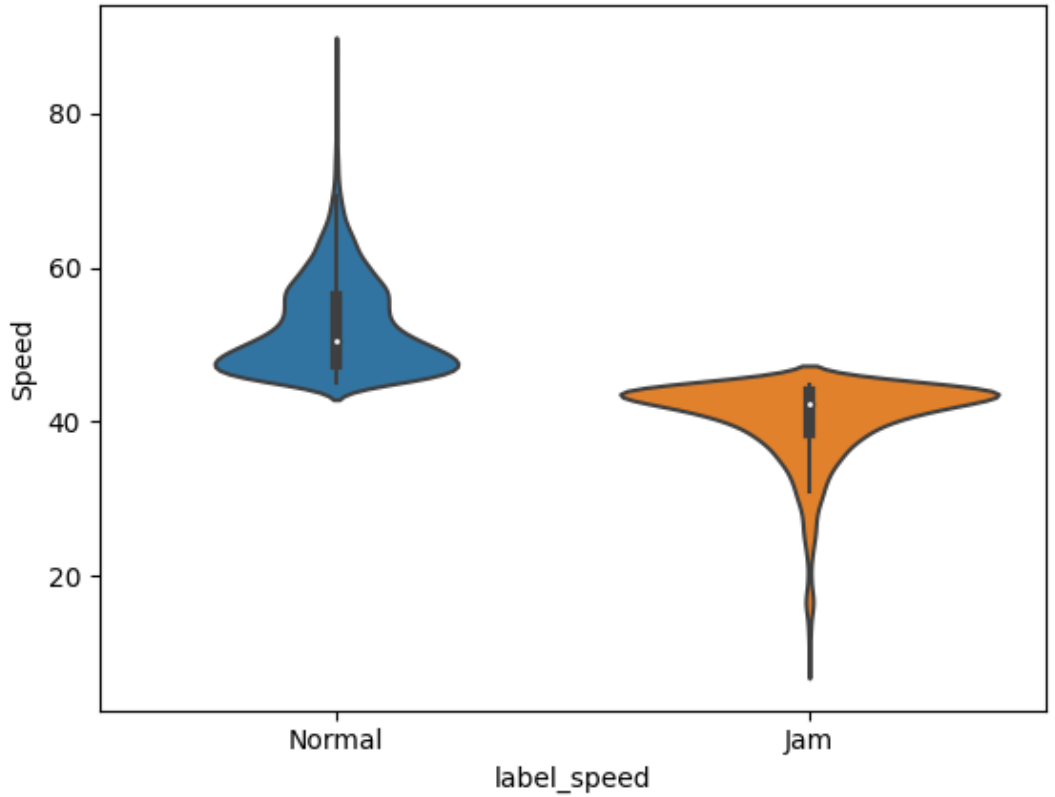


Seabon violin 가시화

```
sns.violinplot(x='label_speed',y='ToVol',data=df)
```

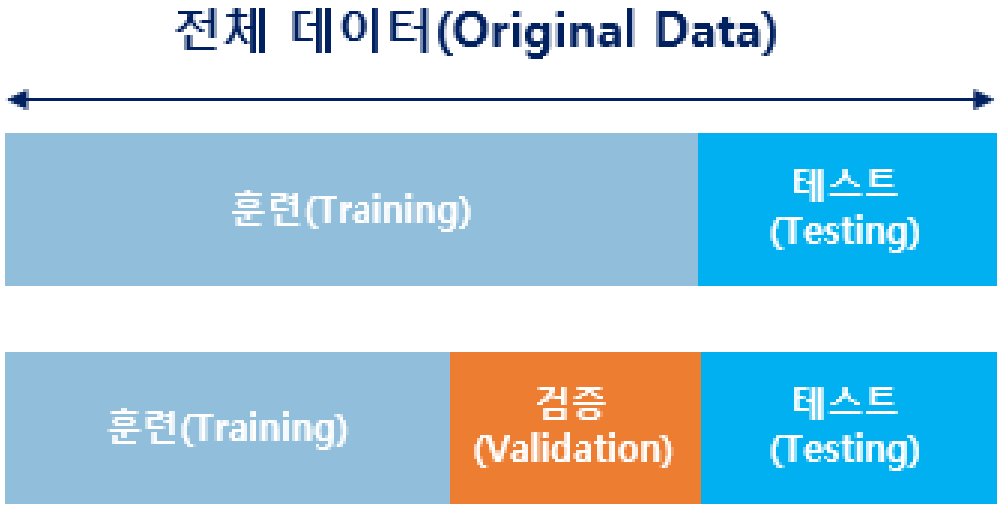



```
sns.violinplot(x='label_speed',y='Speed',data=df)
```



머신 러닝 모델을 위한 데이터 분할

- 실제 모델 훈련 및 평가하기
- 데이터를 훈련용, 검증용, 테스트용 이렇게 세 가지로 분리
  - ✓테스트 데이터는 20%~30% 정도
  - ✓검증용 데이터 약 10%정도 는 과적합을 판단하거나 하이퍼파라미터의 조정을 위한 용도



피처 스케일링과 정규화

❖ 피처 스케일링(feature scaling):

✓서로 다른 변수의 값 범위를 일정하게 맞추는 작업

✓표준화(Standardization)

- 데이터 피처 각각이 평균이 0이고 분산이 1인 가우시안 분포로 변환

$$x_i\_new = \frac{x_i - mean(x)}{stdev(x)}$$

✓정규화(Normalization)

