Keras iris Modeling

```
import warnings
warnings.filterwarnings('ignore')
```

- 실습용 데이터 설정
  - iris.csv

```
import seaborn as sns

iris = sns.load_dataset('iris')
```

- pandas DataFrame

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

I. Data Preprocessing

1) iris.Species 빈도분석

- Species : setosa, virginica, versicolor

```
iris.species.value_counts()
```

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

2) DataFrame to Array & Casting

```
iris_AR = iris.values

iris_AR
```

```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],
```

```
[4.9, 3.0, 1.4, 0.2, 'setosa'],
[4.7, 3.2, 1.3, 0.2, 'setosa'],
[4.6, 3.1, 1.5, 0.2, 'setosa'],
[5.0, 3.6, 1.4, 0.2, 'setosa'],
[5.4, 3.9, 1.7, 0.4, 'setosa'],
[4.6, 3.4, 1.4, 0.3, 'setosa'],
[5.0, 3.4, 1.5, 0.2, 'setosa'],
[4.4, 2.9, 1.4, 0.2, 'setosa'],
[4.9, 3.1, 1.5, 0.1, 'setosa'],
[5.4, 3.7, 1.5, 0.2, 'setosa'],
[4.8, 3.4, 1.6, 0.2, 'setosa'],
[4.8, 3.0, 1.4, 0.1, 'setosa'],
[4.3, 3.0, 1.1, 0.1, 'setosa'],
[5.8, 4.0, 1.2, 0.2, 'setosa'],
[5.7, 4.4, 1.5, 0.4, 'setosa'],
[5.4, 3.9, 1.3, 0.4, 'setosa'],
[5.1, 3.5, 1.4, 0.3, 'setosa'],
[5.7, 3.8, 1.7, 0.3, 'setosa'],
[5.1, 3.8, 1.5, 0.3, 'setosa'],
[5.4, 3.4, 1.7, 0.2, 'setosa'],
[5.1, 3.7, 1.5, 0.4, 'setosa'],
[4.6, 3.6, 1.0, 0.2, 'setosa'],
[5.1, 3.3, 1.7, 0.5, 'setosa'],
[4.8, 3.4, 1.9, 0.2, 'setosa'],
[5.0, 3.0, 1.6, 0.2, 'setosa'],
[5.0, 3.4, 1.6, 0.4, 'setosa'],
[5.2, 3.5, 1.5, 0.2, 'setosa'],
[5.2, 3.4, 1.4, 0.2, 'setosa'],
[4.7, 3.2, 1.6, 0.2, 'setosa'],
[4.8, 3.1, 1.6, 0.2, 'setosa'],
[5.4, 3.4, 1.5, 0.4, 'setosa'],
[5.2, 4.1, 1.5, 0.1, 'setosa'],
[5.5, 4.2, 1.4, 0.2, 'setosa'],
[4.9, 3.1, 1.5, 0.2, 'setosa'],
[5.0, 3.2, 1.2, 0.2, 'setosa'],
[5.5, 3.5, 1.3, 0.2, 'setosa'],
[4.9, 3.6, 1.4, 0.1, 'setosa'],
[4.4, 3.0, 1.3, 0.2, 'setosa'],
[5.1, 3.4, 1.5, 0.2, 'setosa'],
[5.0, 3.5, 1.3, 0.3, 'setosa'],
[4.5, 2.3, 1.3, 0.3, 'setosa'],
[4.4, 3.2, 1.3, 0.2, 'setosa'],
[5.0, 3.5, 1.6, 0.6, 'setosa'],
[5.1, 3.8, 1.9, 0.4, 'setosa'],
[4.8, 3.0, 1.4, 0.3, 'setosa'],
[5.1, 3.8, 1.6, 0.2, 'setosa'],
[4.6, 3.2, 1.4, 0.2, 'setosa'],
[5.3, 3.7, 1.5, 0.2, 'setosa'],
[5.0, 3.3, 1.4, 0.2, 'setosa'],
[7.0, 3.2, 4.7, 1.4, 'versicolor'],
[6.4, 3.2, 4.5, 1.5, 'versicolor'],
[6.9, 3.1, 4.9, 1.5, 'versicolor'],
[5.5, 2.3, 4.0, 1.3, 'versicolor'],
[6.5, 2.8, 4.6, 1.5, 'versicolor'],
[5.7, 2.8, 4.5, 1.3, 'versicolor'],
[6.3, 3.3, 4.7, 1.6, 'versicolor'],
[4.9, 2.4, 3.3, 1.0, 'versicolor'],
[6.6, 2.9, 4.6, 1.3, 'versicolor'],
[5.2, 2.7, 3.9, 1.4, 'versicolor'],
```

