```
#Nguyễn Thiên Tính
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Reshape
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import Dropout
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
from tensorflow.keras.callbacks import History

```

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
df = pd.read_csv('/content/drive/MyDrive/5percent_MSSQL.csv')
df = df.applymap(lambda x: x if not isinstance(x, str) else None)
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df = df[~df.isin([np.nan]).any(axis=1)]
df = df.dropna(axis=1, how='all')
```

```
<ipython-input-5-70022ddfaade>:1: DtypeWarning: Columns (86) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('/content/drive/MyDrive/5percent_MSSQL.csv')
```

```
print(df)
```

```
288785       4631911      121994        16055             33609        17
288786       2610375        9448        32855              6197        17
288787       1998542       24043        15143             14340        17
288788       4572056       12645        32253             43549        17

        Flow Duration  Total Fwd Packets  Total Backward Packets  \
0                   1                  2                       0
1                  50                  2                       0
2                   2                  2                       0
3                  49                  2                       0
4                   1                  2                       0
...               ...                ...                     ...
288784              1                  2                       0
288785              1                  2                       0
288786              1                  2                       0
288787              1                  2                       0
288788              1                  2                       0

        Total Length of Fwd Packets  Total Length of Bwd Packets  ...  \
0                             888.0                          0.0  ...
1                             862.0                          0.0  ...
2                            1108.0                          0.0  ...
3                             952.0                          0.0  ...
4                            2200.0                          0.0  ...
...                             ...                          ...  ...
288784                        980.0                          0.0  ...
288785                        888.0                          0.0  ...
288786                        988.0                          0.0  ...
288787                       2944.0                          0.0  ...
288788                        856.0                          0.0  ...

        Active Mean  Active Std  Active Max  Active Min  Idle Mean  Idle Std  \
```

```
288786          0.0         0.0         0.0         0.0         0.0       0.0
288787          0.0         0.0         0.0         0.0         0.0       0.0
288788          0.0         0.0         0.0         0.0         0.0       0.0

        Idle Max  Idle Min  SimillarHTTP  Inbound
0            0.0       0.0           0.0        1
1            0.0       0.0           0.0        1
2            0.0       0.0           0.0        1
3            0.0       0.0           0.0        1
4            0.0       0.0           0.0        1
...          ...       ...           ...      ...
288784       0.0       0.0           0.0        1
288785       0.0       0.0           0.0        1
288786       0.0       0.0           0.0        1
288787       0.0       0.0           0.0        1
288788       0.0       0.0           0.0        1

[262869 rows x 84 columns]
```

```
1
```

```
1 df.head(100)
```

	Unnamed: 0.1	Unnamed: 0	Source Port	Destination Port	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	T Le of Pac
0	568269	42641	55989	50529	17	1	2	0	8
1	3914458	89976	61850	36734	17	50	2	0	8
2	2789854	85749	39757	45349	17	2	2	0	11
3	2834358	86263	11300	20633	17	49	2	0	9
4	5649515	25038	61850	3134	17	1	2	0	22
...	...	...	...	...	...	...	...	...	
102	3048392	41314	5955	28910	17	2	2	0	9
103	1037127	20190	55788	22408	17	2	2	0	8
104	2612643	93692	34863	57068	17	1	2	0	29
105	2746016	70311	5606	63388	17	1	2	0	11
106	394004	101238	65378	21023	17	1	2	0	19

```
1 #Chia dữ liệu thành hai phần
2 X = df.iloc[:, :-1]
3 y = df['Flow Duration']
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
1 #chuẩn hóa dữ liệu
2 scaler = StandardScaler()
3 X_train_scaled = scaler.fit_transform(X_train)
```

```
1   #huấn luyện mô hình phân loại
2   knn_model = KNeighborsClassifier(n_neighbors=5)
3   knn_model.fit(X_train_scaled, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
1   # Chuyển đổi X_test trước khi dự đoán
2   X_test_scaled = scaler.transform(X_test)
```

```
1   # Dự đoán với mô hình KNN
2   y_pred_knn = knn_model.predict(X_test_scaled)
```

