Problem Statement

Continuing with the same scenario, now that you have been able to successfuly predict each student GPA, now you will classify each Student based on they probability to have a successful GPA score.

The different classes are:

- Low : Students where final GPA is predicted to be between: 0 and 2
- Medium : Students where final GPA is predicted to be between: 2 and 3.5
- High : Students where final GPA is predicted to be between: 3.5 and 5

1) Import Libraries

First let's import the following libraries, if there is any library that you need and is not in the list bellow feel free to include it

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import seaborn as sns
```

2) Load Data

- You will use the same file from the previous activity (Student Performance Data)

In [ ]:
```
data = pd.read_csv("Student_performance_data _.csv")
data
```

Out[ ]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Abser
0	1001	17	1	0	2	19.833723	
1	1002	18	0	0	1	15.408756	
2	1003	15	0	2	3	4.210570	
3	1004	17	1	0	3	10.028829	
4	1005	17	1	0	2	4.672495	
...	...	...	...	...	...	...	
2387	3388	18	1	0	3	10.680555	
2388	3389	17	0	0	1	7.583217	
2389	3390	16	1	0	2	6.805500	
2390	3391	16	1	1	0	12.416653	
2391	3392	16	1	0	2	17.819907	

2392 rows × 15 columns

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   StudentID          2392 non-null   int64  
 1   Age                2392 non-null   int64  
 2   Gender             2392 non-null   int64  
 3   Ethnicity          2392 non-null   int64  
 4   ParentalEducation  2392 non-null   int64  
 5   StudyTimeWeekly    2392 non-null   float64
 6   Absences           2392 non-null   int64  
 7   Tutoring           2392 non-null   int64  
 8   ParentalSupport    2392 non-null   int64  
 9   Extracurricular    2392 non-null   int64  
 10  Sports             2392 non-null   int64  
 11  Music              2392 non-null   int64  
 12  Volunteering       2392 non-null   int64  
 13  GPA                2392 non-null   float64
 14  GradeClass         2392 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB
```

3) Add a new column called 'Profile' this column will have the following information

Based on the value of GPA for each student:

- If GPA values between 0 and 2 will be labeled 'Low',
- Values between 2 and 3.5 will be 'Medium',
- And values between 3.5 and 5 will be 'High'.

In [ ]:
```
intervals = [0, 2, 3.5, 5]
categories = ['Low', 'Medium', 'High']

data['Profile'] = pd.cut(data['GPA'], bins=intervals, labels=categories, rig

data.iloc[:10]
```

Out[ ]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences
0	1001	17	1	0	2	19.833723	7
1	1002	18	0	0	1	15.408756	0
2	1003	15	0	2	3	4.210570	26
3	1004	17	1	0	3	10.028829	14
4	1005	17	1	0	2	4.672495	17
5	1006	18	0	0	1	8.191219	0
6	1007	15	0	1	1	15.601680	10
7	1008	15	1	1	4	15.424496	22
8	1009	17	0	0	0	4.562008	1
9	1010	16	1	0	1	18.444466	0

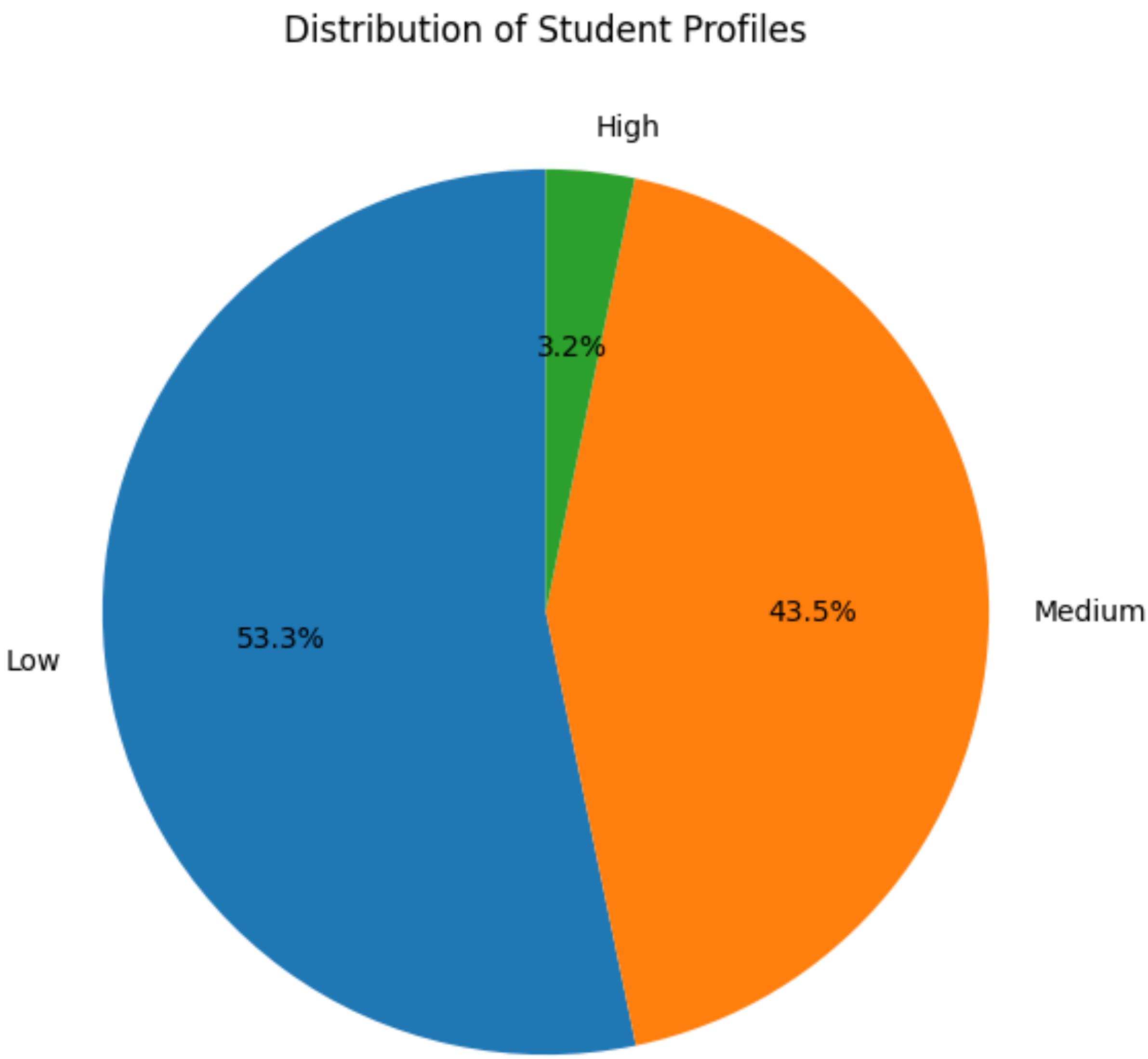
4) Use Matplotlib to show a Pie chart to show the percentage of students in each profile.

- Title: Students distribution of Profiles
- Graph Type: pie

In [ ]:
```
profile_distribution = data['Profile'].value_counts()

plt.figure(figsize=(7, 7))
plt.pie(profile_distribution, labels=profile_distribution.index, autopct='%0
```

```
plt.title('Distribution of Student Profiles')
plt.gca().set_aspect('equal')
plt.show()
```



5) Convert the Profile column into a Categorical Int

You have already created a column with three different values: 'Low', 'Medium', 'High'. These are Categorical values. But, it is important to notice that Neural Networks works better with numbers, since we apply mathematical operations to them.

Next you need to convert Profile values from Low, Medium and High, to 0, 1 and 2. IMPORTANT, the order does not matter, but make sure you always assign the same number to Low, same number to Medium and same number to High.

Make sure to use the fit_transform method from LabelEncoder.

In [ ]:
```
encoder = LabelEncoder()
data['Profile'] = encoder.fit_transform(data['Profile'])
data.iloc[:5]
```

Out[ ]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absences
0	1001	17	1	0	2	19.833723	7
1	1002	18	0	0	1	15.408756	0
2	1003	15	0	2	3	4.210570	26
3	1004	17	1	0	3	10.028829	14
4	1005	17	1	0	2	4.672495	17

6) Select the columns for your model.

Same as the last excersice we need a dataset for features and a dataset for label.

- Create the following dataset:
  - A dataset with the columns for the model.
  - From that data set generate the 'X' dataset. This dataset will have all the features (make sure Profile is NOT in this dataset)
  - Generate a second 'y' dataset, This dataset will only have our label column, which is 'Profile'.
  - Generate the Train and Test datasets for each X and y:
    - X_train with 80% of the data
    - X_test with 20% of the data
    - y_train with 80% of the data
    - y_test with 20% of the data

In [ ]:
```
features = data.drop(['Profile'], axis=1)
target = data['Profile']

X_train, X_test, y_train, y_test = train_test_split(features, target, test_s
```

7) All Feature datasets in the same scale.

