Problem Statement

You are a data scientist working for a school

You are asked to predict the GPA of the current students based on the following provided data:

0 StudentID int64
1 Age int64
2 Gender int64
3 Ethnicity int64
4 ParentalEducation int64
5 StudyTimeWeekly float64 6 Absences int64
7 Tutoring int64
8 ParentalSupport int64
9 Extracurricular int64
10 Sports int64
11 Music int64
12 Volunteering int64
13 GPA float64 14 GradeClass float64

The GPA is the Grade Point Average, typically ranges from 0.0 to 4.0 in most educational systems, with 4.0 representing an 'A' or excellent performance.

The minimum passing GPA can vary by institution, but it's often around 2.0. This usually corresponds to a 'C' grade, which is considered satisfactory.

You need to create a Deep Learning model capable to predict the GPA of a Student based on a set of provided features. The data provided represents 2,392 students.

In this excersice you will be requested to create a total of three models and select the most performant one.

In [32]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

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, Dropout, BatchNormalization
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
```

2) Load Data

- You will be provided with a cvs (comma separated value) file.
- You will need to add that file into a pandas dataframe, you can use the following code as reference
- The file will be available in canvas

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

Out[ ]:

	StudentID	Age	Gender	Ethnicity	ParentalEducation	StudyTimeWeekly	Absenc
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

3) Review you data:

Make sure you review your data. Place special attention of null or empty values.

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
```

4. Remove the columns not needed for Student performance prediction

- Choose only the columns you consider to be valuable for your model training.
- For example, StudentID might not be a good feature for your model, and thus should be removed from your main dataset, which other columns should also be removed?
- You can name that final dataset as 'dataset'

In [ ]:
```
# Your code here
data.drop(columns=['StudentID', 'Ethnicity', 'Volunteering', 'Gender'], inplace=
```

5. Check if the columns has any null values:

- Here you now have your final dataset to use in your model training.
- Before moving foward review your data check for any null or empty value that might be needed to be removed

In [ ]:
```
# Your code here
data.isnull().sum()
```

Out[ ]:

	0
Age	0
ParentalEducation	0
StudyTimeWeekly	0
Absences	0
Tutoring	0
ParentalSupport	0
Extracurricular	0
Sports	0
Music	0
GPA	0
GradeClass	0

dtype: int64

6. Prepare your data for training and for testing set:

- First create a dataset named X, with all columns but GPA. These are the |features
- Next create another dataset named y, with only GPA column. This is the label
- If you go to your Imports, you will see the following import: **'from sklearn.model_selection import train_test_split'**
- Use that *train_test_split* function to create: X_train, X_test, y_train and y_test respectively. Use X and y datasets as parameters. Other parameters to use are: Test Size = 0.2, Random State = 42.
- Standarize your features (X_train and X_test) by using the StandardScaler (investigate how to use fit_transform and transform functions). This will help the training process by dealing with normilized data.

Note: Your X_train shape should be around (1913, 10). This means the dataset has 10 columns which should be the input.

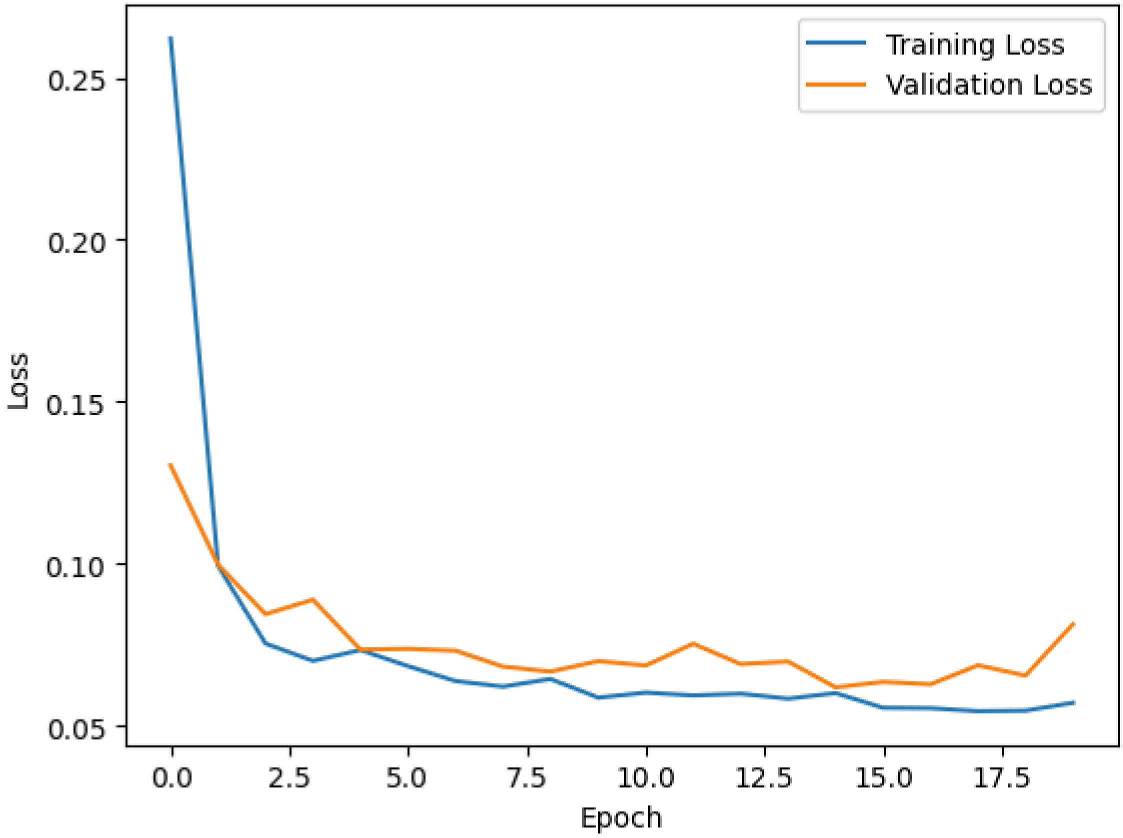
In [ ]:
```
# Your code here
X_train, X_test, y_train, y_test = train_test_split(data.drop(columns=['GPA']),
```