```
1   # Tính toán các chỉ số đánh giá
2   print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
3   print("KNN Precision:", precision_score(y_test, y_pred_knn, average='weighted'))
4   print("KNN Recall:", recall_score(y_test, y_pred_knn, average='weighted'))
5   print("KNN F1 Score:", f1_score(y_test, y_pred_knn, average='weighted'))
```

```
print('KNN F1 Score:', f1_score(y_test, y_pred_knn, average='weighted'))
```

```
KNN Accuracy: 0.9008445239091566
KNN Precision: 0.8960723430194333
KNN Recall: 0.9008445239091566
KNN F1 Score: 0.8975692405855584
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, msg_start, len(result))
```

```
#mô hình Multilayer Perceptron (MLP) từ thư viện scikit-learn để tạo và huấn luyện một mạng nơ-ron
ann_model = MLPClassifier(hidden_layer_sizes=(100,), activation='tanh', max_iter=10)
ann_model.fit(X_train_scaled, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  warnings.warn(
```

```
MLPClassifier(activation='tanh', max_iter=10)
```

```
#đánh giá hiệu suất của mô hình Neural Network (DNN) trên tập dữ liệu kiểm thử đã được chuẩn hóa.
y_pred_ann = ann_model.predict(X_test_scaled)
print("DNN Accuracy:", accuracy_score(y_test, y_pred_ann))
print("DNN Precision:", precision_score(y_test, y_pred_ann, average='weighted'))
print("DNN Recall:", recall_score(y_test, y_pred_ann, average='weighted'))
print("DNN F1 Score:", f1_score(y_test, y_pred_ann, average='weighted'))
```

```
DNN Accuracy: 0.950850230151786
DNN Precision: 0.9349856454255758
DNN Recall: 0.950850230151786
DNN F1 Score: 0.9415727875654288
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, msg_start, len(result))
```

Double-click (or enter) to edit

```
#chuyển đổi dữ liệu từ DataFrame sang NumPy arrays
X_np = X.values
y_np = y.values
```

```
# chuyển đổi các giá trị trong mảng NumPy y_np từ chuỗi thành số nguyên
y_np = np.where(y_np == 'BENIGN', 0, 1)
```

```
#sử dụng Min-Max Scaling để chuẩn hóa dữ liệu trong DataFrame df
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df)
```

```
#bộ dữ liệu huấn luyện (X_tr, y_tr) và bộ dữ liệu kiểm thử (X_t, y_t).
X_tr, X_t, y_tr, y_t = train_test_split(data_scaled[:, :-1], data_scaled[:, -1], test_size=0.2, random_state=42)
```

```
#chuyển đổi mảng hai chiều thành mảng ba chiều để phù hợp với đầu vào mong đợi của một số mô hình học máy; 83 đặc
X_tr = X_tr.reshape((X_tr.shape[0], 1, 83))
X_t = X_t.reshape((X_t.shape[0], 1, 83))
```

```
#lưu trữ thông tin về quá trình huấn luyện mô hình
history = History()
```

```
#xây dựng, biên dịch và huấn luyện một mô hình LSTM
model_lstm = Sequential()
model_lstm.add(LSTM(units=10, input_shape=(X_tr.shape[1], X_tr.shape[2])))
model_lstm.add(Dense(units=1, activation='sigmoid'))
custom_optimizer = Adam(learning_rate=0.0001)
model_lstm.compile(loss='binary_crossentropy', optimizer=custom_optimizer, metrics=['accuracy'])
model_lstm.fit(X_tr, y_tr, epochs=100, batch_size=32, validation_data=(X_t, y_t), callbacks=[history])
```