- 서로 다른 피처의 크기를 통일하기 위해 크기를 변환해주는 개념
- 값이 0과 1 사이로 변환하는 것

$$x_i\_new = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

텐서플로우 필수 문서

- TensorFlow 2 예제는 Jupyter 노트북으로 작성
- Google Colab에서 실행

텐서플로우 2.0 시작하기: 초보자용 참고

- 데이터셋 로드
- 모델 만들기
  - ✓tf.keras.Sequential
- 모델은 훈련하기
  - ✓model.compile
  - ✓model.fit
- 모델 평가 하기
  - ✓model.evaluate
- 모델 결과 가시화
  - ✓plt.plot(history.history['loss'])

MLP 분류를 위한 데이터 전처리

데이터 전처리를 위하여 sklearn을 사용하자

In [33]:
```
df.head()
```

Out[33]:

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

레이블을 숫자로 변환

In [34]:

```
#class_dic = {'Jam':0, 'Slow':1, 'Normal':2}
class_dic = {'Jam':0, 'Normal':1}
df['label_speed'] = df['label_speed'].apply(lambda i : class_dic[i])
df.head()
```

Out[34]:

Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label_speed
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	1
2017-04-02 0:10	46	34	12	0	50.6	1.87	1
2017-04-02 0:15	45	36	9	0	50.9	1.72	1
2017-04-02 0:20	27	13	13	1	62.2	1.12	1

Train_test_split

- Speed는 레이블 만들때 사용해서 feature에서 제외해보자

In [35]:
```
features = ['ToVol','LaVol','MeVol','SmVol','Occ.Rate']
X = df[features]
y = df['label_speed']
```

In [36]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,shuffle=False)
print(X_train.shape, y_train.shape)
print(X_test.shape,  y_test.shape)
```

```
(6451, 5) (6451,)
(1613, 5) (1613,)
```

X_train, X_test

In [37]:
```
X_train
```

Date	ToVol	LaVol	MeVol	SmVol	Occ.Rate
2017-04-02 0:00	43	0	9	34	1.90
2017-04-02 0:05	45	0	13	32	1.84
2017-04-02 0:10	46	0	12	34	1.87
2017-04-02 0:15	45	0	9	36	1.72
2017-04-02 0:20	27	1	13	13	1.12
...	...	...	...	...	...
2017-04-24 9:10	119	1	29	89	4.58
2017-04-24 9:15	145	0	31	114	5.47
2017-04-24 9:20	165	2	44	119	7.90
2017-04-24 9:25	177	1	50	126	10.16
2017-04-24 9:30	148	4	43	101	6.52

6451 rows × 5 columns

```
X_test
```

Date	ToVol	LaVol	MeVol	SmVol	Occ.Rate
2017-04-24 9:35	151	2	42	107	7.67
2017-04-24 9:40	147	4	53	90	6.88
2017-04-24 9:45	152	1	49	102	8.36
2017-04-24 9:50	140	0	45	95	5.94
2017-04-24 9:55	173	2	61	110	8.36
...	...	...	...	...	...
2017-04-29 23:35	45	0	10	35	2.01
2017-04-29 23:40	47	0	14	33	1.89
2017-04-29 23:45	32	0	4	28	1.36
2017-04-29 23:50	31	0	10	21	1.40
2017-04-29 23:55	39	0	6	33	1.74

1613 rows × 5 columns

입력 데이터를 표준화 하자

KISTI SINCE 1962

In [40]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [41]:
```
X_train
```

Out[41]:
```
array([[-1.03431236, -0.75262163, -1.06819167, -0.94977398, -0.61081493],
       [-1.0032784 , -0.75262163, -0.8617752 , -0.99216665, -0.61941343],
       [-0.98776142, -0.75262163, -0.91337932, -0.94977398, -0.61511418],
       ...,
       [ 0.85875951,  0.55781349,  0.73795251,  0.85191465,  0.24903504],
       [ 1.0449633 , -0.09740407,  1.04757722,  1.00028901,  0.57291187],
       [ 0.59497081,  1.86824862,  0.68634839,  0.47038059,  0.05126955]])
```

In [42]:
```
X_test
```

Out[42]:
```
array([[ 0.64152176,  0.55781349,  0.63474427,  0.59755861,  0.21607413],
       [ 0.57945382,  1.86824862,  1.20238958,  0.23722088,  0.10286055],
       [ 0.65703874, -0.09740407,  0.9959731 ,  0.49157693,  0.31495687],
       ...,
       [-1.20499917, -0.75262163, -1.32621227, -1.076952  , -0.68820143],
       [-1.22051616, -0.75262163, -1.01658755, -1.22532635, -0.6824691 ],
       [-1.0963803 , -0.75262163, -1.22300403, -0.97097031, -0.63374426]])
```

X_train[1].shape은 feature이다.

In [43]:
```
print(X_train.shape)
```

```
(6451, 5)
```

In [44]:
```
print(X_train[1].shape)
```

```
(5,)
```

In [45]:
```
num_features = len(X_train[1])
print('number of features :', num_features)
```

```
number of features : 5
```

Tensorflow를 사용하자

Deep Neural Network

In [46]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential, optimizers
from tensorflow.keras.layers import Flatten, Dense, Softmax
```

- binary classification 문제로, 마지막 Dense층에는 'sigmoid'를 사용해야한다.

