```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
train_data = pd.read_excel('/content/drive/MyDrive/PCA_STAGE_BASED_CLASSIFICATIO
train_data.head()
```

	PCA_STAGE	GSHG0000008	GSHG0000017	GSHG0000018	GSHG0000026	GSHG000002
0	pT3a	7.604725	5.93074	9.28309	7.242785	7.00552
1	N_1	6.465740	6.08746	8.97728	7.531295	7.26675
2	pT3b	7.317235	6.58496	8.29002	7.139515	7.33960
3	N_2	6.445800	6.85798	8.85175	7.647455	7.50382
4	pT3a	7.021685	6.32193	8.47978	7.554565	7.44177

5 rows × 16204 columns

Data Wrangling

```
train_data.isnull().sum()
```

```
PCA_STAGE      0
GSHG0000008    0
GSHG0000017    0
GSHG0000018    0
GSHG0000026    0
              ..
GSHG0051591    0
GSHG0051597    0
GSHG0051601    0
GSHG0051602    0
Outcome        0
Length: 16204, dtype: int64
```

```
train_data = train_data.drop(['PCA_STAGE'], axis = 1)
```

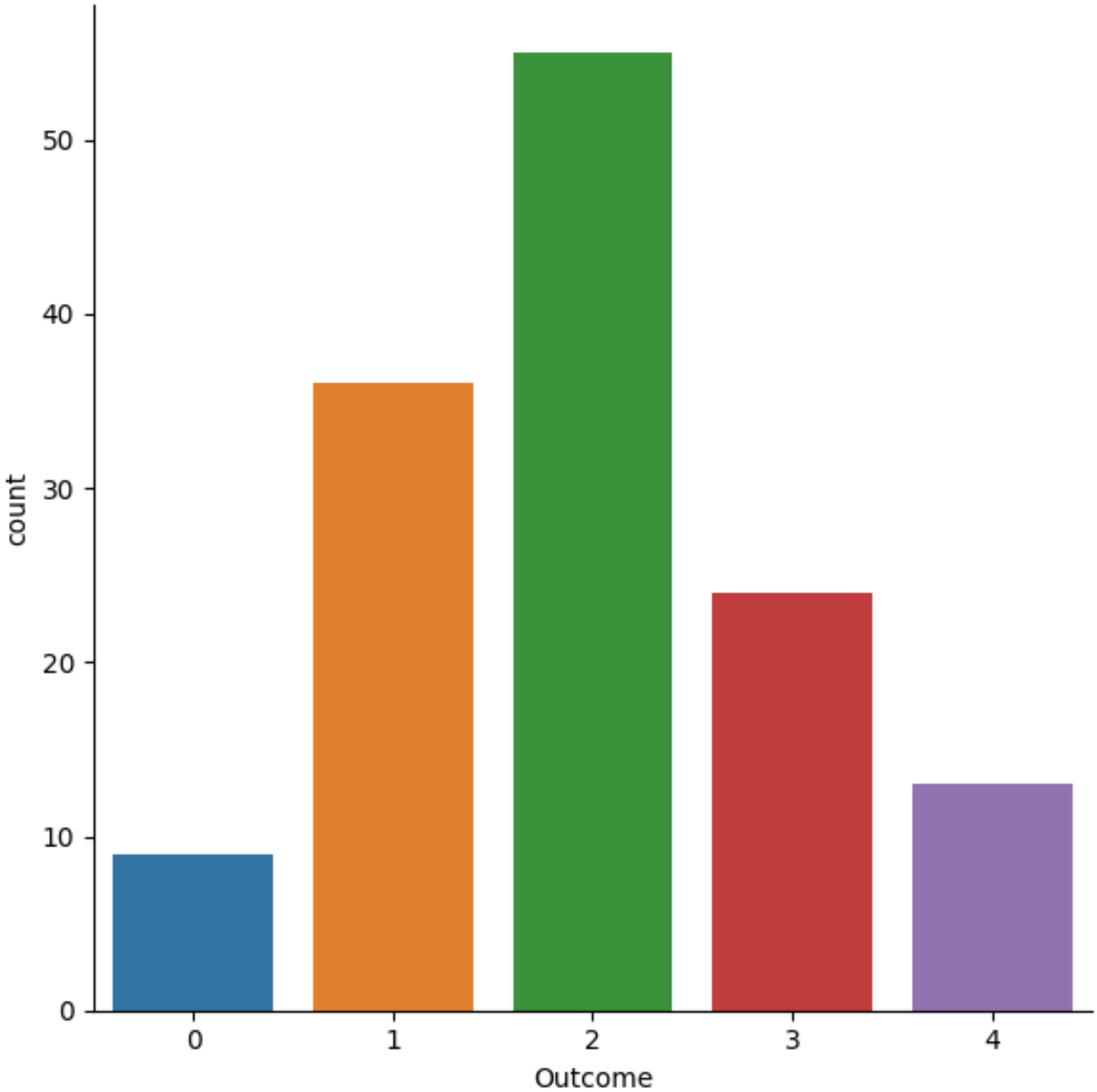
```
train_data.shape
```

```
(137, 16203)
```

```
train_data['Outcome'] = train_data['Outcome'].astype('int')
```

```
# Display counts of classes
sns.catplot(x = 'Outcome', kind = "count", data = train_data, height = 6)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fa3e7a752b0>
```



```
train_data['Outcome'].value_counts()
```

```
2    55
1    36
3    24
4    13
0     9
Name: Outcome, dtype: int64
```

```
# Splitting data into classes
df_0 = train_data[train_data['Outcome'] == 0]
df_1 = train_data[train_data['Outcome'] == 1]
df_2 = train_data[train_data['Outcome'] == 2]
df_3 = train_data[train_data['Outcome'] == 3]
df_4 = train_data[train_data['Outcome'] == 4]
```

```
# Resample using "Bootstrapping" method to regenerate samples by upsampling for
from sklearn.utils import resample
```

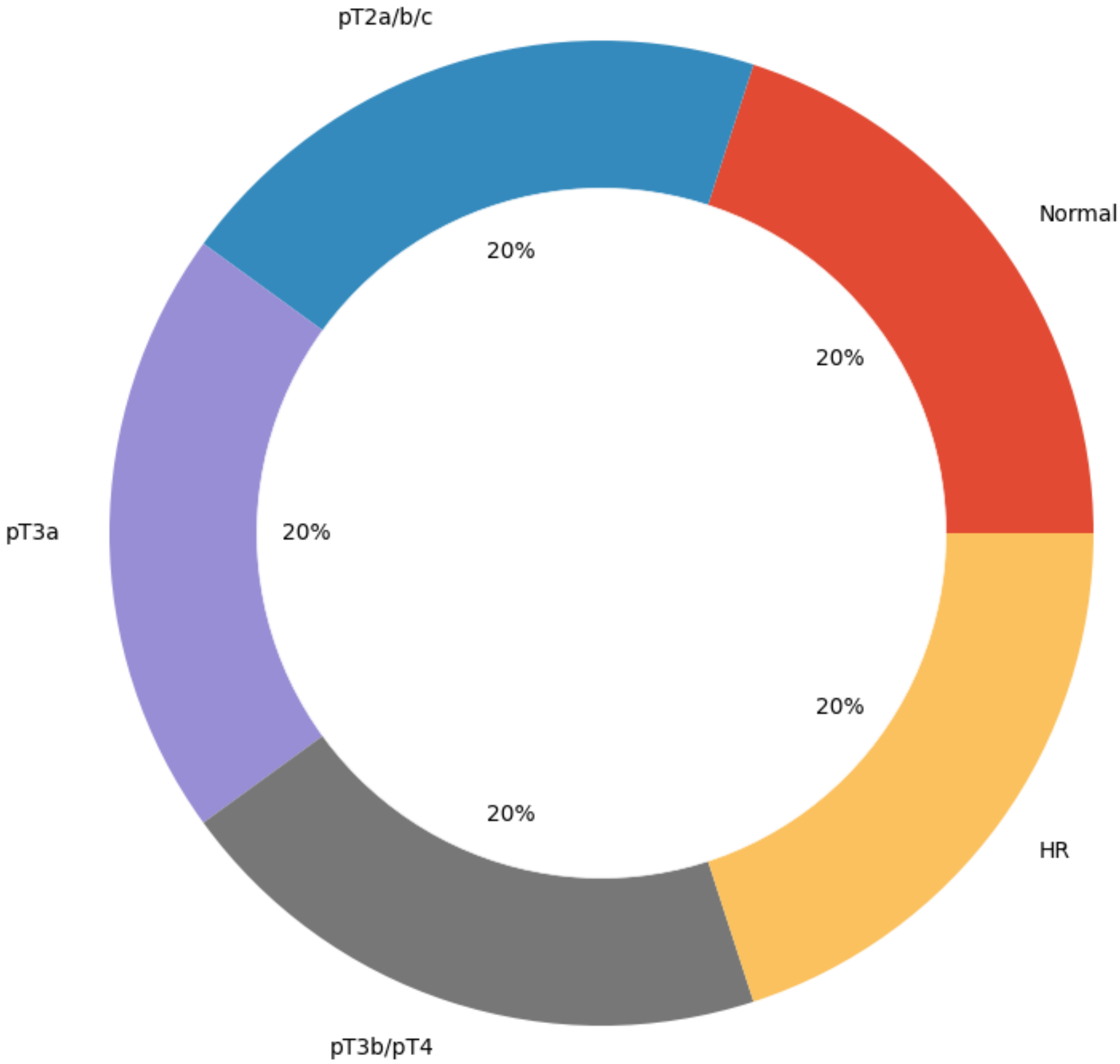
```
df_0_upsample = resample(df_0, n_samples = 100, replace = True, random_state = 1