In [ ]:
```
model_loss_data = []
```

In [ ]:
```
input_shape = X_train.shape[1]

model_one = Sequential()
model_one.add(Dense(64, input_dim=input_shape, activation='relu'))
model_one.add(Dense(32, activation='relu'))
```

```
model_one.add(Dense(1))
model_one.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model_one.fit(X_train, y_train, epochs=20, batch_size=20, validation_s
loss, mae = model_one.evaluate(X_test, y_test)
model_loss_data.append({"Model": "Model 1", "loss": loss, "mae": mae, "val_loss"
print(model_loss_data)

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0717 - mae: 0.2069
[{'Model': 'Model 1', 'loss': 0.07223042100667953, 'mae': 0.2117377519607544, 'va
l_loss': 0.08148396760225296, 'val_mae': 0.23174971342086792}]
```

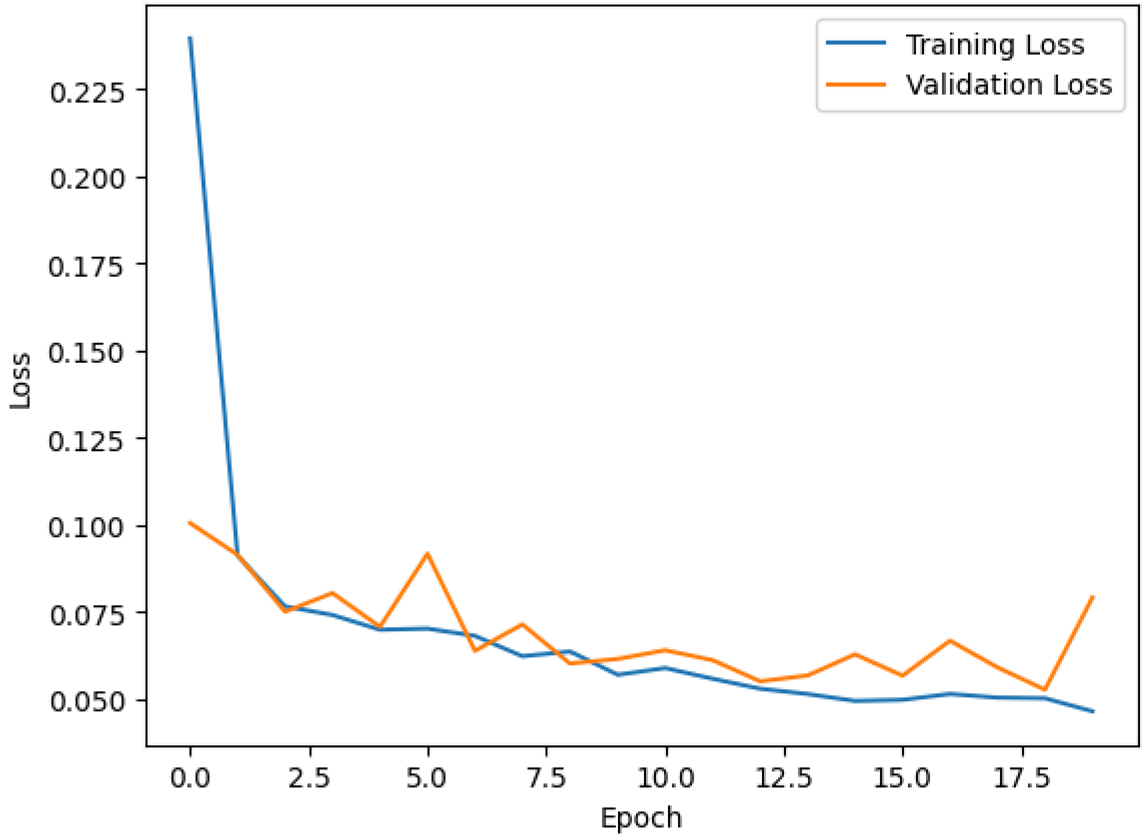


In [ ]:

```
input_shape = X_train.shape[1]

model_two = Sequential()
model_two.add(Dense(64, input_dim=input_shape, activation='relu'))