- object to float

```
AR_X = iris_AR[:, 0:4].astype(float)
AR_y = iris_AR[:, 4]

AR_X.shape, AR_y.shape
```

```
((150, 4), (150,))
```

3) One Hot Encoding with sklearn & Keras

- LabelEncoder( )
  - ['setosa', 'virginica', 'virsicolor'] to [0, 1, 2]

```
from sklearn.preprocessing import LabelEncoder

encoder =  LabelEncoder()
AR_yLBE = encoder.fit_transform(AR_y)
```

```
AR_yLBE
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

- One-Hot Encoding - to_categorical( )

```
from tensorflow.keras.utils import to_categorical

AR_yOHE = to_categorical(AR_yLBE)

AR_yOHE
```

```
array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
```

4) Version check

- tensorFlow Version

```
import tensorflow

tensorflow.__version__
```

```
'2.5.0'
```

- Keras Version

```
tensorflow.keras.__version__
```

```
'2.5.0'
```

5) train_test_split( )

- 7 : 3

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(AR_X, AR_yOHE,
                                                    test_size = 0.3,
                                                    random_state = 2045)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((105, 4), (45, 4), (105, 3), (45, 3))
```

II. Keras Modeling

1) Keras models & layers Import

```
from tensorflow.keras import models
from tensorflow.keras import layers
```

2) Model Define

- 모델 신경망 구조 정의

```
Model_iris = models.Sequential()

Model_iris.add(layers.Dense(16, activation = 'relu', input_shape = (4,)))
Model_iris.add(layers.Dense(8, activation = 'relu'))
Model_iris.add(layers.Dense(3, activation = 'softmax'))
```

- 모델 구조 확인
  - Layers & Parameters

```
Model_iris.summary()
```

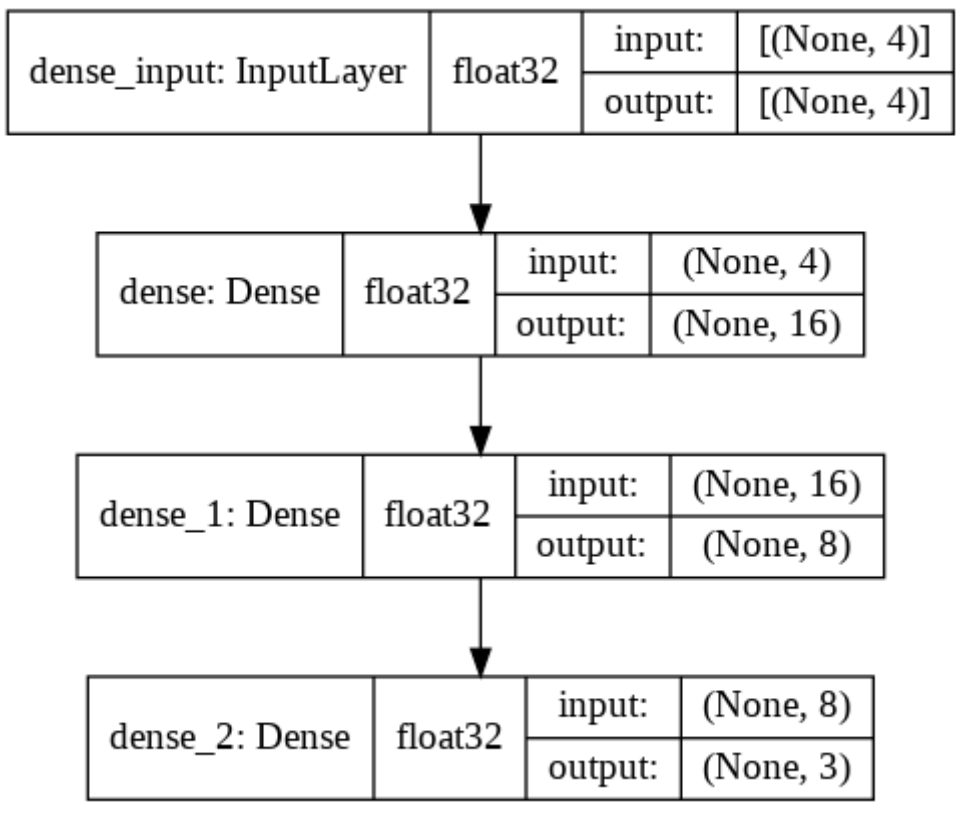
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                80
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 27
=================================================================
```