df_1_upsample = resample(df_1, n_samples = 100, replace = True, random_state = 1
df_2_upsample = resample(df_2, n_samples = 100, replace = True, random_state = 1
df_3_upsample = resample(df_3, n_samples = 100, replace = True, random_state = 1
df_4_upsample = resample(df_4, n_samples = 100, replace = True, random_state = 1
```

```
# Merge all dataframes to create new train samples
train_df = pd.concat([df_0_upsample, df_1_upsample, df_2_upsample, df_3_upsample
```

```
train_df['Outcome'].value_counts()
```

```
0    100
1    100
2    100
3    100
4    100
Name: Outcome, dtype: int64
```

```
plt.style.use('ggplot')
plt.figure(figsize=(10,10))
my_circle = plt.Circle((0,0), 0.7, color = 'white')
plt.pie(train_df['Outcome'].value_counts(), labels = ['Normal','pT2a/b/c','pT3a'
                                                       'pT3b/pT4', 'HR'], autopct = '
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.show()
```




```
X = train_df.drop('Outcome', axis = 1)
Y = train_df['Outcome']
```

```
from sklearn.model_selection import train_test_split
# spliting of training & test is 80% to 20% ratio
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, rando
```

```
x_train.shape
```

```
(400, 16202)
```

```
x_test.shape
```

```
(100, 16202)
```

```
from keras.utils.np_utils import to_categorical
```

```
y_train = to_categorical(y_train)
```

```
y_train
```

```
array([[0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]], dtype=float32)
```

```
y_test = to_categorical(y_test)
```

```
y_test
```

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

```
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
```

```
x_train = x_train.iloc[:, :-1].values
x_test = x_test.iloc[:, :-1].values
```

```
x_train.shape