model_two.add(Dense(32, activation='relu'))
model_two.add(Dense(16, activation='relu'))
model_two.add(Dense(8, activation='relu'))
model_two.add(Dense(1))
model_two.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model_two.fit(X_train, y_train, epochs=20, batch_size=20, validation_s
loss, mae = model_two.evaluate(X_test, y_test)
```

```
model_loss_data.append({"Model": "Model 2", "loss": loss, "mae": mae, "val_loss"
print(model_loss_data)

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

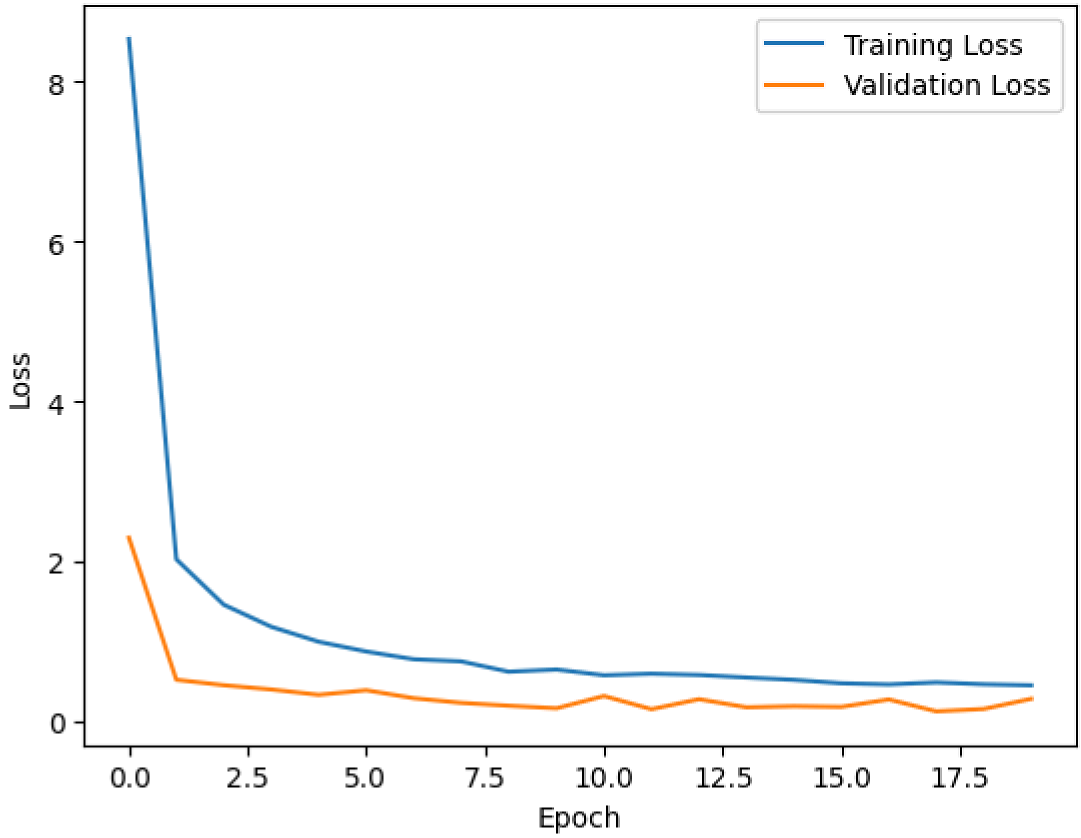
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0733 - mae: 0.2169
[{'Model': 'Model 1', 'loss': 0.07223042100667953, 'mae': 0.2117377519607544, 'va
l_loss': 0.08148396760225296, 'val_mae': 0.23174971342086792}, {'Model': 'Model
2', 'loss': 0.07398424297571182, 'mae': 0.21946609020233154, 'val_loss': 0.079194
99278068542, 'val_mae': 0.2290734201669693}]
```