Use StandardScaler to make sure all features in the X_train and X_test datasets are on the same scale.

Standardization transforms your data so that it has a mean of 0 and a standard deviation of 1. This is important because many machine learning algorithms perform better when the input features are on a similar scale.

Reason for Using StandardScaler:

- Consistent Scale: Features with different scales (e.g., age in years, income in dollars) can bias the model. StandardScaler ensures all features contribute equally.
- Improved Convergence: Algorithms like gradient descent converge faster with standardized data.
- Regularization: Helps in achieving better performance in regularization methods like Ridge and Lasso regression.

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

8. Define your Deep Neural Network.

- This will be a Sequential Neural Network.
- With a Dense input layer with 64 units, and input dimention based on the X_train size and Relu as the activation function.
- A Dense hidden layer with 32 units, and Relu as the activation function.
- And a Dense output layer with the number of different values in the y dataset, activation function = to sofmax

This last part of the output layer is super important, since we want to do a classification and not a regression, we will use activation functions that fits better a classification scenario.

In [ ]:
```
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

9. Compile your Neural Network

- Choose Adam as the optimizer
- And sparse_categorical_crossentropy as the Loss function
- Also add the following metrics: accuracy

In [ ]:
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy

model.compile(
    optimizer=Adam(),
    loss=SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` r
uns slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, loc
ated at `tf.keras.optimizers.legacy.Adam`.
```

10. Fit (or train) your model

- Use the X_train and y_train datasets for the training
- Do 50 data iterations
- Choose the batch size = 10
- Also select a validation_split of 0.2
- Save the result of the fit function in a variable called 'history'

In [ ]:
```
history = model.fit(
    X_train,
    y_train,
    epochs=50,
    batch_size=10,
    validation_split=0.2,
    verbose=1
)
```

```
Epoch 1/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7881 - loss: 0.6340 -
val_accuracy: 0.9269 - val_loss: 0.2411
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9482 - loss: 0.1904 -
val_accuracy: 0.9399 - val_loss: 0.1628
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9638 - loss: 0.1208 -
val_accuracy: 0.9634 - val_loss: 0.1241
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9660 - loss: 0.0990 -
val_accuracy: 0.9634 - val_loss: 0.0971
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9709 - loss: 0.0773 -
val_accuracy: 0.9661 - val_loss: 0.0866
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9819 - loss: 0.0561 -
val_accuracy: 0.9739 - val_loss: 0.0754
```

```
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9878 - loss: 0.0510 -
val_accuracy: 0.9765 - val_loss: 0.0676
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9887 - loss: 0.0412 -
val_accuracy: 0.9765 - val_loss: 0.0650
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9930 - loss: 0.0318 -
val_accuracy: 0.9817 - val_loss: 0.0558
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9939 - loss: 0.0262 -
val_accuracy: 0.9765 - val_loss: 0.0573
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9960 - loss: 0.0202 -
val_accuracy: 0.9869 - val_loss: 0.0501
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9953 - loss: 0.0200 -
val_accuracy: 0.9765 - val_loss: 0.0572
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9969 - loss: 0.0180 -
val_accuracy: 0.9739 - val_loss: 0.0567
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9963 - loss: 0.0137 -
val_accuracy: 0.9791 - val_loss: 0.0479
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9947 - loss: 0.0150 -
val_accuracy: 0.9791 - val_loss: 0.0510
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9985 - loss: 0.0098 -
val_accuracy: 0.9765 - val_loss: 0.0516
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9983 - loss: 0.0084 -
val_accuracy: 0.9739 - val_loss: 0.0514
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0076 -
val_accuracy: 0.9817 - val_loss: 0.0462
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0049 -
val_accuracy: 0.9791 - val_loss: 0.0464
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0054 -
val_accuracy: 0.9791 - val_loss: 0.0491
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9997 - loss: 0.0045 -
val_accuracy: 0.9791 - val_loss: 0.0471
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0054 -
val_accuracy: 0.9791 - val_loss: 0.0460
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0030 -
```

```
val_accuracy: 0.9765 - val_loss: 0.0489
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0022 -
val_accuracy: 0.9791 - val_loss: 0.0561
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9995 - loss: 0.0039 -
val_accuracy: 0.9791 - val_loss: 0.0464
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0025 -
val_accuracy: 0.9791 - val_loss: 0.0479
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0020 -
val_accuracy: 0.9791 - val_loss: 0.0448
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 0.0013 -
val_accuracy: 0.9791 - val_loss: 0.0501
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0016 -
val_accuracy: 0.9765 - val_loss: 0.0494
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 0.0012 -
val_accuracy: 0.9791 - val_loss: 0.0517
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 9.4939e-
04 - val_accuracy: 0.9765 - val_loss: 0.0491
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 8.9147e-
04 - val_accuracy: 0.9791 - val_loss: 0.0597
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 9.9095e-
04 - val_accuracy: 0.9791 - val_loss: 0.0501
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 8.8120e-
04 - val_accuracy: 0.9817 - val_loss: 0.0512
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 6.5189e-
04 - val_accuracy: 0.9791 - val_loss: 0.0521
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 6.6768e-
04 - val_accuracy: 0.9765 - val_loss: 0.0516
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 6.3192e-
04 - val_accuracy: 0.9791 - val_loss: 0.0527
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 4.6992e-
04 - val_accuracy: 0.9817 - val_loss: 0.0556
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 4.8944e-
04 - val_accuracy: 0.9791 - val_loss: 0.0543
Epoch 40/50
```

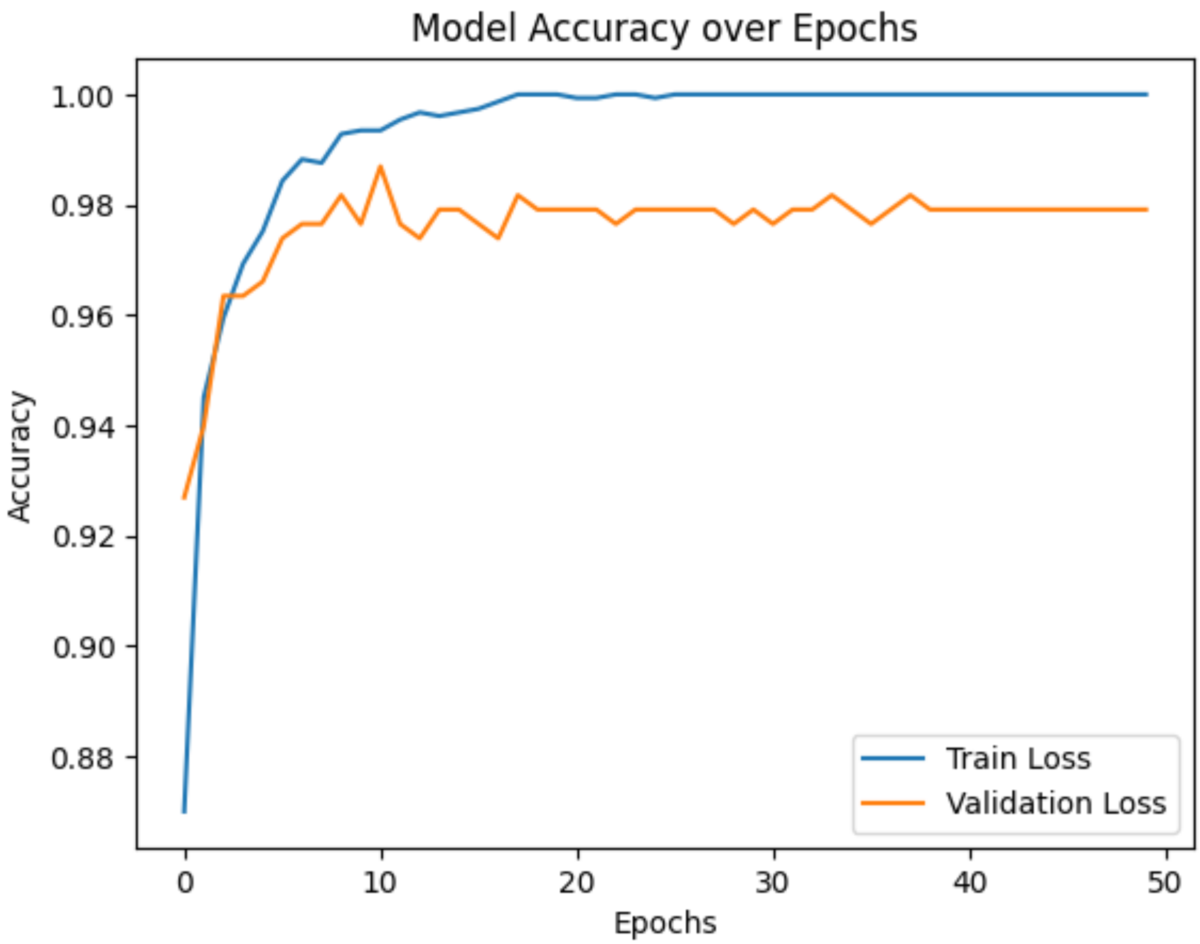
```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 3.2909e-
04 - val_accuracy: 0.9791 - val_loss: 0.0535
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.7859e-
04 - val_accuracy: 0.9791 - val_loss: 0.0562
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 3.0450e-
04 - val_accuracy: 0.9791 - val_loss: 0.0558
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.3448e-
04 - val_accuracy: 0.9791 - val_loss: 0.0549
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.4995e-
04 - val_accuracy: 0.9791 - val_loss: 0.0563
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 1.8776e-
04 - val_accuracy: 0.9791 - val_loss: 0.0583
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 1.7650e-
04 - val_accuracy: 0.9791 - val_loss: 0.0606
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 1.4224e-
04 - val_accuracy: 0.9791 - val_loss: 0.0585
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 2.0514e-
04 - val_accuracy: 0.9791 - val_loss: 0.0615
Epoch 49/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 1.3210e-
04 - val_accuracy: 0.9791 - val_loss: 0.0590
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 1.0000 - loss: 1.2630e-
04 - val_accuracy: 0.9791 - val_loss: 0.0612
```