간단한 MLP 모델

KISTI SINCE 1962

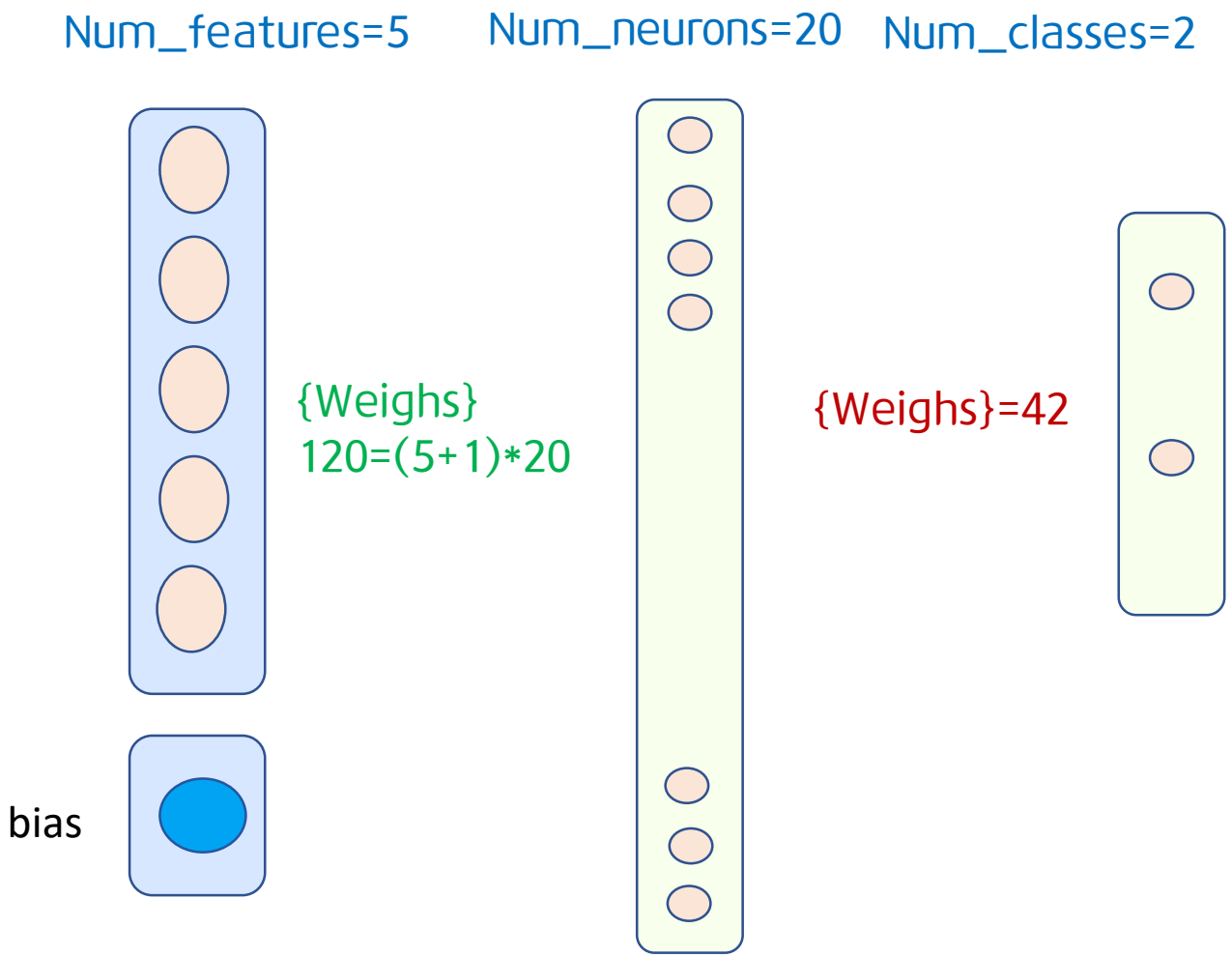
```
num_neurons = 20
def model_mlp():
    model = Sequential([
        Dense(num_neurons, activation='relu', input_shape=[num_features]),
        Dense(num_classes, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

MLP 모델과 파라미터 개수

```
model = model_mlp()
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 20)                120

 dense_6 (Dense)             (None, 2)                 42

=================================================================
Total params: 162
Trainable params: 162
Non-trainable params: 0
_________________________________________________________________
```

MLP 모델 (1번)


Num_features=5
Num_neurons=20
Num_classes=2
{Weighs}
120=(5+1)*20
{Weighs}=42
bias

MLP 모델 아키텍처

```
print('number of features (Input) : ',num_features)
print('number of classes (Output) : ', num_classes)
print('number of neurons (Hidden) : ', num_neurons)
```

```
number of features (Input) :  5
number of classes (Output) :  2
number of neurons (Hidden) :  20
```