```
Epoch 40/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 41/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 42/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 43/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 44/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 45/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 46/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 47/100
6572/6572 [==============================] - 29s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 48/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 49/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 50/100
6572/6572 [==============================] - 26s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 51/100
6572/6572 [==============================] - 25s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 52/100
6572/6572 [==============================] - 30s 5ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 53/100
6572/6572 [==============================] - 26s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 54/100
6572/6572 [==============================] - 25s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 55/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 56/100
6572/6572 [==============================] - 29s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 57/100
6572/6572 [==============================] - 29s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 58/100
6572/6572 [==============================] - 28s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 59/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 60/100
6572/6572 [==============================] - 30s 5ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 61/100
6572/6572 [==============================] - 26s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 62/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 63/100
6572/6572 [==============================] - 26s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 64/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 65/100
6572/6572 [==============================] - 29s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 66/100
6572/6572 [==============================] - 27s 4ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0016 - val_accuracy: 0.9998
```

```
#thực hiện dự đoán trên dữ liệu kiểm thử (X_t) bằng mô hình LSTM đã được huấn luyện (model_lstm)
y_pred_prob = model_lstm.predict(X_t)
y_pred = (y_pred_prob > 0.5).astype(int)
```

```
1643/1643 [==============================] - 3s 2ms/step
```

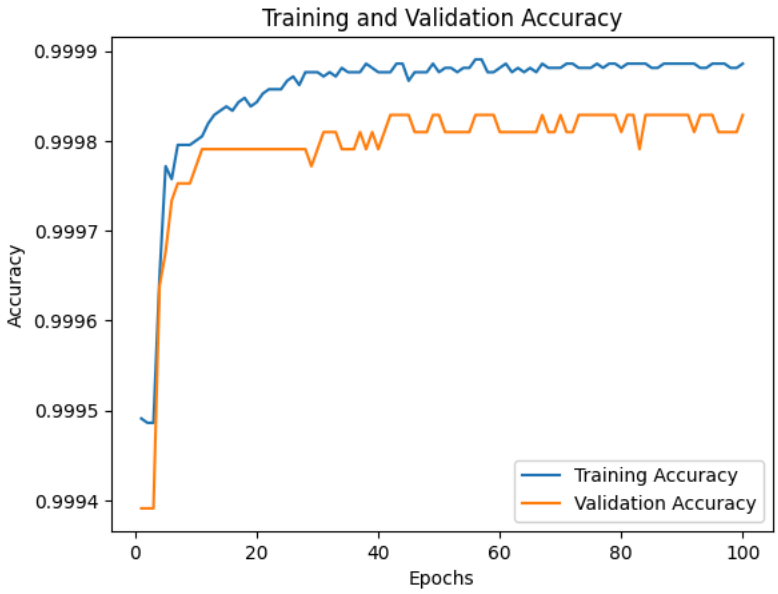
```
#đánh giá hiệu suất của mô hình
precision = precision_score(y_t, y_pred, average='weighted')
recall = recall_score(y_t, y_pred, average='weighted')
accuracy = accuracy_score(y_t, y_pred)
f1 = f1_score(y_t, y_pred, average='weighted')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'A: {accuracy}')
```

```
Precision: 0.9998217806463271
Recall: 0.9998288127211169
F1 Score: 0.9998215630952948
A: 0.9998288127211169
```

```
#truy xuất thông tin về độ chính xác (accuracy) của mô hình trên tập huấn luyện và tập kiểm thử qua các epoch tro
training_acc = history.history['accuracy']
validation_acc = history.history['val_accuracy']
```

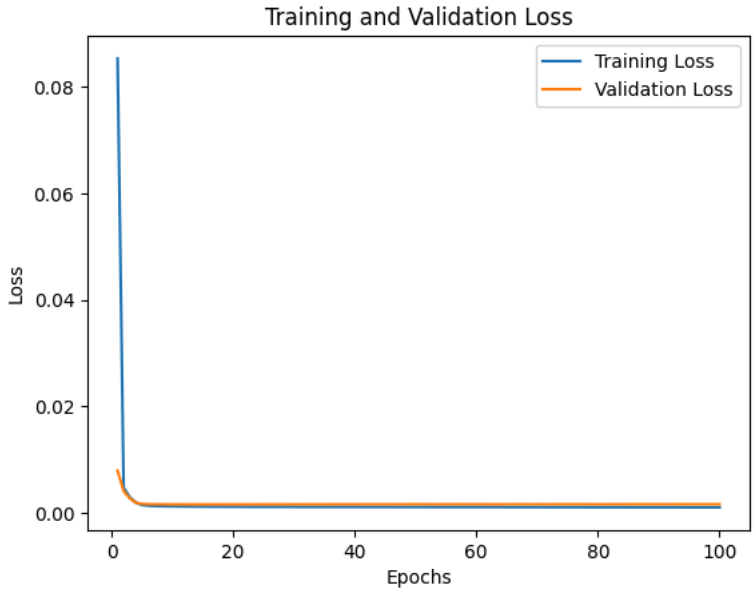
```
import matplotlib.pyplot as plt
# Lấy thông tin về accuracy từ lịch sử đào tạo
train_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
# Tạo mảng với số lượng epochs
epochs = range(1, len(train_accuracy) + 1)
```