11. View your history variable:

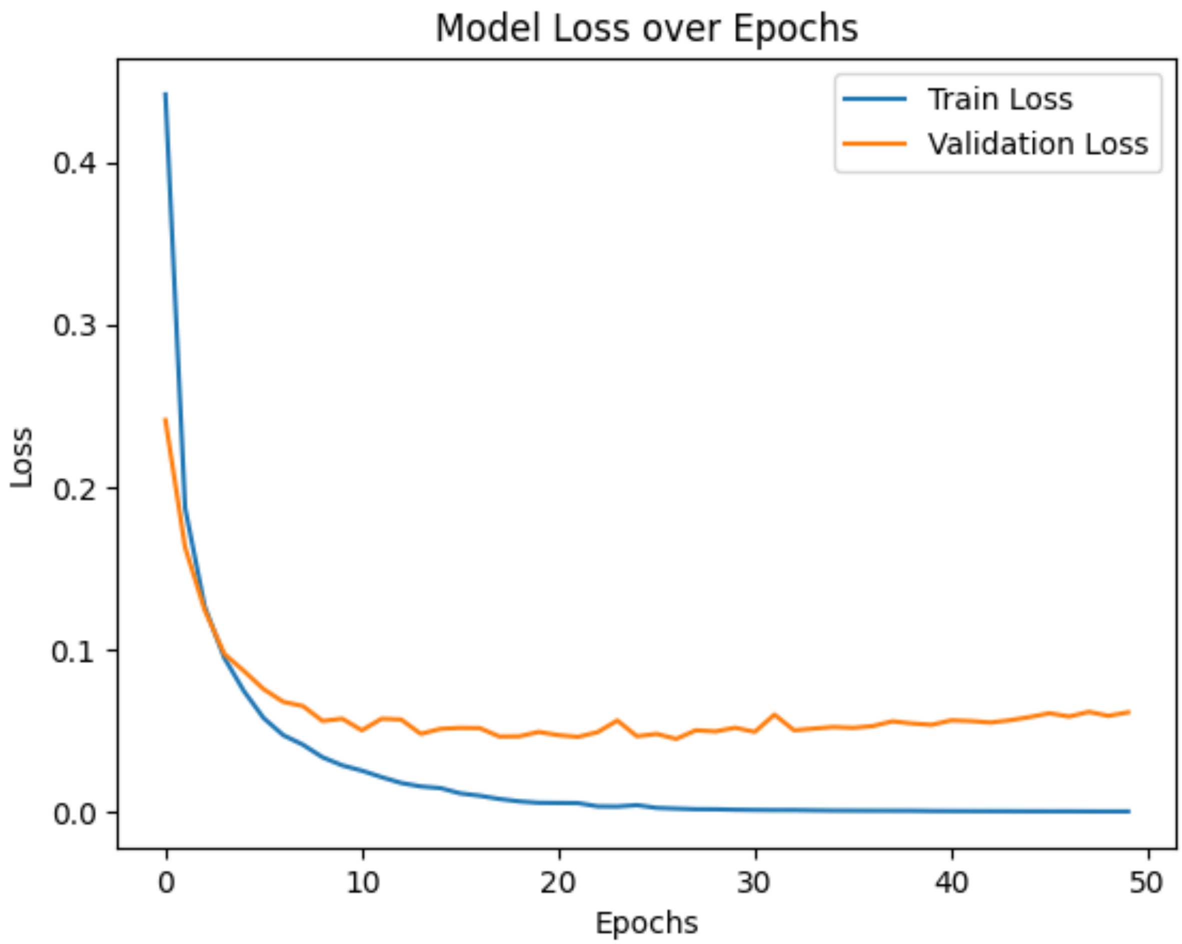
- Use Matplotlib.pyplot to show graphs of your model traning history
- In one graph:
  - Plot the Training Accuracy and the Validation Accuracy
  - X Label = Epochs
  - Y Label = Accuracy
  - Title = Model Accuracy over Epochs
- In a second graph:
  - Plot the Training Loss and the Validation Loss
  - X Label = Epochs
  - Y Label = Loss
  - Title = Model Loss over Epochs

In [ ]:
```
history_df = pd.DataFrame(history.history)

plt.plot(history_df['accuracy'], label='Training Accuracy')
plt.plot(history_df['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Across Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='best')
plt.grid(True)
plt.show()
```



In [ ]:
```
plt.plot(history_df['loss'], label='Training Loss')
plt.plot(history_df['val_loss'], label='Validation Loss')
plt.title('Loss Across Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(axis='y')
plt.show()
```



12. Evaluate your model:

- See the result of your loss function.
- What can you deduct from there?

In [ ]:
```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9861 - loss: 0.0621
loss 0.07819881290197372
accuracy 0.9791231751441956
```

13. Use your model to make some predictions:

- Make predictions of your X_test dataset
- Print the each of the predictions and the actual value (which is in y_test)
- Replace the 'Low', 'Medium' and 'High' to your actual and predicted values.
- How good was your model?

```
In [ ]: predictions = np.round(model.predict(X_test), decimals=0)

        predicted_labels = np.argmax(predictions, axis=1)
        for idx in range(len(y_test)):
            print(f'Predicted: {predicted_labels[idx]} | Actual: {y_test.iloc[idx]}'
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
```

```
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 0
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
```

```
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
```

```
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 0 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 2
Prediction: 1 Actual: 1
```

```
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
```

```
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
```

```
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
```

```
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 0 Actual: 0
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 0
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
```

```
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 0
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 0 Actual: 2
Prediction: 2 Actual: 2
```

```
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 0 Actual: 0
Prediction: 2 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 1 Actual: 1
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 2 Actual: 2
Prediction: 1 Actual: 1
```

14. Compete against this model:

- Create two more different models to compete with this model
- Here are a few ideas of things you can change:
    - During Dataset data engineering:
        - You can remove features that you think do not help in the training and prediction
        - Feature Scaling: Ensure all features are on a similar scale (as you already did with StandardScaler)
    - During Model Definition:

- You can change the Model Architecture (change the type or number of layers or the number of units)
- You can add dropout layers to prevent overfitting
- During Model Compile:
  - You can try other optimizer when compiling your model, here some optimizer samples: Adam, RMSprop, or Adagrad.
  - Try another Loss Function
- During Model Training:
  - Encrease the number of Epochs
  - Adjust the size of your batch
- Explain in a Markdown cell which changes are you implementing
- Show the comparison of your model versus the original model

Model 2:

- Changes:
  - Dataset Data Engineering
  - Model Definition
  - Model Compile
  - Model Training