```
print('(x+b)*w :', (num_features+1)*num_neurons)
print('(neurons+1)*classes:', (num_neurons+1)*num_classes)
```

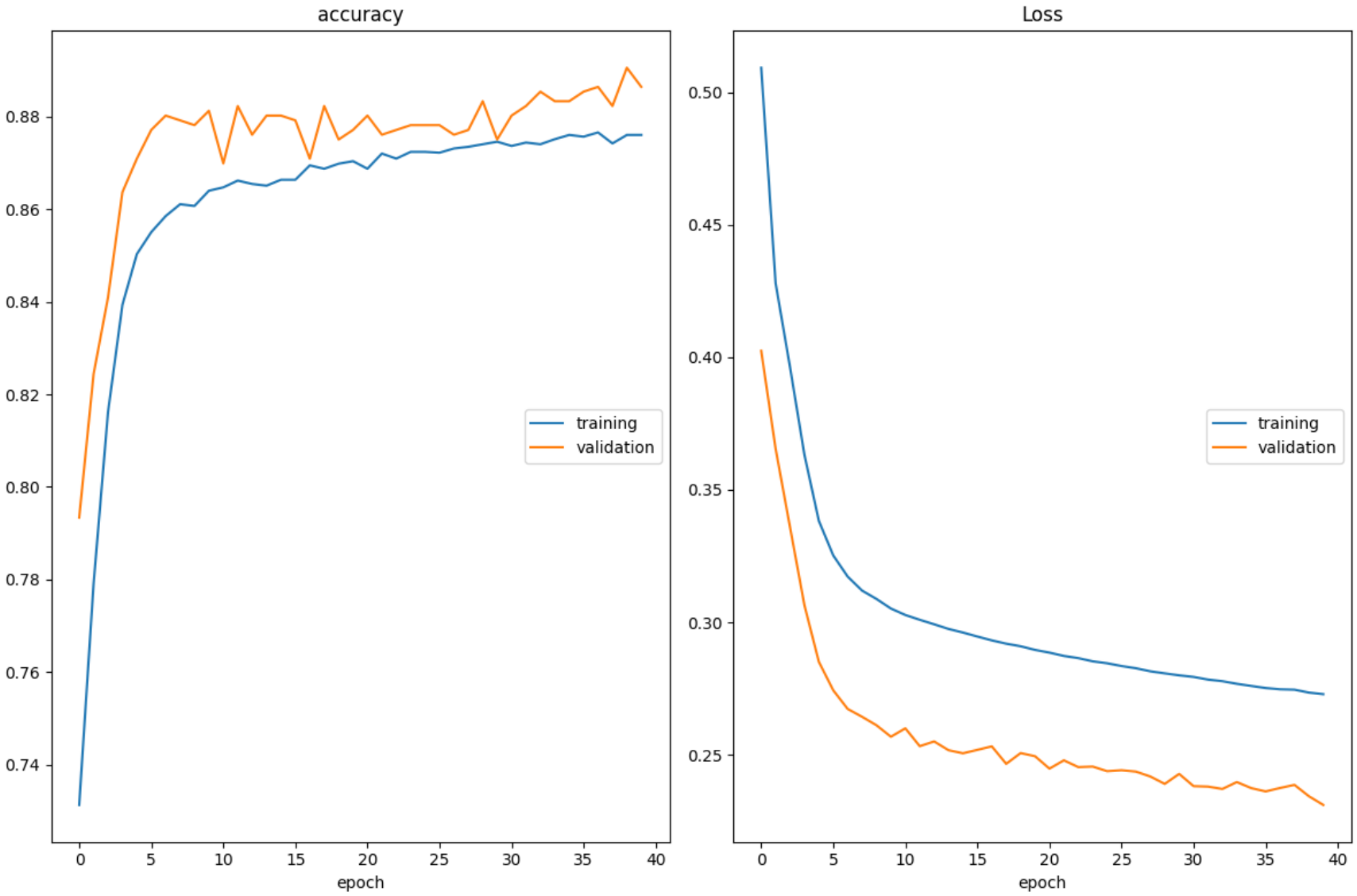
```
(x+b)*w : 120
(neurons+1)*classes: 42
```

모델 실시간 훈련

```
# !pip install livelossplot
from livelossplot import PlotLossesKeras
```

```
history = model.fit(X_train,y_train, epochs=40,
                    validation_split=0.15,
                    batch_size=64,
                    callbacks=[PlotLossesKeras()])
```

PlotLossesKeras



모델의 정확도

```
accuracy
        training                (min:    0.731, max:    0.877, cur:    0.876)
        validation              (min:    0.793, max:    0.890, cur:    0.886)
Loss
        training                (min:    0.273, max:    0.509, cur:    0.273)
        validation              (min:    0.231, max:    0.402, cur:    0.231)
86/86 [==============================] - 1s 9ms/step - loss: 0.2729 - accuracy: 0.8760 - val_loss: 0.2311 - val_accuracy: 0.8864
```

```
model.evaluate(X_test,y_test)
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.3349 - accuracy: 0.8308