```

```
(400, 16201)
```

```
x_train = x_train.reshape(len(x_train), x_train.shape[1], 1)
x_test = x_test.reshape(len(x_test), x_test.shape[1], 1)
```

```
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1)
x_test = x_test.reshape( x_test.shape[0],x_train.shape[1], 1)
```

```
x_train.shape
```

```
(400, 16201, 1)
```

```
x_train [0]
```

```
array([[7.03136 ],
       [7.11894 ],
       [8.89482 ],
       ...,
       [7.29462 ],
       [6.25736 ],
       [8.201625]])
```

```
x_test.shape
```

```
(100, 16201, 1)
```

CNN 1D

```
from keras.models import Sequential
from keras.layers import Dense

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.optimizers import Adam
# Avoid Overfitting of NN by Normalizing the samples
from tensorflow.keras.layers import BatchNormalization
```

```
def build_model():
    model = Sequential()
    # Filters = No. of Neurons
    # Padding = 'same' : Zero Padding; Padding = 'valid' : valid padding
    model.add(Conv1D(filters = 64, kernel_size = 5, activation = 'relu', padding
    # BatchNormalization to avoid overfitting
    model.add(BatchNormalization())
    # Pooling
    model.add(MaxPooling1D(pool_size=(2), strides=(2), padding='same'))
```

```
    # Conv Layer - II
    model.add(Conv1D(filters = 64, kernel_size = 5, activation = 'relu', padding
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(2), strides=(2), padding='same'))

    # Conv Layer - III
    model.add(Conv1D(filters = 64, kernel_size = 5, activation = 'relu', padding
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(2), strides=(2), padding='same'))

    # Conv Layer - IV
    model.add(Conv1D(filters = 64, kernel_size = 5, activation = 'relu', padding
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(2), strides=(2), padding='same'))