In [ ]:
```
features2 = data.drop(['Profile', 'Extracurricular', 'Sports', 'Music', 'Vol
target2 = data['Profile']

X2_train, X2_test, y2_train, y2_test = train_test_split(features2, target2,

scaler2 = StandardScaler()
X2_train_scaled = scaler2.fit_transform(X2_train)
X2_test_scaled = scaler2.transform(X2_test)

model2 = Sequential()
model2.add(Dense(64, activation='relu', input_dim=X2_train_scaled.shape[1]))
model2.add(Dropout(0.2))
model2.add(Dense(32, activation='relu'))
model2.add(Dense(3, activation='softmax'))

model2.compile(optimizer=Adam(), loss=SparseCategoricalCrossentropy(), metri

history2 = model2.fit(X2_train_scaled, y2_train, epochs=50, batch_size=10, v

final_loss, final_accuracy = model2.evaluate(X2_test_scaled, y2_test, verbos
print(f"Test Loss: {final_loss}")
print(f"Test Accuracy: {final_accuracy}")
```

```
Epoch 1/50
c:\Users\oskga\AppData\Local\Programs\Python\Python312\Lib\site-packages\ker
as\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`i
nput_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
153/153 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7315 - loss: 0.6705 -
val_accuracy: 0.9295 - val_loss: 0.2436
Epoch 2/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9318 - loss: 0.2341 -
val_accuracy: 0.9373 - val_loss: 0.1647
Epoch 3/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9467 - loss: 0.1658 -
val_accuracy: 0.9582 - val_loss: 0.1361
Epoch 4/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9585 - loss: 0.1350 -
val_accuracy: 0.9608 - val_loss: 0.1070
Epoch 5/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9618 - loss: 0.1194 -
val_accuracy: 0.9713 - val_loss: 0.0950
Epoch 6/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9619 - loss: 0.0931 -
val_accuracy: 0.9661 - val_loss: 0.0805
Epoch 7/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9658 - loss: 0.0901 -
val_accuracy: 0.9739 - val_loss: 0.0769
Epoch 8/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9779 - loss: 0.0731 -
val_accuracy: 0.9713 - val_loss: 0.0719
Epoch 9/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9801 - loss: 0.0639 -
val_accuracy: 0.9713 - val_loss: 0.0657
Epoch 10/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9699 - loss: 0.0786 -
val_accuracy: 0.9765 - val_loss: 0.0627
Epoch 11/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9851 - loss: 0.0459 -
val_accuracy: 0.9791 - val_loss: 0.0633
Epoch 12/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9875 - loss: 0.0516 -
val_accuracy: 0.9791 - val_loss: 0.0611
Epoch 13/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9824 - loss: 0.0504 -
val_accuracy: 0.9791 - val_loss: 0.0576
Epoch 14/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9826 - loss: 0.0509 -
val_accuracy: 0.9817 - val_loss: 0.0534
Epoch 15/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9834 - loss: 0.0449 -
val_accuracy: 0.9791 - val_loss: 0.0604
```

```
Epoch 16/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9803 - loss: 0.0502 -
val_accuracy: 0.9713 - val_loss: 0.0627
Epoch 17/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9893 - loss: 0.0375 -
val_accuracy: 0.9843 - val_loss: 0.0417
Epoch 18/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9821 - loss: 0.0385 -
val_accuracy: 0.9765 - val_loss: 0.0465
Epoch 19/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9920 - loss: 0.0279 -
val_accuracy: 0.9817 - val_loss: 0.0446
Epoch 20/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9857 - loss: 0.0326 -
val_accuracy: 0.9843 - val_loss: 0.0398
Epoch 21/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9886 - loss: 0.0345 -
val_accuracy: 0.9843 - val_loss: 0.0375
Epoch 22/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9853 - loss: 0.0457 -
val_accuracy: 0.9869 - val_loss: 0.0330
Epoch 23/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9863 - loss: 0.0309 -
val_accuracy: 0.9869 - val_loss: 0.0341
Epoch 24/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9902 - loss: 0.0300 -
val_accuracy: 0.9869 - val_loss: 0.0376
Epoch 25/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9927 - loss: 0.0229 -
val_accuracy: 0.9843 - val_loss: 0.0389
Epoch 26/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9940 - loss: 0.0163 -
val_accuracy: 0.9896 - val_loss: 0.0311
Epoch 27/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9931 - loss: 0.0199 -
val_accuracy: 0.9817 - val_loss: 0.0381
Epoch 28/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9919 - loss: 0.0236 -
val_accuracy: 0.9896 - val_loss: 0.0211
Epoch 29/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9933 - loss: 0.0193 -
val_accuracy: 0.9922 - val_loss: 0.0243
Epoch 30/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9938 - loss: 0.0167 -
val_accuracy: 0.9948 - val_loss: 0.0184
Epoch 31/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9911 - loss: 0.0256 -
val_accuracy: 0.9922 - val_loss: 0.0206
Epoch 32/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9957 - loss: 0.0173 -
```

```
val_accuracy: 0.9869 - val_loss: 0.0270
Epoch 33/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9950 - loss: 0.0188 -
val_accuracy: 0.9896 - val_loss: 0.0238
Epoch 34/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9932 - loss: 0.0155 -
val_accuracy: 0.9922 - val_loss: 0.0180
Epoch 35/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9967 - loss: 0.0164 -
val_accuracy: 0.9922 - val_loss: 0.0178
Epoch 36/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9979 - loss: 0.0092 -
val_accuracy: 0.9922 - val_loss: 0.0212
Epoch 37/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9942 - loss: 0.0191 -
val_accuracy: 0.9896 - val_loss: 0.0192
Epoch 38/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9958 - loss: 0.0103 -
val_accuracy: 0.9948 - val_loss: 0.0143
Epoch 39/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9925 - loss: 0.0197 -
val_accuracy: 0.9896 - val_loss: 0.0226
Epoch 40/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9978 - loss: 0.0122 -
val_accuracy: 0.9948 - val_loss: 0.0156
Epoch 41/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9976 - loss: 0.0087 -
val_accuracy: 0.9922 - val_loss: 0.0161
Epoch 42/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9967 - loss: 0.0092 -
val_accuracy: 0.9922 - val_loss: 0.0192
Epoch 43/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9937 - loss: 0.0140 -
val_accuracy: 0.9922 - val_loss: 0.0174
Epoch 44/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9963 - loss: 0.0133 -
val_accuracy: 0.9869 - val_loss: 0.0258
Epoch 45/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9975 - loss: 0.0099 -
val_accuracy: 0.9922 - val_loss: 0.0128
Epoch 46/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9991 - loss: 0.0085 -
val_accuracy: 0.9948 - val_loss: 0.0086
Epoch 47/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9963 - loss: 0.0112 -
val_accuracy: 0.9948 - val_loss: 0.0134
Epoch 48/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9966 - loss: 0.0091 -
val_accuracy: 0.9948 - val_loss: 0.0130
Epoch 49/50
```

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9981 - loss: 0.0091 -
val_accuracy: 0.9922 - val_loss: 0.0164
Epoch 50/50
153/153 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9957 - loss: 0.0120 -
val_accuracy: 0.9948 - val_loss: 0.0122
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9727 - loss: 0.0562
loss 0.047554709017276764
accuracy 0.9791231751441956
```

Model 3:

- Changes:
  - Dataset Data Engineering
  - Model Definition
  - Model Compile
  - Model Training

In [ ]:
```
features3 = data.drop(['Profile', 'Extracurricular', 'Sports', 'Music', 'Vol
target3 = data['Profile']

X3_train, X3_test, y3_train, y3_test = train_test_split(features3, target3,

scaler3 = StandardScaler()
X3_train_scaled = scaler3.fit_transform(X3_train)
X3_test_scaled = scaler3.transform(X3_test)

model3 = Sequential()
model3.add(Dense(64, activation='relu', input_dim=X3_train_scaled.shape[1]))
model3.add(Dropout(0.25))
model3.add(Dense(32, activation='relu'))
model3.add(Dense(3, activation='softmax'))

model3.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', met

history3 = model3.fit(
    X3_train_scaled,
    y3_train,
    epochs=75,
    batch_size=10,
    validation_split=0.25,
    verbose=1
)

final_loss3, final_accuracy3 = model3.evaluate(X3_test_scaled, y3_test, verb
print(f"Final Loss: {final_loss3}")
print(f"Final Accuracy: {final_accuracy3}")
```

```
Epoch 1/75
```

```
c:\Users\oskga\AppData\Local\Programs\Python\Python312\Lib\site-packages\ker
as\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`i
nput_dim` argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
144/144 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6713 - loss: 0.7277 -
val_accuracy: 0.9228 - val_loss: 0.2671
Epoch 2/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9358 - loss: 0.2443 -
val_accuracy: 0.9436 - val_loss: 0.1880
Epoch 3/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9519 - loss: 0.1648 -
val_accuracy: 0.9582 - val_loss: 0.1510
Epoch 4/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9496 - loss: 0.1491 -
val_accuracy: 0.9666 - val_loss: 0.1230
Epoch 5/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9597 - loss: 0.1277 -
val_accuracy: 0.9582 - val_loss: 0.1079
Epoch 6/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9542 - loss: 0.1248 -
val_accuracy: 0.9687 - val_loss: 0.0840
Epoch 7/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9653 - loss: 0.0935 -
val_accuracy: 0.9687 - val_loss: 0.0799
Epoch 8/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9630 - loss: 0.0868 -
val_accuracy: 0.9770 - val_loss: 0.0778
Epoch 9/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9702 - loss: 0.0852 -
val_accuracy: 0.9749 - val_loss: 0.0667
Epoch 10/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9719 - loss: 0.0775 -
val_accuracy: 0.9812 - val_loss: 0.0708
Epoch 11/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9664 - loss: 0.0675 -
val_accuracy: 0.9770 - val_loss: 0.0567
Epoch 12/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9758 - loss: 0.0687 -
val_accuracy: 0.9812 - val_loss: 0.0548
Epoch 13/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9792 - loss: 0.0552 -
val_accuracy: 0.9770 - val_loss: 0.0553
Epoch 14/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9837 - loss: 0.0449 -
val_accuracy: 0.9791 - val_loss: 0.0482
Epoch 15/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9791 - loss: 0.0703 -
val_accuracy: 0.9833 - val_loss: 0.0513
Epoch 16/75
```