[0.3348623514175415, 0.8307501673698425]
```

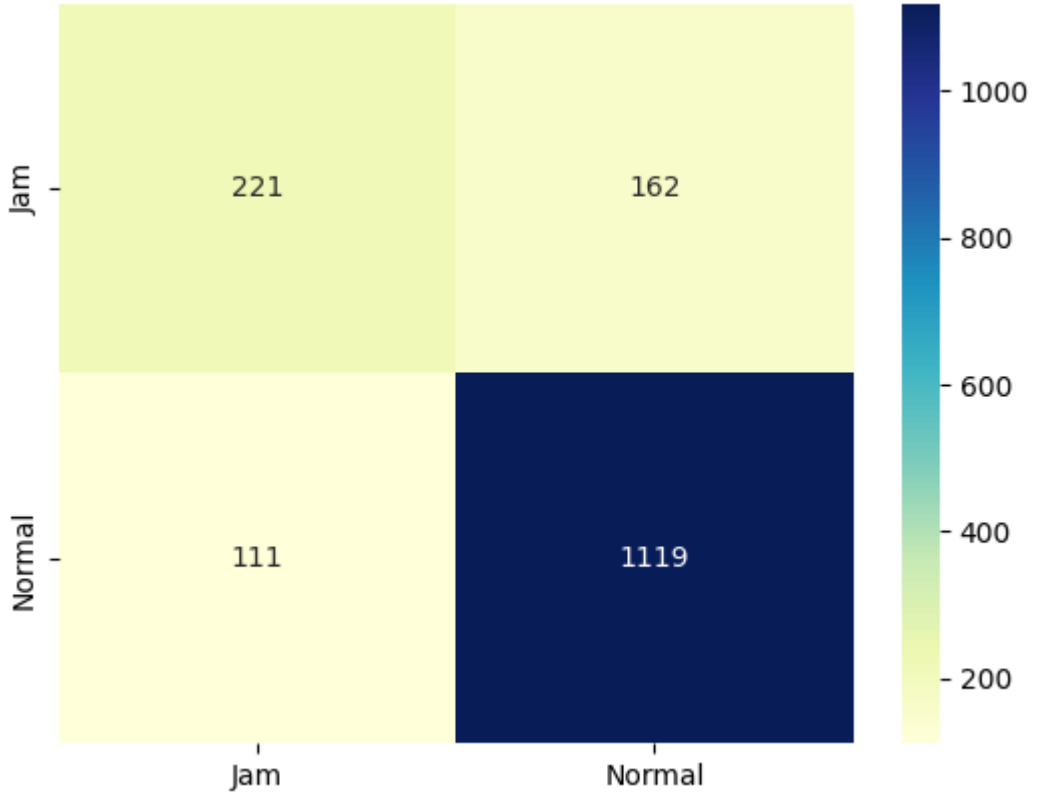
Confusion Matrix

```
y_pred = model.predict(X_test)
```

```
51/51 [==============================] - 0s 1ms/step
```

```
from sklearn.metrics import confusion_matrix
import numpy as np
y_pred_labels = [np.argmax(label) for label in y_pred]
cm = confusion_matrix(y_test,y_pred_labels)
sns.heatmap(cm , annot = True,  cmap='YlGnBu',
            fmt = 'd',xticklabels = class_labels, yticklabels = class_labels)
```

Confusion Matrix



Deep Neural Network

- MLP와 ReLu를 활용한 신경망을 사용하자

In [15]:
```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import Sequential, optimizers
from tensorflow.keras.layers import Flatten, Dense, Softmax

num_features = len(X_train[1])

model = Sequential([
                    Dense(64, activation='relu',input_shape=[num_features]),
                    Dense(64, activation='relu'),
                    Dense(32, activation='relu'),
                    Dense(3, activation='softmax')
])
```

HW3: DNN 분류 정확도의 최고 성능은 얼마인가?

- DNN을 이용한 VDS 데이터 분석
  - ✓가장 성능이 좋은 파라미터는?

	Model	Score
4	Deep Neural Network	0.891506
0	Support Vector Machines	0.874768
3	K-Nearest Neighbours	0.871048
2	RandomForest	0.847489
1	Decision Tree	0.822071

Multi-Class Classification

입력 데이터와 판다스

KISTI SINCE 1962

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv('./vds.csv')
```

In [3]:
```
df.head(3)
```

Out[3]:

	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

In [4]:
```
df.set_index('Date', inplace=True)
```

클래스 레이블링은 3개

KISTI SINCE 1962

3개의 클래스로 레이블링

In [5]:
```
num_classes = 3
class_labels= ['Jam', 'Slow', 'Normal']
```

In [6]:
```
def get_score(speed):
    if speed < 30:
        label = 'Jam'
    elif speed < 50:
        label = 'Slow'
    else :
        label = 'Normal'
    return label
```

레이블링을 추가

In [7]:
```
df["label_speed"] = df["Speed"].apply(lambda spd: get_score(spd))
df.head()
```

Out[7]:

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

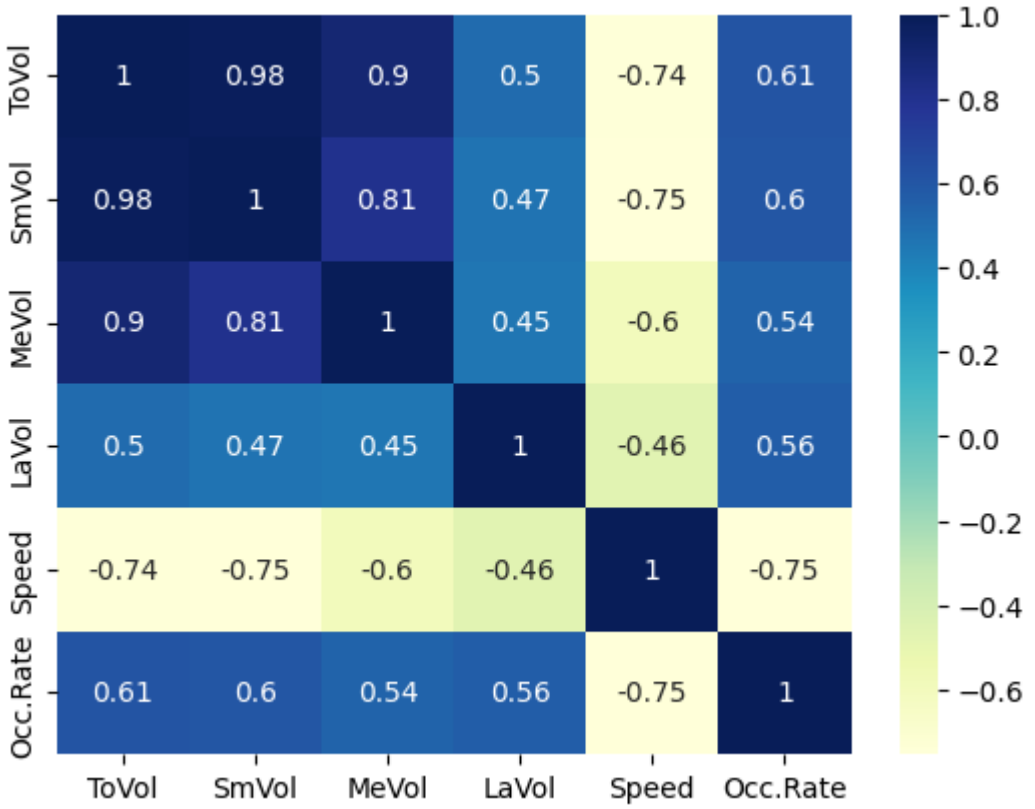
In [8]:
```
df["label_speed"].unique()
```

Out[8]:
```
array(['Normal', 'Slow', 'Jam'], dtype=object)
```

Seaborn의 열지도

```
import matplotlib.pyplot as plt
import seaborn as sns
```

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



클래스를 정수로 인코딩

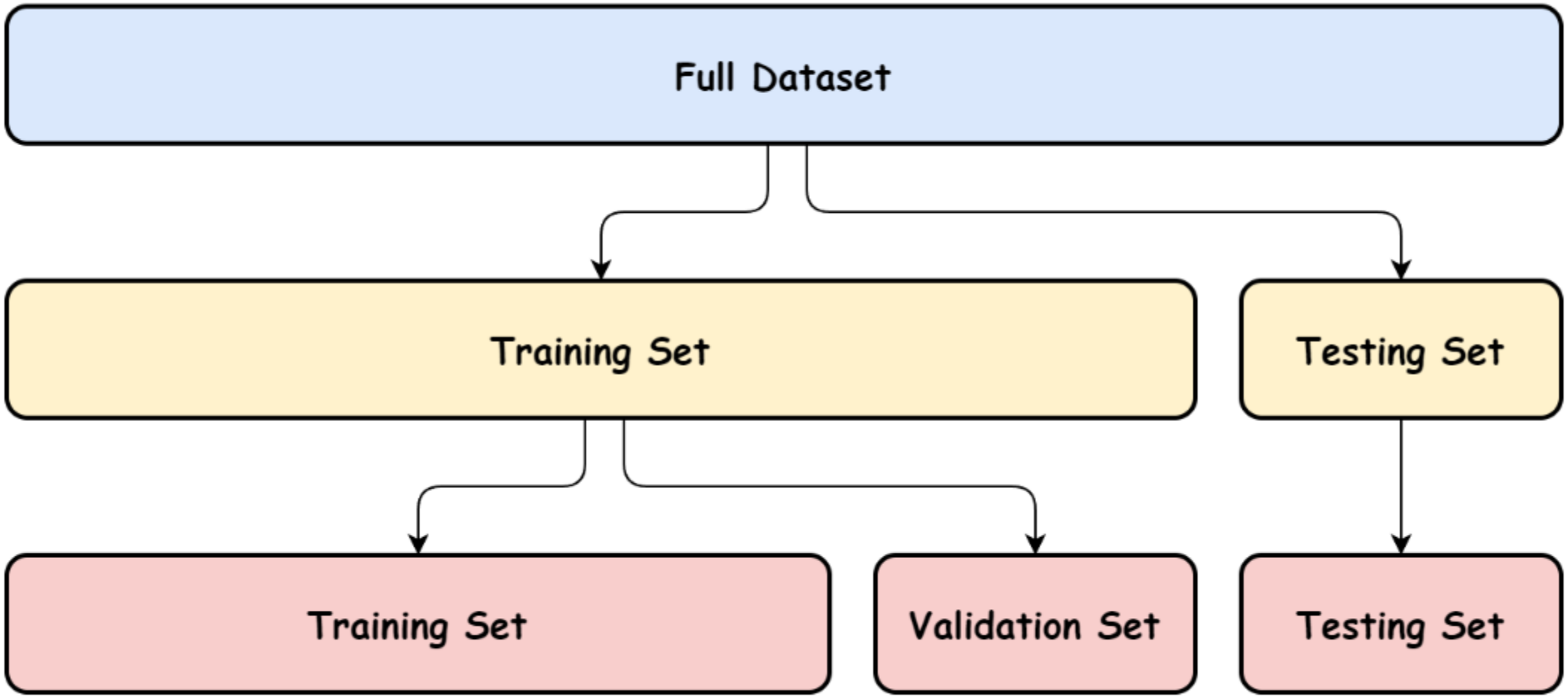
In [11]:
```
class_dic = {'Jam':0, 'Slow':1, 'Normal':2}
#class_dic = {'Jam':0, 'Normal':1}
df['label_speed'] = df['label_speed'].apply(lambda i : class_dic[i])
df.head(2)
```