    # Conv Layer -V
    model.add(Conv1D(filters = 64, kernel_size = 5, activation = 'relu', padding
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(2), strides=(2), padding='same'))

    # Flatten
    model.add(Flatten())

    # Fully Connected Layer (FC - Layer)
    model.add(Dense(units = 64, activation='relu'))
    # Hidden Layer
    model.add(Dense(units = 64, activation='relu'))
    # Output Layer
    model.add(Dense(units = 5, activation='softmax'))

    # loss = 'categorical_crossentropy'
    model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics

    return model

model = build_model()

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 16201, 64)         384

 batch_normalization (BatchN  (None, 16201, 64)        256
 ormalization)

 max_pooling1d (MaxPooling1D  (None, 8101, 64)         0
```

```
 max_pooling1d (MaxPooling1D  (None, 8101, 64)         0
 )

 conv1d_1 (Conv1D)           (None, 8101, 64)          20544

 batch_normalization_1 (Batc  (None, 8101, 64)         256
 hNormalization)

 max_pooling1d_1 (MaxPooling  (None, 4051, 64)         0
 1D)

 conv1d_2 (Conv1D)           (None, 4051, 64)          20544

 batch_normalization_2 (Batc  (None, 4051, 64)         256
 hNormalization)

 max_pooling1d_2 (MaxPooling  (None, 2026, 64)         0
 1D)

 conv1d_3 (Conv1D)           (None, 2026, 64)          20544

 batch_normalization_3 (Batc  (None, 2026, 64)         256
 hNormalization)

 max_pooling1d_3 (MaxPooling  (None, 1013, 64)         0
 1D)

 conv1d_4 (Conv1D)           (None, 1013, 64)          20544

 batch_normalization_4 (Batc  (None, 1013, 64)         256
 hNormalization)

 max_pooling1d_4 (MaxPooling  (None, 507, 64)          0
 1D)

 flatten (Flatten)           (None, 32448)             0