```
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9805 - loss: 0.0465 -
val_accuracy: 0.9812 - val_loss: 0.0503
Epoch 17/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9867 - loss: 0.0381 -
val_accuracy: 0.9833 - val_loss: 0.0455
Epoch 18/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9876 - loss: 0.0505 -
val_accuracy: 0.9812 - val_loss: 0.0476
Epoch 19/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9848 - loss: 0.0455 -
val_accuracy: 0.9812 - val_loss: 0.0499
Epoch 20/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9829 - loss: 0.0448 -
val_accuracy: 0.9875 - val_loss: 0.0351
Epoch 21/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9815 - loss: 0.0457 -
val_accuracy: 0.9875 - val_loss: 0.0364
Epoch 22/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9801 - loss: 0.0494 -
val_accuracy: 0.9896 - val_loss: 0.0351
Epoch 23/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9909 - loss: 0.0298 -
val_accuracy: 0.9916 - val_loss: 0.0354
Epoch 24/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9771 - loss: 0.0462 -
val_accuracy: 0.9896 - val_loss: 0.0395
Epoch 25/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9854 - loss: 0.0335 -
val_accuracy: 0.9937 - val_loss: 0.0316
Epoch 26/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9830 - loss: 0.0354 -
val_accuracy: 0.9875 - val_loss: 0.0331
Epoch 27/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9912 - loss: 0.0286 -
val_accuracy: 0.9875 - val_loss: 0.0293
Epoch 28/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9913 - loss: 0.0275 -
val_accuracy: 0.9896 - val_loss: 0.0299
Epoch 29/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9851 - loss: 0.0376 -
val_accuracy: 0.9916 - val_loss: 0.0283
Epoch 30/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9952 - loss: 0.0188 -
val_accuracy: 0.9812 - val_loss: 0.0415
Epoch 31/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9910 - loss: 0.0269 -
val_accuracy: 0.9875 - val_loss: 0.0262
Epoch 32/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9890 - loss: 0.0283 -
val_accuracy: 0.9937 - val_loss: 0.0255
```

```
Epoch 33/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9843 - loss: 0.0319 -
val_accuracy: 0.9875 - val_loss: 0.0224
Epoch 34/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9928 - loss: 0.0217 -
val_accuracy: 0.9916 - val_loss: 0.0233
Epoch 35/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9929 - loss: 0.0227 -
val_accuracy: 0.9916 - val_loss: 0.0213
Epoch 36/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9903 - loss: 0.0263 -
val_accuracy: 0.9937 - val_loss: 0.0217
Epoch 37/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9872 - loss: 0.0267 -
val_accuracy: 0.9937 - val_loss: 0.0182
Epoch 38/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9936 - loss: 0.0218 -
val_accuracy: 0.9958 - val_loss: 0.0163
Epoch 39/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9927 - loss: 0.0196 -
val_accuracy: 0.9958 - val_loss: 0.0167
Epoch 40/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9972 - loss: 0.0153 -
val_accuracy: 0.9979 - val_loss: 0.0160
Epoch 41/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9930 - loss: 0.0176 -
val_accuracy: 0.9896 - val_loss: 0.0201
Epoch 42/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9950 - loss: 0.0201 -
val_accuracy: 0.9937 - val_loss: 0.0159
Epoch 43/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9960 - loss: 0.0125 -
val_accuracy: 0.9916 - val_loss: 0.0190
Epoch 44/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9966 - loss: 0.0114 -
val_accuracy: 0.9937 - val_loss: 0.0159
Epoch 45/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9899 - loss: 0.0162 -
val_accuracy: 0.9958 - val_loss: 0.0175
Epoch 46/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9970 - loss: 0.0146 -
val_accuracy: 0.9979 - val_loss: 0.0141
Epoch 47/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9958 - loss: 0.0136 -
val_accuracy: 0.9916 - val_loss: 0.0157
Epoch 48/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9913 - loss: 0.0198 -
val_accuracy: 0.9958 - val_loss: 0.0142
Epoch 49/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9933 - loss: 0.0175 -
```

```
val_accuracy: 0.9937 - val_loss: 0.0155
Epoch 50/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9939 - loss: 0.0192 -
val_accuracy: 0.9896 - val_loss: 0.0184
Epoch 51/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9961 - loss: 0.0128 -
val_accuracy: 0.9958 - val_loss: 0.0127
Epoch 52/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9944 - loss: 0.0167 -
val_accuracy: 0.9937 - val_loss: 0.0154
Epoch 53/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9971 - loss: 0.0136 -
val_accuracy: 0.9958 - val_loss: 0.0114
Epoch 54/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9983 - loss: 0.0083 -
val_accuracy: 0.9937 - val_loss: 0.0129
Epoch 55/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9980 - loss: 0.0079 -
val_accuracy: 0.9937 - val_loss: 0.0147
Epoch 56/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9922 - loss: 0.0171 -
val_accuracy: 0.9979 - val_loss: 0.0110
Epoch 57/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9974 - loss: 0.0097 -
val_accuracy: 0.9979 - val_loss: 0.0084
Epoch 58/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9919 - loss: 0.0203 -
val_accuracy: 0.9958 - val_loss: 0.0094
Epoch 59/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9926 - loss: 0.0164 -
val_accuracy: 0.9958 - val_loss: 0.0142
Epoch 60/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9947 - loss: 0.0166 -
val_accuracy: 0.9958 - val_loss: 0.0131
Epoch 61/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9892 - loss: 0.0271 -
val_accuracy: 0.9958 - val_loss: 0.0094
Epoch 62/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9959 - loss: 0.0120 -
val_accuracy: 0.9979 - val_loss: 0.0090
Epoch 63/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9946 - loss: 0.0127 -
val_accuracy: 0.9979 - val_loss: 0.0080
Epoch 64/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9974 - loss: 0.0088 -
val_accuracy: 0.9979 - val_loss: 0.0082
Epoch 65/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9978 - loss: 0.0083 -
val_accuracy: 0.9958 - val_loss: 0.0112
Epoch 66/75
```

```
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9988 - loss: 0.0091 -
val_accuracy: 0.9979 - val_loss: 0.0077
Epoch 67/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9991 - loss: 0.0074 -
val_accuracy: 0.9979 - val_loss: 0.0074
Epoch 68/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9973 - loss: 0.0120 -
val_accuracy: 0.9979 - val_loss: 0.0074
Epoch 69/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9976 - loss: 0.0095 -
val_accuracy: 0.9979 - val_loss: 0.0061
Epoch 70/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9990 - loss: 0.0059 -
val_accuracy: 0.9958 - val_loss: 0.0096
Epoch 71/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9965 - loss: 0.0095 -
val_accuracy: 0.9979 - val_loss: 0.0075
Epoch 72/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9954 - loss: 0.0126 -
val_accuracy: 0.9937 - val_loss: 0.0122
Epoch 73/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9972 - loss: 0.0065 -
val_accuracy: 0.9937 - val_loss: 0.0080
Epoch 74/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9972 - loss: 0.0098 -
val_accuracy: 0.9979 - val_loss: 0.0046
Epoch 75/75
144/144 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9972 - loss: 0.0078 -
val_accuracy: 0.9958 - val_loss: 0.0082
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9793 - loss: 0.0702
loss 0.04870617017149925
accuracy 0.9853861927986145
```

In [ ]:

```
model2 = Sequential([
    Dense(64, activation='relu', input_shape=(X2_train.shape[1],)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(3, activation='softmax')
])

model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

model2.fit(X2_train, y2_train, epochs=50, batch_size=10, validation_split=0.
model3 = Sequential([
    Dense(64, activation='relu', input_shape=(X3_train.shape[1],)),
    Dropout(0.25),
    Dense(32, activation='relu'),
    Dense(3, activation='softmax')
])
```

```
model3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', met

model3.fit(X3_train, y3_train, epochs=75, batch_size=10, validation_split=0.