Out[11]:

Date	ToVol	SmVol	MeVol	LaVol	Speed	Occ.Rate	label_speed
2017-04-02 0:00	43	34	9	0	50.3	1.90	2
2017-04-02 0:05	45	32	13	0	58.9	1.84	2

- Speed는 레이블 만들때 사용해서 feature에서 제외해보자

Splitting Your Data: Training, Testing, and Validation Datasets



Sklearn을 이용한 데이터 분할

In [12]:
```
features = ['ToVol','LaVol','MeVol','SmVol','Occ.Rate']
X = df[features]
y = df['label_speed']
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,shuffle=False)
print(X_train.shape, y_train.shape)
print(X_test.shape,  y_test.shape)
```

```
(6451, 5) (6451,)
(1613, 5) (1613,)
```

입력 데이터를 표준화하자

KISTI SINCE 1962

In [14]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [15]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [16]:
```
print(X_train.shape)
```

```
(6451, 5)
```

In [ ]:
```
print(X_train[1].shape)
```

In [17]:
```
num_features = len(X_train[1])
print('number of features :', num_features)
```

```
number of features : 5
```

Tensorflow 설정

Deep Neural Network ¶

In [18]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential, optimizers
from tensorflow.keras.layers import Flatten, Dense, Softmax
```

- binary classification 문제로, 마지막 Dense층에는 'sigmoid'를 사용해야한다.

멀티클래스 MLP 모델 (#2)

KISTI SINCE 1962

```
num_neurons = 20
def model_mlp_softmax():
    model = Sequential([
        Dense(num_neurons, activation='relu', input_shape=[num_features]),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

모델 파라미터

In [20]:
```
print('number of features (Input) : ',num_features)
print('number of classes (Output) : ', num_classes)
print('number of neurons (Hidden) : ', num_neurons)
```

```
number of features (Input) :  5
number of classes (Output) :  3
number of neurons (Hidden) :  20
```

In [21]:
```
print('(x+b)*w :', (num_features+1)*num_neurons)
print('(neurons+1)*classes:', (num_neurons+1)*num_classes)
```

```
(x+b)*w : 120
(neurons+1)*classes: 63
```

Model.summary()

KISTI SINCE 1962

In [22]:

```
model = model_mlp_softmax()
model.summary()
```

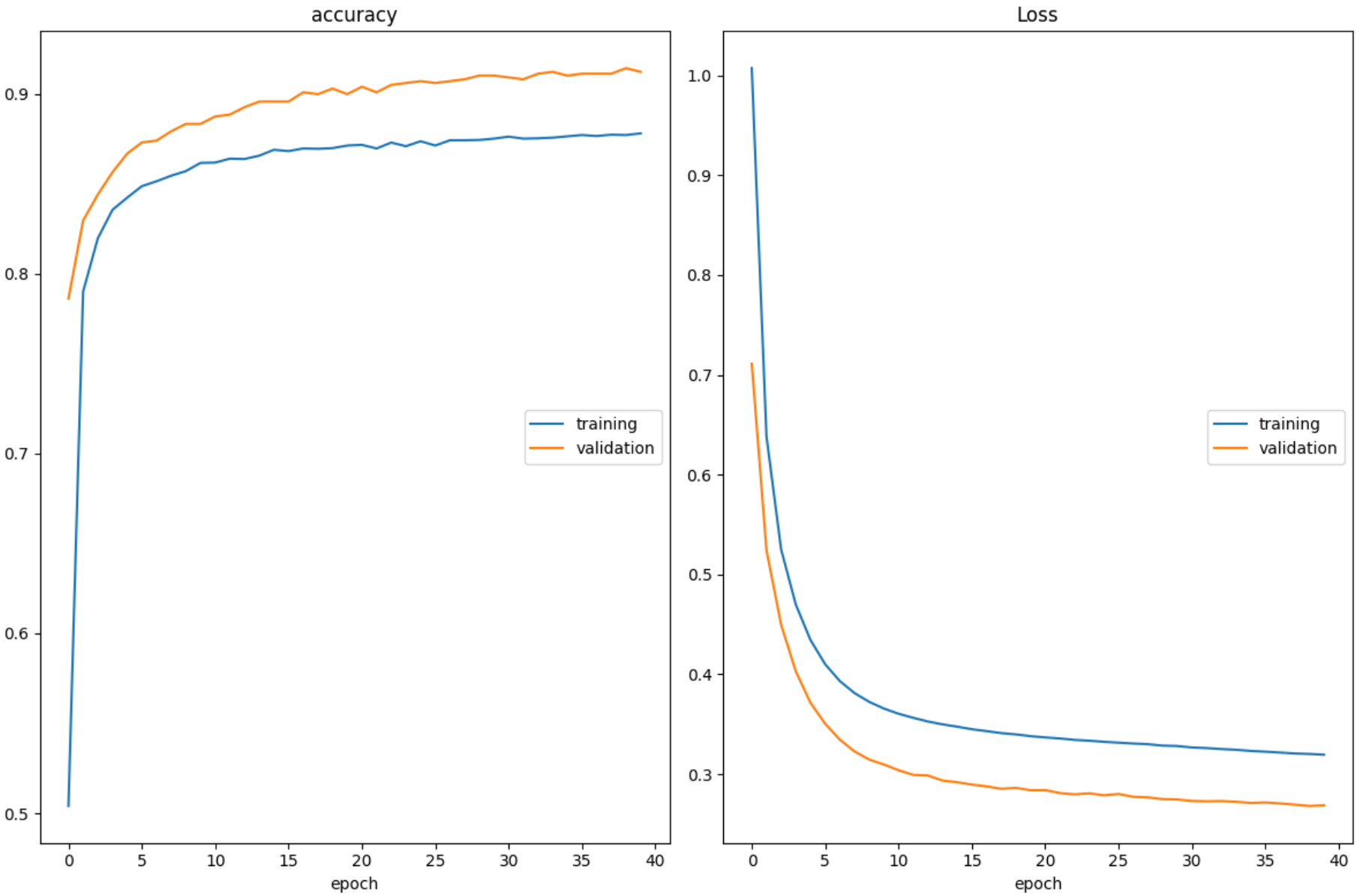
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                120