```
Total params: 243
Trainable params: 243
Non-trainable params: 0
_________________________________________________________________
```

- 모델 레이어 시각화

```
from tensorflow.keras import utils

utils.plot_model(Model_iris,
                 show_shapes = True,
                 show_dtype = True)
```



3) Model Compile

- 모델 학습방법 설정

```
Model_iris.compile(loss = 'categorical_crossentropy',
                   optimizer = 'adam',
                   metrics = ['accuracy'])
```

4) Model Fit

- 모델 학습 수행

```
History_iris = Model_iris.fit(X_train, y_train,
                              epochs = 500,
                              batch_size = 7,
                              validation_data = (X_test, y_test))
```

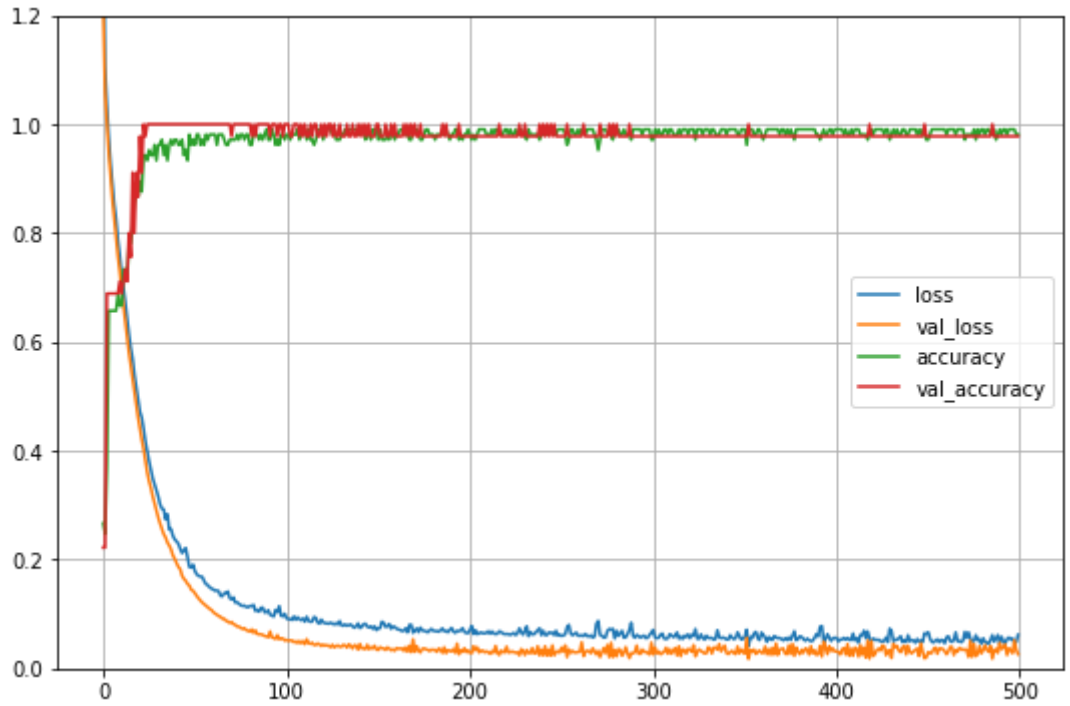
```
Epoch 1/500
15/15 [==============================] - 1s 13ms/step - loss: 1.4514 - accuracy: 0.2667 - val_loss: 1.2099 - val_accuracy: 0.2222
Epoch 2/500
15/15 [==============================] - 0s 2ms/step - loss: 1.1064 - accuracy: 0.2476 - val_loss: 1.0585 - val_accuracy: 0.2222
Epoch 3/500
15/15 [==============================] - 0s 4ms/step - loss: 1.0366 - accuracy: 0.3714 - val_loss: 1.0025 - val_accuracy: 0.6889
Epoch 4/500
15/15 [==============================] - 0s 2ms/step - loss: 0.9765 - accuracy: 0.6571 - val_loss: 0.9476 - val_accuracy: 0.6889
Epoch 5/500
15/15 [==============================] - 0s 2ms/step - loss: 0.9290 - accuracy: 0.6571 - val_loss: 0.9019 - val_accuracy: 0.6889
Epoch 6/500
15/15 [==============================] - 0s 3ms/step - loss: 0.8901 - accuracy: 0.6571 - val_loss: 0.8610 - val_accuracy: 0.6889
Epoch 7/500
15/15 [==============================] - 0s 3ms/step - loss: 0.8531 - accuracy: 0.6571 - val_loss: 0.8256 - val_accuracy: 0.6889
Epoch 8/500
15/15 [==============================] - 0s 2ms/step - loss: 0.8263 - accuracy: 0.6571 - val_loss: 0.7927 - val_accuracy: 0.6889
Epoch 9/500
```