X2 = data.drop(columns=['Profile', 'Extracurricular', 'Sports', 'Music', 'Vo
y2 = data['Profile']

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.

scaler2 = StandardScaler()
X2_train = scaler2.fit_transform(X2_train)
X2_test = scaler2.transform(X2_test)

X3 = data.drop(columns=['Profile', 'Extracurricular', 'Sports', 'Music', 'Vo
y3 = data['Profile']

X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.

scaler3 = StandardScaler()
X3_train = scaler3.fit_transform(X3_train)
X3_test = scaler3.transform(X3_test)



X_test_sample1 = X_test[:5]
y_test_sample1 = y_test[:5]

X2_test_sample = X2_test[:5]
X3_test_sample = X3_test[:5]


preds_model1 = model.predict(X_test_sample1)
preds_model2 = model2.predict(X2_test_sample)
preds_model3 = model3.predict(X3_test_sample)


pred_classes_model1 = np.argmax(preds_model1, axis=1)
pred_classes_model2 = np.argmax(preds_model2, axis=1)
pred_classes_model3 = np.argmax(preds_model3, axis=1)
results_df = pd.DataFrame({
    'Actual Values': y_test_sample1.tolist() if isinstance(y_test_sample1, r
    'Model 1 Prediction': pred_classes_model1,
    'Model 2 Prediction': pred_classes_model2,
    'Model 3 Prediction': pred_classes_model3
})


print(results_df)
```

```
Epoch 1/50
```

```
153/153 [==============================] - 0s 1ms/step - loss: 0.5496 - accu
racy: 0.8288 - val_loss: 0.2750 - val_accuracy: 0.9191
Epoch 2/50
153/153 [==============================] - 0s 939us/step - loss: 0.2268 - ac
curacy: 0.9379 - val_loss: 0.1734 - val_accuracy: 0.9452
Epoch 3/50
153/153 [==============================] - 0s 795us/step - loss: 0.1548 - ac
curacy: 0.9490 - val_loss: 0.1246 - val_accuracy: 0.9634
Epoch 4/50
153/153 [==============================] - 0s 731us/step - loss: 0.1192 - ac
curacy: 0.9569 - val_loss: 0.1086 - val_accuracy: 0.9608
Epoch 5/50
153/153 [==============================] - 0s 705us/step - loss: 0.1007 - ac
curacy: 0.9641 - val_loss: 0.0881 - val_accuracy: 0.9582
Epoch 6/50
153/153 [==============================] - 0s 700us/step - loss: 0.0909 - ac
curacy: 0.9654 - val_loss: 0.0762 - val_accuracy: 0.9713
Epoch 7/50
153/153 [==============================] - 0s 712us/step - loss: 0.0793 - ac
curacy: 0.9745 - val_loss: 0.0757 - val_accuracy: 0.9817
Epoch 8/50
153/153 [==============================] - 0s 720us/step - loss: 0.0769 - ac
curacy: 0.9732 - val_loss: 0.0700 - val_accuracy: 0.9817
Epoch 9/50
153/153 [==============================] - 0s 705us/step - loss: 0.0623 - ac
curacy: 0.9817 - val_loss: 0.0624 - val_accuracy: 0.9869
Epoch 10/50
153/153 [==============================] - 0s 711us/step - loss: 0.0613 - ac
curacy: 0.9758 - val_loss: 0.0612 - val_accuracy: 0.9713
Epoch 11/50
153/153 [==============================] - 0s 699us/step - loss: 0.0614 - ac
curacy: 0.9791 - val_loss: 0.0541 - val_accuracy: 0.9817
Epoch 12/50
153/153 [==============================] - 0s 700us/step - loss: 0.0501 - ac
curacy: 0.9856 - val_loss: 0.0536 - val_accuracy: 0.9791
Epoch 13/50
153/153 [==============================] - 0s 707us/step - loss: 0.0589 - ac
curacy: 0.9758 - val_loss: 0.0485 - val_accuracy: 0.9869
Epoch 14/50
153/153 [==============================] - 0s 691us/step - loss: 0.0485 - ac
curacy: 0.9817 - val_loss: 0.0503 - val_accuracy: 0.9791
Epoch 15/50
153/153 [==============================] - 0s 721us/step - loss: 0.0466 - ac
curacy: 0.9850 - val_loss: 0.0443 - val_accuracy: 0.9896
Epoch 16/50
153/153 [==============================] - 0s 746us/step - loss: 0.0381 - ac
curacy: 0.9876 - val_loss: 0.0442 - val_accuracy: 0.9843
Epoch 17/50
153/153 [==============================] - 0s 771us/step - loss: 0.0378 - ac
curacy: 0.9856 - val_loss: 0.0469 - val_accuracy: 0.9896
```

```
Epoch 18/50
153/153 [==============================] - 0s 762us/step - loss: 0.0340 - ac
curacy: 0.9876 - val_loss: 0.0530 - val_accuracy: 0.9843
Epoch 19/50
153/153 [==============================] - 0s 697us/step - loss: 0.0394 - ac
curacy: 0.9843 - val_loss: 0.0406 - val_accuracy: 0.9896
Epoch 20/50
153/153 [==============================] - 0s 693us/step - loss: 0.0346 - ac
curacy: 0.9889 - val_loss: 0.0380 - val_accuracy: 0.9869
Epoch 21/50
153/153 [==============================] - 0s 697us/step - loss: 0.0298 - ac
curacy: 0.9902 - val_loss: 0.0359 - val_accuracy: 0.9869
Epoch 22/50
153/153 [==============================] - 0s 747us/step - loss: 0.0301 - ac
curacy: 0.9869 - val_loss: 0.0422 - val_accuracy: 0.9791
Epoch 23/50
153/153 [==============================] - 0s 728us/step - loss: 0.0221 - ac
curacy: 0.9922 - val_loss: 0.0292 - val_accuracy: 0.9896
Epoch 24/50
153/153 [==============================] - 0s 703us/step - loss: 0.0282 - ac
curacy: 0.9895 - val_loss: 0.0424 - val_accuracy: 0.9817
Epoch 25/50
153/153 [==============================] - 0s 732us/step - loss: 0.0267 - ac
curacy: 0.9902 - val_loss: 0.0294 - val_accuracy: 0.9896
Epoch 26/50
153/153 [==============================] - 0s 691us/step - loss: 0.0256 - ac
curacy: 0.9902 - val_loss: 0.0277 - val_accuracy: 0.9869
Epoch 27/50
153/153 [==============================] - 0s 692us/step - loss: 0.0235 - ac
curacy: 0.9908 - val_loss: 0.0255 - val_accuracy: 0.9896
Epoch 28/50
153/153 [==============================] - 0s 693us/step - loss: 0.0177 - ac
curacy: 0.9928 - val_loss: 0.0255 - val_accuracy: 0.9922
Epoch 29/50
153/153 [==============================] - 0s 691us/step - loss: 0.0212 - ac
curacy: 0.9922 - val_loss: 0.0300 - val_accuracy: 0.9896
Epoch 30/50
153/153 [==============================] - 0s 686us/step - loss: 0.0197 - ac
curacy: 0.9915 - val_loss: 0.0223 - val_accuracy: 0.9948
Epoch 31/50
153/153 [==============================] - 0s 736us/step - loss: 0.0169 - ac
curacy: 0.9948 - val_loss: 0.0290 - val_accuracy: 0.9869
Epoch 32/50
153/153 [==============================] - 0s 689us/step - loss: 0.0143 - ac
curacy: 0.9961 - val_loss: 0.0263 - val_accuracy: 0.9896
Epoch 33/50
153/153 [==============================] - 0s 689us/step - loss: 0.0147 - ac
curacy: 0.9954 - val_loss: 0.0196 - val_accuracy: 0.9922
Epoch 34/50
153/153 [==============================] - 0s 690us/step - loss: 0.0176 - ac
```