In [ ]:
```
input_shape = X_train.shape[1]

model_three = Sequential()
model_three.add(Dense(64, input_dim=input_shape, activation='relu'))
model_three.add(Dense(32, activation='relu'))
model_three.add(Dropout(0.3))
model_three.add(Dense(16, activation='relu'))
model_three.add(Dropout(0.3))
model_three.add(Dense(8, activation='relu'))
model_three.add(Dropout(0.3))
model_three.add(Dense(1))
model_three.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model_three.fit(X_train, y_train, epochs=20, batch_size=20, validation
```

```
loss, mae = model_three.evaluate(X_test, y_test)
model_loss_data.append({"Model": "Model 3", "loss": loss, "mae": mae, "val_loss"
print(model_loss_data)

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.2603 - mae: 0.4215
[{'Model': 'Model 1', 'loss': 0.07223042100667953, 'mae': 0.2117377519607544, 'va
l_loss': 0.08148396760225296, 'val_mae': 0.23174971342086792}, {'Model': 'Model
2', 'loss': 0.07398424297571182, 'mae': 0.21946609020233154, 'val_loss': 0.079194
99278068542, 'val_mae': 0.2290734201669693}, {'Model': 'Model 3', 'loss': 0.26451
030373573303, 'mae': 0.4270596504211426, 'val_loss': 0.2840319275856018, 'val_ma
e': 0.45203959941864014}]
```



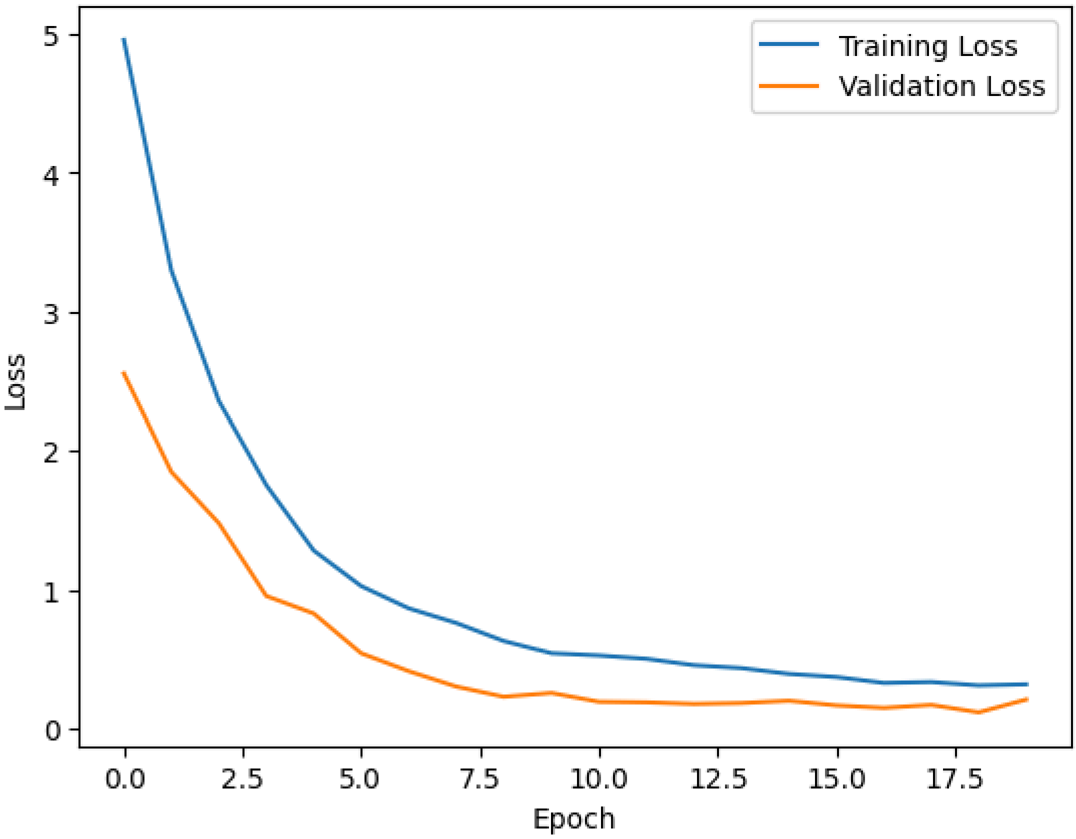
In [ ]:

```
input_shape = X_train.shape[1]

model_four = Sequential()
model_four.add(Dense(64, input_dim=input_shape, activation='relu'))
model_four.add(Dense(32, activation='relu'))
model_four.add(Dropout(0.3))
model_four.add(BatchNormalization())
model_four.add(Dense(16, activation='relu'))
model_four.add(Dropout(0.3))
model_four.add(BatchNormalization())
```

```
model_four.add(Dense(8, activation='relu'))
model_four.add(Dropout(0.3))
model_four.add(BatchNormalization())
model_four.add(Dense(1))
model_four.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model_four.fit(X_train, y_train, epochs=20, batch_size=20, validation_
loss, mae = model_four.evaluate(X_test, y_test)
model_loss_data.append({"Model": "Model 4", "loss": loss, "mae": mae, "val_loss"
print(model_loss_data)

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.1930 - mae: 0.3553
[{'Model': 'Model 1', 'loss': 0.07223042100667953, 'mae': 0.2117377519607544, 'va
l_loss': 0.08148396760225296, 'val_mae': 0.23174971342086792}, {'Model': 'Model
2', 'loss': 0.07398424297571182, 'mae': 0.21946609020233154, 'val_loss': 0.079194
99278068542, 'val_mae': 0.2290734201669693}, {'Model': 'Model 3', 'loss': 0.26451
030373573303, 'mae': 0.4270596504211426, 'val_loss': 0.2840319275856018, 'val_ma
e': 0.45203959941864014}, {'Model': 'Model 4', 'loss': 0.19432052969932556, 'ma
e': 0.35353323817253113, 'val_loss': 0.20898553729057312, 'val_mae': 0.3753891587
257385}]
```