```
15/15 [==============================] - 0s 3ms/step - loss: 0.7884 - accuracy: 0.6857 - val_loss: 0.7575 - val_accuracy: 0.6889
Epoch 10/500
15/15 [==============================] - 0s 4ms/step - loss: 0.7595 - accuracy: 0.6667 - val_loss: 0.7292 - val_accuracy: 0.7111
Epoch 11/500
15/15 [==============================] - 0s 3ms/step - loss: 0.7269 - accuracy: 0.6667 - val_loss: 0.6956 - val_accuracy: 0.6889
Epoch 12/500
15/15 [==============================] - 0s 4ms/step - loss: 0.7005 - accuracy: 0.7333 - val_loss: 0.6666 - val_accuracy: 0.7111
Epoch 13/500
15/15 [==============================] - 0s 3ms/step - loss: 0.6736 - accuracy: 0.7333 - val_loss: 0.6369 - val_accuracy: 0.7333
Epoch 14/500
15/15 [==============================] - 0s 2ms/step - loss: 0.6425 - accuracy: 0.7143 - val_loss: 0.6087 - val_accuracy: 0.7111
Epoch 15/500
15/15 [==============================] - 0s 2ms/step - loss: 0.6148 - accuracy: 0.7810 - val_loss: 0.5806 - val_accuracy: 0.8000
Epoch 16/500
15/15 [==============================] - 0s 3ms/step - loss: 0.5892 - accuracy: 0.7810 - val_loss: 0.5566 - val_accuracy: 0.7556
Epoch 17/500
15/15 [==============================] - 0s 2ms/step - loss: 0.5695 - accuracy: 0.8095 - val_loss: 0.5309 - val_accuracy: 0.9111
Epoch 18/500
15/15 [==============================] - 0s 3ms/step - loss: 0.5402 - accuracy: 0.8571 - val_loss: 0.5073 - val_accuracy: 0.8000
Epoch 19/500
15/15 [==============================] - 0s 2ms/step - loss: 0.5166 - accuracy: 0.8667 - val_loss: 0.4827 - val_accuracy: 0.9111
Epoch 20/500
15/15 [==============================] - 0s 4ms/step - loss: 0.4956 - accuracy: 0.8667 - val_loss: 0.4612 - val_accuracy: 0.8667
Epoch 21/500
15/15 [==============================] - 0s 2ms/step - loss: 0.4721 - accuracy: 0.8952 - val_loss: 0.4403 - val_accuracy: 0.9778
Epoch 22/500
15/15 [==============================] - 0s 2ms/step - loss: 0.4609 - accuracy: 0.8762 - val_loss: 0.4206 - val_accuracy: 0.9111
Epoch 23/500
15/15 [==============================] - 0s 4ms/step - loss: 0.4405 - accuracy: 0.9429 - val_loss: 0.4033 - val_accuracy: 1.0000
Epoch 24/500
15/15 [==============================] - 0s 4ms/step - loss: 0.4170 - accuracy: 0.9429 - val_loss: 0.3825 - val_accuracy: 0.9778
Epoch 25/500
15/15 [==============================] - 0s 4ms/step - loss: 0.3982 - accuracy: 0.9333 - val_loss: 0.3600 - val_accuracy: 1.0000
Epoch 26/500
15/15 [==============================] - 0s 3ms/step - loss: 0.3808 - accuracy: 0.9524 - val_loss: 0.3440 - val_accuracy: 1.0000
Epoch 27/500
15/15 [==============================] - 0s 2ms/step - loss: 0.3656 - accuracy: 0.9429 - val_loss: 0.3316 - val_accuracy: 1.0000
Epoch 28/500
15/15 [==============================] - 0s 3ms/step - loss: 0.3493 - accuracy: 0.9429 - val_loss: 0.3162 - val_accuracy: 1.0000
Epoch 29/500
15/15 [==============================] - 0s 3ms/step - loss: 0.3389 - accuracy: 0.9619 - val_loss: 0.3027 - val_accuracy: 1.0000
Epoch 30/500
```

5) 학습 결과 시각화