 dense_1 (Dense)             (None, 3)                 63

=================================================================
Total params: 183
Trainable params: 183
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련

In [23]:
```
# !pip install livelossplot
from livelossplot import PlotLossesKeras
```

In [24]:
```
history = model.fit(X_train,y_train, epochs=40,
                    validation_split=0.15,
                    batch_size=64,
                    callbacks=[PlotLossesKeras()])
```

PlotLossKeras



정확도 (validation vs test)

```
accuracy
        training                (min:    0.504, max:    0.878, cur:    0.878)
        validation              (min:    0.786, max:    0.914, cur:    0.912)
Loss
        training                (min:    0.319, max:    1.007, cur:    0.319)
        validation              (min:    0.268, max:    0.711, cur:    0.269)
86/86 [==============================] - 1s 9ms/step - loss: 0.3195 - accuracy: 0.8780 - val_loss: 0.2689 - val_accuracy: 0.9122
```

```
model.evaluate(X_test,y_test)
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.3263 - accuracy: 0.8785
```

```
[0.32632961869239807, 0.8784872889518738]
```

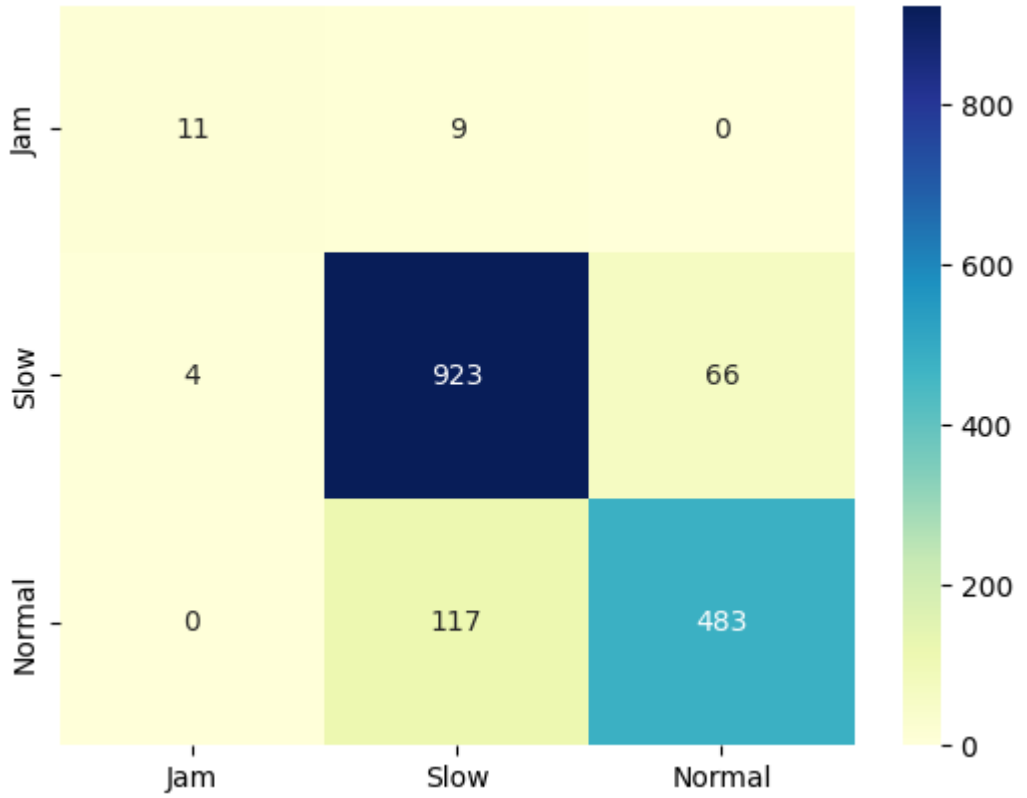
Confusion matrix

```
y_pred = model.predict(X_test)
```

```
51/51 [==============================] - 0s 2ms/step
```

```
from sklearn.metrics import confusion_matrix
import numpy as np
y_pred_labels = [np.argmax(label) for label in y_pred]
cm = confusion_matrix(y_test,y_pred_labels)
sns.heatmap(cm , annot = True,  cmap='YlGnBu',
            fmt = 'd',xticklabels = class_labels, yticklabels = class_labels)
```

Confusion matrix heatmap



KISTI SINCE 1962

2023 TRUST KISTI

Korea Institute of Science and Technology Information