 dense (Dense)               (None, 64)                2076736

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 5)                 325

=================================================================
Total params: 2,165,061
Trainable params: 2,164,421
Non-trainable params: 640
_________________________________________________________________
```

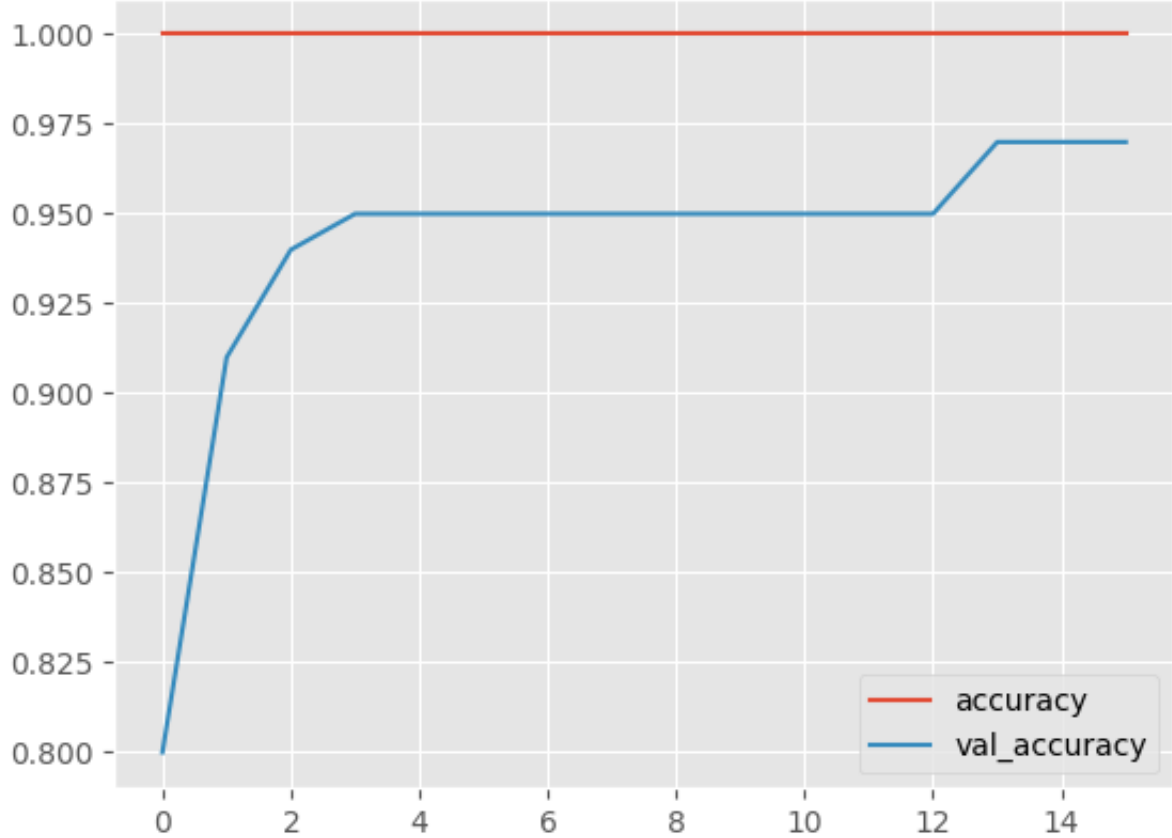
```
history = model.fit(x_train, y_train, epochs = 16, batch_size = 10, validation_d

    Epoch 1/16
    40/40 [==============================] - 48s 1s/step - loss: 5.2275e-04 - a
    Epoch 2/16
    40/40 [==============================] - 55s 1s/step - loss: 3.9035e-04 - a
    Epoch 3/16
    40/40 [==============================] - 47s 1s/step - loss: 3.1947e-04 - a
    Epoch 4/16
    40/40 [==============================] - 51s 1s/step - loss: 2.6571e-04 - a
    Epoch 5/16
    40/40 [==============================] - 48s 1s/step - loss: 2.3767e-04 - a
    Epoch 6/16
    40/40 [==============================] - 48s 1s/step - loss: 2.0314e-04 - a
    Epoch 7/16
    40/40 [==============================] - 57s 1s/step - loss: 1.7630e-04 - a
    Epoch 8/16
    40/40 [==============================] - 49s 1s/step - loss: 1.6090e-04 - a
    Epoch 9/16
    40/40 [==============================] - 49s 1s/step - loss: 1.4755e-04 - a
    Epoch 10/16
    40/40 [==============================] - 48s 1s/step - loss: 1.3630e-04 - a
    Epoch 11/16
    40/40 [==============================] - 51s 1s/step - loss: 1.2363e-04 - a
    Epoch 12/16
    40/40 [==============================] - 48s 1s/step - loss: 1.1339e-04 - a
    Epoch 13/16
    40/40 [==============================] - 48s 1s/step - loss: 1.0222e-04 - a
    Epoch 14/16
    40/40 [==============================] - 46s 1s/step - loss: 9.4306e-05 - a