```
curacy: 0.9928 - val_loss: 0.0254 - val_accuracy: 0.9922
Epoch 35/50
153/153 [==============================] - 0s 690us/step - loss: 0.0133 - ac
curacy: 0.9941 - val_loss: 0.0169 - val_accuracy: 0.9948
Epoch 36/50
153/153 [==============================] - 0s 690us/step - loss: 0.0235 - ac
curacy: 0.9922 - val_loss: 0.0156 - val_accuracy: 0.9974
Epoch 37/50
153/153 [==============================] - 0s 690us/step - loss: 0.0136 - ac
curacy: 0.9948 - val_loss: 0.0158 - val_accuracy: 0.9948
Epoch 38/50
153/153 [==============================] - 0s 692us/step - loss: 0.0125 - ac
curacy: 0.9948 - val_loss: 0.0150 - val_accuracy: 0.9974
Epoch 39/50
153/153 [==============================] - 0s 690us/step - loss: 0.0095 - ac
curacy: 0.9967 - val_loss: 0.0177 - val_accuracy: 0.9922
Epoch 40/50
153/153 [==============================] - 0s 691us/step - loss: 0.0180 - ac
curacy: 0.9954 - val_loss: 0.0139 - val_accuracy: 0.9974
Epoch 41/50
153/153 [==============================] - 0s 690us/step - loss: 0.0138 - ac
curacy: 0.9954 - val_loss: 0.0124 - val_accuracy: 0.9974
Epoch 42/50
153/153 [==============================] - 0s 688us/step - loss: 0.0109 - ac
curacy: 0.9961 - val_loss: 0.0131 - val_accuracy: 0.9948
Epoch 43/50
153/153 [==============================] - 0s 684us/step - loss: 0.0110 - ac
curacy: 0.9974 - val_loss: 0.0209 - val_accuracy: 0.9896
Epoch 44/50
153/153 [==============================] - 0s 691us/step - loss: 0.0095 - ac
curacy: 0.9961 - val_loss: 0.0165 - val_accuracy: 0.9922
Epoch 45/50
153/153 [==============================] - 0s 688us/step - loss: 0.0084 - ac
curacy: 0.9974 - val_loss: 0.0098 - val_accuracy: 0.9974
Epoch 46/50
153/153 [==============================] - 0s 690us/step - loss: 0.0094 - ac
curacy: 0.9967 - val_loss: 0.0111 - val_accuracy: 1.0000
Epoch 47/50
153/153 [==============================] - 0s 693us/step - loss: 0.0117 - ac
curacy: 0.9967 - val_loss: 0.0172 - val_accuracy: 0.9948
Epoch 48/50
153/153 [==============================] - 0s 693us/step - loss: 0.0157 - ac
curacy: 0.9935 - val_loss: 0.0163 - val_accuracy: 0.9948
Epoch 49/50
153/153 [==============================] - 0s 806us/step - loss: 0.0091 - ac
curacy: 0.9987 - val_loss: 0.0135 - val_accuracy: 0.9974
Epoch 50/50
153/153 [==============================] - 0s 685us/step - loss: 0.0100 - ac
curacy: 0.9954 - val_loss: 0.0096 - val_accuracy: 0.9974
Epoch 1/75
```

```
144/144 [==============================] - 0s 1ms/step - loss: 0.5906 - accu
racy: 0.7887 - val_loss: 0.2882 - val_accuracy: 0.9061
Epoch 2/75
144/144 [==============================] - 0s 756us/step - loss: 0.2579 - ac
curacy: 0.9219 - val_loss: 0.1935 - val_accuracy: 0.9457
Epoch 3/75
144/144 [==============================] - 0s 741us/step - loss: 0.1926 - ac
curacy: 0.9344 - val_loss: 0.1408 - val_accuracy: 0.9457
Epoch 4/75
144/144 [==============================] - 0s 739us/step - loss: 0.1515 - ac
curacy: 0.9421 - val_loss: 0.1110 - val_accuracy: 0.9645
Epoch 5/75
144/144 [==============================] - 0s 740us/step - loss: 0.1278 - ac
curacy: 0.9547 - val_loss: 0.0895 - val_accuracy: 0.9770
Epoch 6/75
144/144 [==============================] - 0s 740us/step - loss: 0.1082 - ac
curacy: 0.9623 - val_loss: 0.0804 - val_accuracy: 0.9812
Epoch 7/75
144/144 [==============================] - 0s 741us/step - loss: 0.1079 - ac
curacy: 0.9609 - val_loss: 0.0757 - val_accuracy: 0.9833
Epoch 8/75
144/144 [==============================] - 0s 742us/step - loss: 0.0881 - ac
curacy: 0.9651 - val_loss: 0.0664 - val_accuracy: 0.9875
Epoch 9/75
144/144 [==============================] - 0s 743us/step - loss: 0.0825 - ac
curacy: 0.9742 - val_loss: 0.0619 - val_accuracy: 0.9770
Epoch 10/75
144/144 [==============================] - 0s 738us/step - loss: 0.0737 - ac
curacy: 0.9742 - val_loss: 0.0529 - val_accuracy: 0.9812
Epoch 11/75
144/144 [==============================] - 0s 738us/step - loss: 0.0713 - ac
curacy: 0.9728 - val_loss: 0.0497 - val_accuracy: 0.9812
Epoch 12/75
144/144 [==============================] - 0s 738us/step - loss: 0.0708 - ac
curacy: 0.9742 - val_loss: 0.0453 - val_accuracy: 0.9854
Epoch 13/75
144/144 [==============================] - 0s 734us/step - loss: 0.0631 - ac
curacy: 0.9756 - val_loss: 0.0527 - val_accuracy: 0.9833
Epoch 14/75
144/144 [==============================] - 0s 734us/step - loss: 0.0580 - ac
curacy: 0.9847 - val_loss: 0.0462 - val_accuracy: 0.9854
Epoch 15/75
144/144 [==============================] - 0s 738us/step - loss: 0.0547 - ac
curacy: 0.9777 - val_loss: 0.0423 - val_accuracy: 0.9875
Epoch 16/75
144/144 [==============================] - 0s 736us/step - loss: 0.0546 - ac
curacy: 0.9812 - val_loss: 0.0360 - val_accuracy: 0.9833
Epoch 17/75
144/144 [==============================] - 0s 737us/step - loss: 0.0514 - ac
curacy: 0.9777 - val_loss: 0.0487 - val_accuracy: 0.9916
```

```
Epoch 18/75
144/144 [==============================] - 0s 739us/step - loss: 0.0470 - ac
curacy: 0.9854 - val_loss: 0.0374 - val_accuracy: 0.9896
Epoch 19/75
144/144 [==============================] - 0s 737us/step - loss: 0.0518 - ac
curacy: 0.9756 - val_loss: 0.0378 - val_accuracy: 0.9896
Epoch 20/75
144/144 [==============================] - 0s 735us/step - loss: 0.0497 - ac
curacy: 0.9805 - val_loss: 0.0349 - val_accuracy: 0.9896
Epoch 21/75
144/144 [==============================] - 0s 736us/step - loss: 0.0426 - ac
curacy: 0.9847 - val_loss: 0.0321 - val_accuracy: 0.9916
Epoch 22/75
144/144 [==============================] - 0s 736us/step - loss: 0.0453 - ac
curacy: 0.9833 - val_loss: 0.0287 - val_accuracy: 0.9937
Epoch 23/75
144/144 [==============================] - 0s 736us/step - loss: 0.0369 - ac
curacy: 0.9895 - val_loss: 0.0249 - val_accuracy: 0.9896
Epoch 24/75
144/144 [==============================] - 0s 740us/step - loss: 0.0398 - ac
curacy: 0.9840 - val_loss: 0.0266 - val_accuracy: 0.9937
Epoch 25/75
144/144 [==============================] - 0s 732us/step - loss: 0.0315 - ac
curacy: 0.9895 - val_loss: 0.0238 - val_accuracy: 0.9958
Epoch 26/75
144/144 [==============================] - 0s 733us/step - loss: 0.0309 - ac
curacy: 0.9923 - val_loss: 0.0222 - val_accuracy: 0.9937
Epoch 27/75
144/144 [==============================] - 0s 738us/step - loss: 0.0293 - ac
curacy: 0.9923 - val_loss: 0.0238 - val_accuracy: 0.9937
Epoch 28/75
144/144 [==============================] - 0s 735us/step - loss: 0.0336 - ac
curacy: 0.9861 - val_loss: 0.0305 - val_accuracy: 0.9937
Epoch 29/75
144/144 [==============================] - 0s 735us/step - loss: 0.0306 - ac
curacy: 0.9888 - val_loss: 0.0216 - val_accuracy: 0.9958
Epoch 30/75
144/144 [==============================] - 0s 738us/step - loss: 0.0294 - ac
curacy: 0.9895 - val_loss: 0.0183 - val_accuracy: 0.9916
Epoch 31/75
144/144 [==============================] - 0s 732us/step - loss: 0.0277 - ac
curacy: 0.9902 - val_loss: 0.0189 - val_accuracy: 0.9979
Epoch 32/75
144/144 [==============================] - 0s 733us/step - loss: 0.0257 - ac
curacy: 0.9888 - val_loss: 0.0299 - val_accuracy: 0.9916
Epoch 33/75
144/144 [==============================] - 0s 740us/step - loss: 0.0319 - ac
curacy: 0.9874 - val_loss: 0.0200 - val_accuracy: 0.9937
Epoch 34/75
144/144 [==============================] - 0s 739us/step - loss: 0.0205 - ac
```