```
# Vẽ đồ thị
plt.plot(epochs, train_accuracy, label='Training Accuracy')
plt.plot(epochs, validation_accuracy, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
# Lấy thông tin về loss từ lịch sử đào tạo
train_loss = history.history['loss']
validation_loss = history.history['val_loss']
# Tạo mảng với số lượng epochs
epochs = range(1, len(train_loss) + 1)
```

```
# Vẽ đồ thị
plt.plot(epochs, train_loss, label='Training Loss')
plt.plot(epochs, validation_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

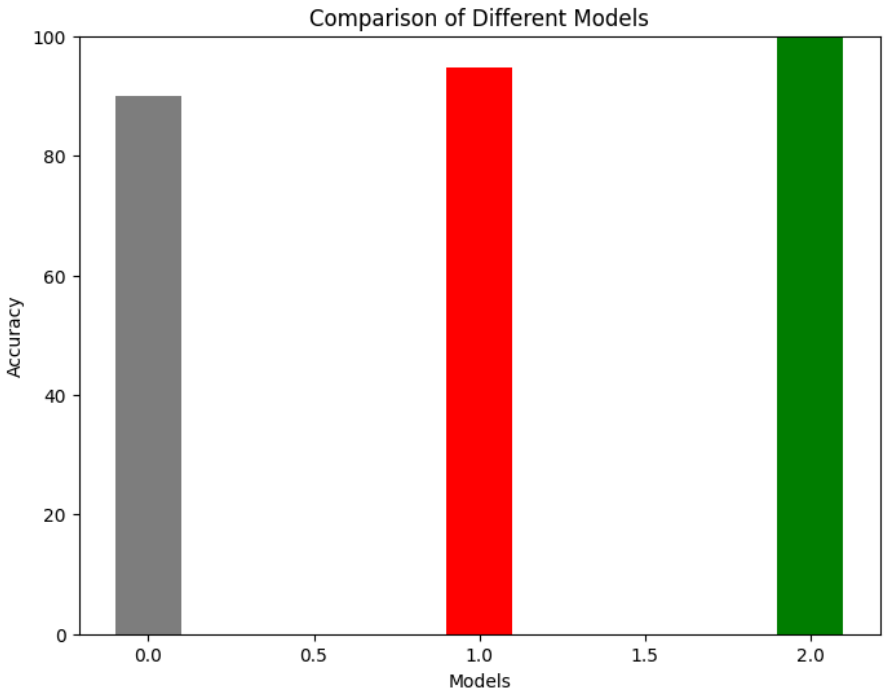



```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Giả sử bạn đã có các mô hình đã được huấn luyện
# Thay thế chúng bằng các mô hình thực tế của bạn
knn_accuracy = 90.08445239091566
dnn_accuracy = 94.75976718530071
lstm_accuracy = 99.97830387132076
```

```
models = ['KNN', 'DNN', 'LSTM']
accuracies = [knn_accuracy, dnn_accuracy, lstm_accuracy]
# Tiếp tục với code của bạn...
```

```
# Tạo biểu đồ
plt.figure(figsize=(8, 6))
bar_width = 0.2
positions = np.arange(len(models))
plt.bar(positions, accuracies, width=bar_width, color=['gray', 'red', 'green'], align='center')
# Thêm chú thích cho biểu đồ
plt.title('Comparison of Different Models')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.ylim([0, 100])
plt.show()
```




```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # Giả sử bạn đã có các mô hình đã được huấn luyện
4 # Thay thế chúng bằng các mô hình thực tế của bạn
5 knn_accuracy = 90.08445239091566
6 ann_accuracy = 94.75976718530071
7 lstm_accuracy = 99.97830387132076
8 models = ['KNN', 'DNN', 'LSTM']
9 accuracies = [knn_accuracy, ann_accuracy, lstm_accuracy]
```

```
1 # Màu sắc tương ứng với từng mô hình
2 colors = ['gray', 'red', 'green']
```

```
1 # Tạo biểu đồ
2 plt.figure(figsize=(8, 6))
3 bar_width = 0.2 # Tăng chiều rộng cột
4 positions = np.arange(len(models))
```