```
import matplotlib.pyplot as plt

plt.figure(figsize = (9, 6))
plt.ylim(0, 1.2)
plt.plot(History_iris.history['loss'])
plt.plot(History_iris.history['val_loss'])
plt.plot(History_iris.history['accuracy'])
plt.plot(History_iris.history['val_accuracy'])
plt.legend(['loss', 'val_loss', 'accuracy', 'val_accuracy'])
plt.grid()
plt.show()
```



6) Model Evaluate

- Loss & Accuracy

```
loss, accuracy = Model_iris.evaluate(X_test, y_test)

print('Loss = {:.2f}'.format(loss))
print('Accuracy = {:.2f}'.format(accuracy))
```

```
2/2 [==============================] - 0s 4ms/step - loss: 0.0262 - accuracy: 0.9778
Loss = 0.03
Accuracy = 0.98
```

7) Model Predict

- Probability

```
import numpy as np
np.set_printoptions(suppress = True, precision = 5)

Model_iris.predict(X_test)
```

```
array([[0.99992, 0.00008, 0.     ],
       [0.99916, 0.00084, 0.     ],
       [0.00248, 0.99751, 0.00001],
       [0.     , 0.00007, 0.99993],
       [0.99999, 0.00001, 0.     ],
       [0.00012, 0.99985, 0.00002],
       [0.     , 0.00151, 0.99849],
       [0.99999, 0.00001, 0.     ],
       [0.     , 0.00294, 0.99706],
       [0.99991, 0.00009, 0.     ],
       [0.00008, 0.98341, 0.01652],
       [0.     , 0.00114, 0.99886],
       [0.00002, 0.99908, 0.0009 ],
       [1.     , 0.     , 0.     ],
       [0.99999, 0.00001, 0.     ],
       [0.00007, 0.99814, 0.00179],
       [0.00003, 0.99031, 0.00966],
       [0.99995, 0.00005, 0.     ],
       [0.00002, 0.99979, 0.00019],
       [0.99999, 0.00001, 0.     ],
       [0.99993, 0.00007, 0.     ],
       [0.99993, 0.00007, 0.     ],
       [0.     , 0.00019, 0.99981],
       [0.99997, 0.00003, 0.     ],
       [0.     , 0.0168 , 0.9832 ],
       [0.00001, 0.02977, 0.97022],
       [0.99998, 0.00002, 0.     ],
       [0.00001, 0.99753, 0.00246],
       [0.00001, 0.99863, 0.00136],
       [0.     , 0.06546, 0.93453],
       [0.99999, 0.00001, 0.     ],
       [0.00002, 0.99842, 0.00156],
       [0.99999, 0.00001, 0.     ],
       [0.00002, 0.63522, 0.36476],
       [0.00004, 0.99733, 0.00264],
       [0.00007, 0.99992, 0.00001],
       [0.     , 0.00008, 0.99992],
       [0.00008, 0.99984, 0.00008],
       [0.99995, 0.00005, 0.     ],
       [0.00007, 0.9979 , 0.00203],
       [0.     , 0.00008, 0.99992],
       [0.9999 , 0.0001 , 0.     ],
       [0.     , 0.00225, 0.99775],
       [0.     , 0.00032, 0.99968],
       [0.     , 0.00402, 0.99598]], dtype=float32)
```

- Class

```
y_hat = Model_iris.predict_classes(X_test)

y_hat
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
```

```
       2])
```

- Probability to Class

```
np.argmax(Model_iris.predict(X_test), axis = 1)
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

- One-Hot Encoding to Array
  - np.argmax( ) : 다차원 배열의 차원에 따라 가장 큰 값의 인덱스를 반환
  - axis = 1 : 열기준

```
y = np.argmax(y_test, axis = 1)

y
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 2, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

- Confusion Matrix & Claasification Report