    Epoch 15/16
    40/40 [==============================] - 55s 1s/step - loss: 8.7878e-05 - a
    Epoch 16/16
    40/40 [==============================] - 47s 1s/step - loss: 8.1671e-05 - a
```

```
pd.DataFrame(history.history)
```

	loss	accuracy	val_loss	val_accuracy
0	0.000523	1.0	0.518467	0.80
1	0.000390	1.0	0.276341	0.91
2	0.000319	1.0	0.171294	0.94
3	0.000266	1.0	0.121712	0.95
4	0.000238	1.0	0.091615	0.95
5	0.000203	1.0	0.075892	0.95
6	0.000176	1.0	0.071458	0.95
7	0.000161	1.0	0.061685	0.95
8	0.000148	1.0	0.055725	0.95
9	0.000136	1.0	0.051812	0.95
10	0.000124	1.0	0.048185	0.95
11	0.000113	1.0	0.047040	0.95
12	0.000102	1.0	0.046185	0.95
13	0.000094	1.0	0.044502	0.97
14	0.000088	1.0	0.042889	0.97
15	0.000082	1.0	0.042211	0.97

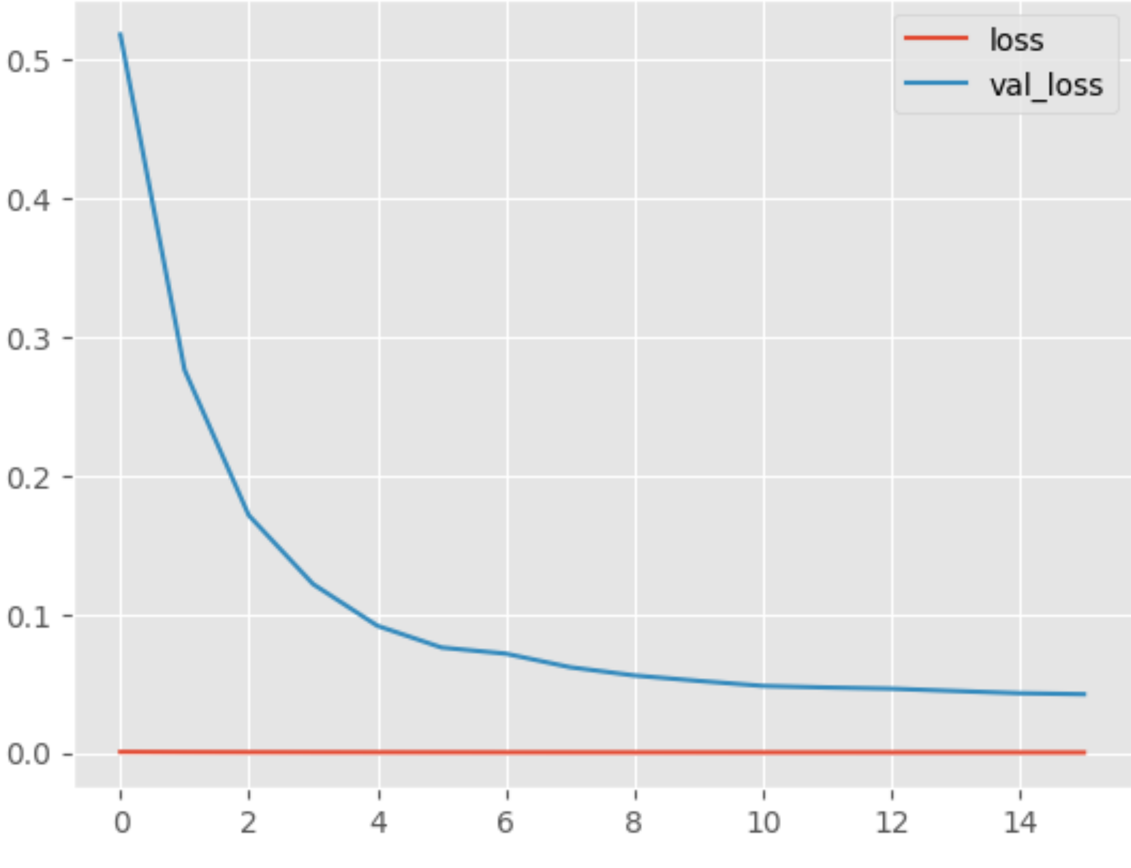
```
pd.DataFrame(history.history)[['accuracy', 'val_accuracy']].plot()
```

```
<Axes: >
```



```
pd.DataFrame(history.history)[['loss','val_loss']].plot()
```

```
<Axes: >
```




```
# Classification Report
```

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

```
4/4 [==============================] - 2s 456ms/step - loss: 0.0422 - accur
[0.04221121221780777, 0.9700000286102295]
```

```
# Make Prediction
predict = model.predict(x_test)
```

```
4/4 [==============================] - 3s 692ms/step
```

```
predict
```

```
array([[6.8529403e-08, 9.9984670e-01, 1.4578918e-04, 7.3713409e-06,
        3.6788908e-08],
       [2.9514680e-07, 3.0521927e-02, 8.6334091e-01, 1.0581854e-01,
        3.1822341e-04],
       [1.0458066e-11, 2.1645706e-12, 1.9805781e-09, 2.3631658e-08,
        9.9999994e-01],
       [1.3193364e-05, 4.5027740e-05, 2.0494021e-05, 9.9992019e-01,
```

```
       [1.3193304e-05, 4.3627740e-05, 2.6494621e-05, 9.9992615e-01,