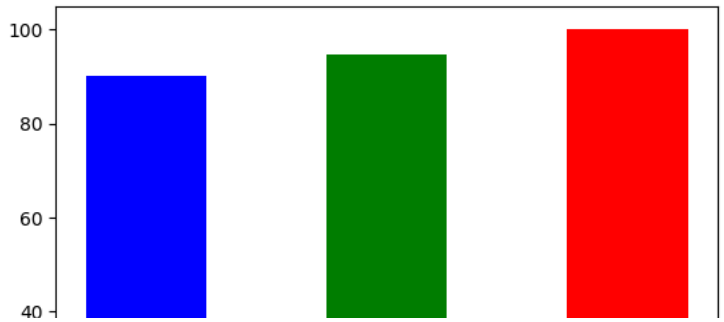
```
    <Figure size 800x600 with 0 Axes>
```

```
1 # Define colors for the bars
2 colors = ['blue', 'green', 'red']
```

```
1 # Define positions for the bars
2 positions = np.arange(len(models))
```

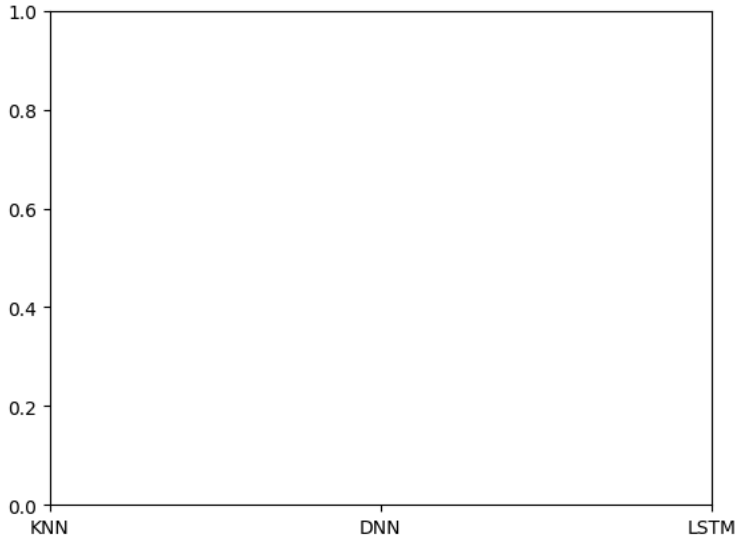
```
1   # Define the width of the bars
2   bar_width = 0.5
```

```
1   # Vẽ biểu đồ với màu sắc tương ứng và nhãn chú thích
2   for i, (model, accuracy, color) in enumerate(zip(models, accuracies, colors)):
3    plt.bar(positions[i], accuracy, width=bar_width, color=color, align='center')
```




```
# Đặt nhãn cho trục x
plt.xticks(positions, models)
```

```
([<matplotlib.axis.XTick at 0x7b823538d9c0>,
  <matplotlib.axis.XTick at 0x7b823538d990>,
  <matplotlib.axis.XTick at 0x7b823538d630>],
 [Text(0, 0, 'KNN'), Text(1, 0, 'DNN'), Text(2, 0, 'LSTM')])
```




```
# Đặt nhãn cho trục y
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```