```
from sklearn.metrics import confusion_matrix, classification_report

confusion_matrix(y, y_hat)
```

```
array([[17,  0,  0],
       [ 0, 14,  0],
       [ 0,  1, 13]])
```

```
print(classification_report(y, y_hat,
                            target_names = ['setosa',
                                            'virginica',
                                            'versicolor']))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        17
   virginica       0.93      1.00      0.97        14
  versicolor       1.00      0.93      0.96        14

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

III. Model Save & Load

1) File System

- Save to Colab File System

```
!ls -l
```

```
total 36
-rw-r--r-- 1 root root 29384 Jul 20 05:47 model.png
drwxr-xr-x 1 root root  4096 Jul 16 13:20 sample_data
```

```
Model_iris.save('Model_iris.h5')

!ls -l
```

```
total 72
-rw-r--r-- 1 root root 34600 Jul 20 05:47 Model_iris.h5
```

```
-rw-r--r-- 1 root root 29384 Jul 20 05:47 model.png
drwxr-xr-x 1 root root  4096 Jul 16 13:20 sample_data
```

- Download Colab File System to Local File System

```
from google.colab import files

files.download('Model_iris.h5')
```

- Load from Colab File System

```
from keras.models import load_model

Model_local = load_model('Model_iris.h5')
```

```
Model_local.predict_classes(X_test)
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

2) Google Drive

- Mount Google Drive

```
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

- Check Mounted_Drive

```
!ls -l '/content/drive/My Drive/Colab Notebooks/datasets'
```

```
total 3850167
-rw------- 1 root root   31374019 Mar 27 09:41 Camel.zip
-rw------- 1 root root      20066 Mar  4 04:45 cat.1700.jpg
-rw------- 1 root root   69155672 Mar  4 04:46 creditCardFraud.zip
-rw------- 1 root root   82003032 Mar 23 07:53 DataSet.pkl
-rw------- 1 root root   90618980 Mar  4 04:51 dogs_and_cats_small.zip
-rw------- 1 root root   54561944 Mar 27 09:42 Face.zip
-rw------- 1 root root  862182613 Mar 21 03:21 GloVe.zip
-rw------- 1 root root  149574867 Apr 12 01:53 horse-or-human.zip
-rw------- 1 root root 1245927936 Mar 12 01:01 imagenetV2.zip
-rw------- 1 root root    8204887 Mar  4 04:45 Images_500.zip
-rw------- 1 root root   60711700 Mar 21 01:09 IMDB.zip
-rw------- 1 root root    4240457 Mar 14 09:13 Kaggle_Customer_Satisfaction.zip
-rw------- 1 root root   80596565 May  4  2020 ko_w2v.zip
-rw------- 1 root root   12929865 Mar  4 04:42 Logo_Data.zip
-rw------- 1 root root   18272469 Mar  4 04:50 MNIST.csv
-rw------- 1 root root    7903524 May  4  2020 naverRatings.zip
-rw------- 1 root root   22824989 Mar  7 07:09 Online_Retail.zip
-rw------- 1 root root        741 Mar  4 04:44 Pll.csv
-rw------- 1 root root 1141460846 Mar  4 04:50 waferImages.zip
```

```
import pandas as pd

DF = pd.read_csv('/content/drive/My Drive/Colab Notebooks/datasets/Pll.csv')

DF.head(3)
```

	Name	Gender	Age	Grade	Picture	BloodType	Height	Weight
0	송태섭	남자	21	3	무	B	179.1	63.9

- Save to Mounted Google Drive Directory

```
Model_iris.save('/content/drive/My Drive/Colab Notebooks/models/001_Model_iris.h5')
```

```
!ls -l '/content/drive/My Drive/Colab Notebooks/models'
```

```
total 34
-rw------- 1 root root 34600 Jul 20 05:48 001_Model_iris.h5
```

- Load from Mounted Google Drive Directory

```
from keras.models import load_model

Model_google = load_model('/content/drive/My Drive/Colab Notebooks/models/001_Model_iris.h5')
```

```
Model_google.predict_classes(X_test)
```

```
array([0, 0, 1, 2, 0, 1, 2, 0, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       2, 0, 2, 2, 0, 1, 1, 2, 0, 1, 0, 1, 1, 1, 2, 1, 0, 1, 2, 0, 2, 2,
       2])
```

#

#

#

The End

#

#

#