        1.0968294e-06],
       [6.5663755e-12, 2.5063023e-04, 9.9974120e-01, 8.1707958e-06,
        2.1931088e-09],
       [7.3658521e-07, 1.3303280e-08, 1.5831541e-09, 1.7568649e-05,
        9.9998158e-01],
       [9.9994117e-01, 4.6037243e-05, 8.6687059e-08, 1.2605078e-05,
        4.8008104e-08],
       [3.2180185e-06, 5.9394122e-08, 1.9556994e-07, 9.9279659e-06,
        9.9998659e-01],
       [9.7988583e-08, 4.0553525e-05, 1.3722980e-04, 9.9982214e-01,
        1.4914482e-07],
       [9.2380157e-07, 4.9835530e-06, 3.5884336e-04, 9.9959117e-01,
        4.4095257e-05],
       [1.9514243e-08, 8.0346547e-09, 1.7409697e-09, 2.4224903e-07,
        9.9999970e-01],
       [4.6781338e-06, 1.7896701e-06, 2.9877419e-04, 9.9959320e-01,
        1.0154758e-04],
       [1.2611645e-12, 9.9993438e-01, 6.5615888e-05, 3.4749004e-09,
        7.3020128e-09],
       [1.6292242e-12, 9.9987459e-01, 1.2541404e-04, 4.1680650e-09,
        7.1376027e-10],
       [5.8738151e-06, 1.6277198e-04, 1.7508627e-04, 9.9965346e-01,
        2.9001417e-06],
       [1.6292242e-12, 9.9987459e-01, 1.2541404e-04, 4.1680650e-09,
        7.1376027e-10],
       [1.7470637e-07, 7.1351942e-06, 3.8298502e-05, 9.9995369e-01,
        5.3723249e-07],
       [3.5941619e-07, 2.0689422e-08, 3.6945602e-07, 1.7938764e-05,
        9.9998122e-01],
       [1.7470637e-07, 7.1351942e-06, 3.8298502e-05, 9.9995369e-01,
        5.3723249e-07],
       [1.6292242e-12, 9.9987459e-01, 1.2541404e-04, 4.1680650e-09,
        7.1376027e-10],
       [7.3394398e-07, 4.3589412e-07, 3.8576079e-07, 4.4269982e-05,
        9.9995416e-01],
       [9.9999958e-01, 1.4513687e-07, 9.9868308e-11, 2.8342151e-07,
        2.3779910e-08],
       [5.4721191e-13, 2.9924489e-04, 9.9970055e-01, 2.5902855e-07,
        2.6039909e-10],