In [ ]:
```
students_sample = pd.read_csv("Student_performance_data _.csv")
students_sample = students_sample.sample(5)
```

```
students_sample.drop(columns=['StudentID', 'Ethnicity', 'Volunteering', 'Gender'
students_sample_gpa = students_sample['GPA']
students_sample.drop(columns=['GPA'], inplace=True)
students_sample_two = students_sample.copy().drop(columns=['Age'])
students_predictions = []
```

In [ ]:
```
predictions_model_one = model_one.predict(students_sample)
predictions_model_two = model_two.predict(students_sample)
predictions_model_three = model_three.predict(students_sample)
predictions_model_four = model_four.predict(students_sample)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_
predict_function.<locals>.one_step_on_data_distributed at 0x7c6c720a3be0> trigger
ed tf.function retracing. Tracing is expensive and the excessive number of tracin
gs could be due to (1) creating @tf.function repeatedly in a loop, (2) passing te
nsors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function h
as reduce_retracing=True option that can avoid unnecessary retracing. For (3), pl
ease refer to https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_
predict_function.<locals>.one_step_on_data_distributed at 0x7c6c72159ea0> trigger
ed tf.function retracing. Tracing is expensive and the excessive number of tracin
gs could be due to (1) creating @tf.function repeatedly in a loop, (2) passing te
nsors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function h
as reduce_retracing=True option that can avoid unnecessary retracing. For (3), pl
ease refer to https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 124ms/step
```

In [ ]:
```
results = pd.DataFrame(
    {'Model 1': predictions_model_one.flatten(),
     'Model 2': predictions_model_two.flatten(),
     'Model 3': predictions_model_three.flatten(),
     'Model 4': predictions_model_four.flatten(),
     'Actual': students_sample_gpa,
     })
```

In [ ]:
```
results
```

Out[ ]:

	Model 1	Model 2	Model 3	Model 4	Actual
26	2.748951	2.856528	2.287121	2.589710	2.948718
815	0.095810	0.200842	0.693806	1.055423	0.019798
2032	1.015447	0.833012	1.333388	1.572101	1.172192
1548	0.929113	0.907471	1.127825	1.382188	1.050669
1413	2.619848	2.486144	2.036676	2.355843	2.644194

In [ ]:
```
models_df = pd.DataFrame(model_loss_data)
models_df
```

Out[ ]:

	Model	loss	mae	val_loss	val_mae
0	Model 1	0.072230	0.211738	0.081484	0.231750
1	Model 2	0.073984	0.219466	0.079195	0.229073
2	Model 3	0.264510	0.427060	0.284032	0.452040
3	Model 4	0.194321	0.353533	0.208986	0.375389

In [ ]:

```
!jupyter nbconvert --to html /content/drive/MyDrive/ColabNotebooks/TimeSeries_Fo
```