```
curacy: 0.9930 - val_loss: 0.0181 - val_accuracy: 0.9958
Epoch 35/75
144/144 [==============================] - 0s 734us/step - loss: 0.0179 - ac
curacy: 0.9951 - val_loss: 0.0188 - val_accuracy: 0.9937
Epoch 36/75
144/144 [==============================] - 0s 737us/step - loss: 0.0241 - ac
curacy: 0.9923 - val_loss: 0.0139 - val_accuracy: 0.9958
Epoch 37/75
144/144 [==============================] - 0s 735us/step - loss: 0.0187 - ac
curacy: 0.9965 - val_loss: 0.0123 - val_accuracy: 0.9958
Epoch 38/75
144/144 [==============================] - 0s 736us/step - loss: 0.0174 - ac
curacy: 0.9944 - val_loss: 0.0134 - val_accuracy: 0.9958
Epoch 39/75
144/144 [==============================] - 0s 731us/step - loss: 0.0223 - ac
curacy: 0.9965 - val_loss: 0.0115 - val_accuracy: 0.9958
Epoch 40/75
144/144 [==============================] - 0s 733us/step - loss: 0.0111 - ac
curacy: 0.9972 - val_loss: 0.0177 - val_accuracy: 0.9958
Epoch 41/75
144/144 [==============================] - 0s 862us/step - loss: 0.0188 - ac
curacy: 0.9902 - val_loss: 0.0103 - val_accuracy: 1.0000
Epoch 42/75
144/144 [==============================] - 0s 738us/step - loss: 0.0114 - ac
curacy: 0.9958 - val_loss: 0.0087 - val_accuracy: 0.9979
Epoch 43/75
144/144 [==============================] - 0s 734us/step - loss: 0.0148 - ac
curacy: 0.9937 - val_loss: 0.0092 - val_accuracy: 1.0000
Epoch 44/75
144/144 [==============================] - 0s 732us/step - loss: 0.0120 - ac
curacy: 0.9972 - val_loss: 0.0122 - val_accuracy: 0.9958
Epoch 45/75
144/144 [==============================] - 0s 742us/step - loss: 0.0206 - ac
curacy: 0.9923 - val_loss: 0.0106 - val_accuracy: 0.9979
Epoch 46/75
144/144 [==============================] - 0s 739us/step - loss: 0.0098 - ac
curacy: 0.9986 - val_loss: 0.0110 - val_accuracy: 0.9937
Epoch 47/75
144/144 [==============================] - 0s 736us/step - loss: 0.0107 - ac
curacy: 0.9979 - val_loss: 0.0122 - val_accuracy: 0.9937
Epoch 48/75
144/144 [==============================] - 0s 740us/step - loss: 0.0081 - ac
curacy: 0.9986 - val_loss: 0.0105 - val_accuracy: 0.9958
Epoch 49/75
144/144 [==============================] - 0s 740us/step - loss: 0.0149 - ac
curacy: 0.9916 - val_loss: 0.0092 - val_accuracy: 0.9979
Epoch 50/75
144/144 [==============================] - 0s 736us/step - loss: 0.0117 - ac
curacy: 0.9937 - val_loss: 0.0124 - val_accuracy: 0.9958
Epoch 51/75
```

```
144/144 [==============================] - 0s 735us/step - loss: 0.0103 - ac
curacy: 0.9972 - val_loss: 0.0079 - val_accuracy: 0.9979
Epoch 52/75
144/144 [==============================] - 0s 739us/step - loss: 0.0118 - ac
curacy: 0.9951 - val_loss: 0.0089 - val_accuracy: 0.9979
Epoch 53/75
144/144 [==============================] - 0s 734us/step - loss: 0.0137 - ac
curacy: 0.9958 - val_loss: 0.0073 - val_accuracy: 0.9979
Epoch 54/75
144/144 [==============================] - 0s 738us/step - loss: 0.0083 - ac
curacy: 0.9972 - val_loss: 0.0077 - val_accuracy: 0.9958
Epoch 55/75
144/144 [==============================] - 0s 736us/step - loss: 0.0080 - ac
curacy: 0.9986 - val_loss: 0.0095 - val_accuracy: 0.9979
Epoch 56/75
144/144 [==============================] - 0s 737us/step - loss: 0.0072 - ac
curacy: 0.9979 - val_loss: 0.0067 - val_accuracy: 0.9979
Epoch 57/75
144/144 [==============================] - 0s 738us/step - loss: 0.0096 - ac
curacy: 0.9958 - val_loss: 0.0052 - val_accuracy: 1.0000
Epoch 58/75
144/144 [==============================] - 0s 736us/step - loss: 0.0073 - ac
curacy: 0.9986 - val_loss: 0.0058 - val_accuracy: 0.9979
Epoch 59/75
144/144 [==============================] - 0s 736us/step - loss: 0.0054 - ac
curacy: 0.9986 - val_loss: 0.0089 - val_accuracy: 0.9958
Epoch 60/75
144/144 [==============================] - 0s 733us/step - loss: 0.0120 - ac
curacy: 0.9951 - val_loss: 0.0105 - val_accuracy: 0.9979
Epoch 61/75
144/144 [==============================] - 0s 738us/step - loss: 0.0106 - ac
curacy: 0.9958 - val_loss: 0.0076 - val_accuracy: 0.9958
Epoch 62/75
144/144 [==============================] - 0s 738us/step - loss: 0.0068 - ac
curacy: 0.9979 - val_loss: 0.0090 - val_accuracy: 0.9958
Epoch 63/75
144/144 [==============================] - 0s 735us/step - loss: 0.0080 - ac
curacy: 0.9979 - val_loss: 0.0108 - val_accuracy: 0.9958
Epoch 64/75
144/144 [==============================] - 0s 733us/step - loss: 0.0097 - ac
curacy: 0.9965 - val_loss: 0.0084 - val_accuracy: 0.9958
Epoch 65/75
144/144 [==============================] - 0s 733us/step - loss: 0.0056 - ac
curacy: 0.9986 - val_loss: 0.0099 - val_accuracy: 0.9958
Epoch 66/75
144/144 [==============================] - 0s 739us/step - loss: 0.0085 - ac
curacy: 0.9965 - val_loss: 0.0069 - val_accuracy: 0.9979
Epoch 67/75
144/144 [==============================] - 0s 732us/step - loss: 0.0051 - ac
curacy: 0.9986 - val_loss: 0.0086 - val_accuracy: 0.9979
```

```
Epoch 68/75
144/144 [==============================] - 0s 735us/step - loss: 0.0112 - ac
curacy: 0.9965 - val_loss: 0.0064 - val_accuracy: 0.9979
Epoch 69/75
144/144 [==============================] - 0s 732us/step - loss: 0.0091 - ac
curacy: 0.9972 - val_loss: 0.0121 - val_accuracy: 0.9958
Epoch 70/75
144/144 [==============================] - 0s 734us/step - loss: 0.0061 - ac
curacy: 0.9986 - val_loss: 0.0053 - val_accuracy: 0.9979
Epoch 71/75
144/144 [==============================] - 0s 738us/step - loss: 0.0082 - ac
curacy: 0.9979 - val_loss: 0.0046 - val_accuracy: 0.9979
Epoch 72/75
144/144 [==============================] - 0s 740us/step - loss: 0.0131 - ac
curacy: 0.9958 - val_loss: 0.0065 - val_accuracy: 0.9979
Epoch 73/75
144/144 [==============================] - 0s 735us/step - loss: 0.0064 - ac
curacy: 0.9986 - val_loss: 0.0049 - val_accuracy: 1.0000
Epoch 74/75
144/144 [==============================] - 0s 734us/step - loss: 0.0066 - ac
curacy: 0.9979 - val_loss: 0.0077 - val_accuracy: 0.9958
Epoch 75/75
144/144 [==============================] - 0s 732us/step - loss: 0.0118 - ac
curacy: 0.9951 - val_loss: 0.0101 - val_accuracy: 0.9958
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 26ms/step
WARNING:tensorflow:5 out of the last 10 calls to <function Model.make_predic
t_function.<locals>.predict_function at 0x345ba8280> triggered tf.function r
etracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors w
ith different shapes, (3) passing Python objects instead of tensors. For (
1), please define your @tf.function outside of the loop. For (2), @tf.functi
on has reduce_retracing=True option that can avoid unnecessary retracing. Fo
r (3), please refer to https://www.tensorflow.org/guide/function#controlling
_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  m
ore details.
```


```
WARNING:tensorflow:5 out of the last 10 calls to <function Model.make_predic
t_function.<locals>.predict_function at 0x345ba8280> triggered tf.function r
etracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors w
ith different shapes, (3) passing Python objects instead of tensors. For (
1), please define your @tf.function outside of the loop. For (2), @tf.functi
on has reduce_retracing=True option that can avoid unnecessary retracing. Fo
r (3), please refer to https://www.tensorflow.org/guide/function#controlling
_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  m
ore details.
```

```
1/1 [==============================] - 0s 28ms/step
   Actual Values  Model 1 Prediction  Model 2 Prediction  Model 3 Prediction
0              1                   2                   1                   1
1              2                   1                   2                   2
2              2                   1                   2                   2
3              0                   2                   0                   0
4              1                   2                   1                   1
```