       [2.6685723e-09, 9.2089891e-08, 2.4613038e-05, 9.9997503e-01,
        2.0700179e-07],
       [9.9998945e-01, 5.0431713e-06, 2.5645882e-08, 5.4872685e-06,
        4.4501498e-08],
       [1.8264791e-08, 9.9952704e-01, 2.6127449e-04, 2.1159904e-04,
        1.0631021e-08],
       [9.9999768e-01, 1.3861766e-07, 9.7735764e-10, 2.1150956e-06,
        2.5858824e-08],
       [1.1749202e-06, 1.4722697e-06, 5.1012066e-06, 9.9999219e-01,
        5.4844939e-08],
       [4.7306361e-07, 7.9932562e-08, 1.3520250e-08, 4.6217897e-06,
        9.9999470e-01],
       [9.9996561e-01, 1.0494685e-05, 6.5743649e-07, 2.2850169e-05,
        2.0371851e-07]
```

```
yhat = np.argmax(predict, axis = 1)
```

```
# Distributed probability to discrete class
yhat = np.argmax(predict, axis = 1)
yhat
```

```
array([1, 2, 4, 3, 2, 4, 0, 4, 3, 3, 4, 3, 1, 1, 3, 1, 3, 4, 3, 1, 4, 0,
       2, 3, 0, 1, 0, 3, 4, 0, 1, 2, 3, 1, 2, 1, 1, 3, 2, 3, 2, 0, 4, 0,
       0, 4, 4, 2, 1, 4, 4, 3, 2, 2, 4, 4, 0, 0, 4, 3, 4, 1, 1, 2, 4, 2,
       2, 0, 0, 3, 0, 0, 3, 1, 3, 1, 0, 0, 4, 4, 4, 2, 3, 0, 1, 1, 2, 4,
       0, 2, 0, 0, 4, 1, 1, 3, 1, 4, 3, 1])
```

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

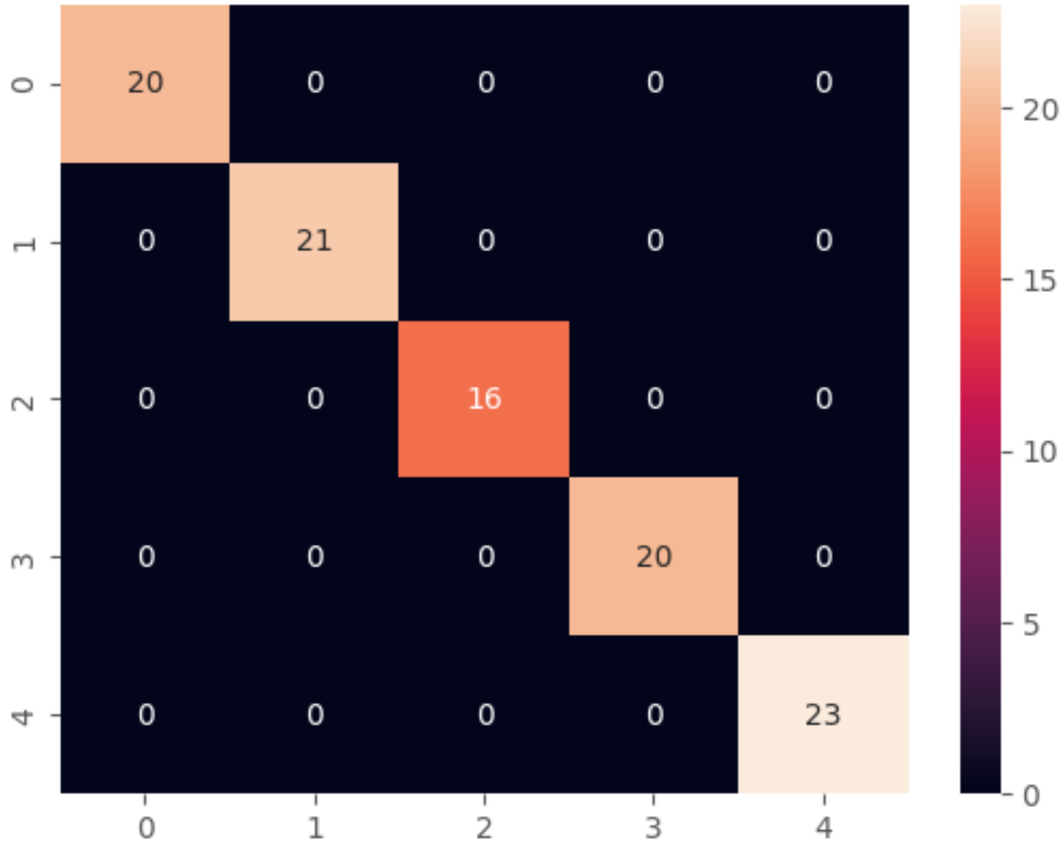
```
from sklearn.metrics import classification_report, confusion_matrix
```

```
confusion_matrix( yhat, y_test)
```

```
array([[20,  0,  0,  0,  0],
       [ 0, 21,  0,  0,  0],
       [ 0,  0, 16,  0,  0],
       [ 0,  0,  0, 20,  0],
       [ 0,  0,  0,  0, 23]])
```

```
sns.heatmap(confusion_matrix(y_test, yhat), annot = True, fmt='0.0f')
```

<Axes: >




```
print(classification_report(yhat, y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        20
           4       1.00      1.00      1.00        23

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```