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

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

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder, StandardScaler

import seaborn as sns

import keras
from keras import layers

import matplotlib.pyplot as plt

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

```
data = pd.read_excel("/content/drive/MyDrive/Naive/Drug_Attributes_DS.xlsx")
```

+ Code + Text

```
data = pd.read_excel("/content/drive/MyDrive/Naive/Disease_Drugs_DS.xlsx")
```

```
data.head()
```

	Drugs	Efficacy	Cost	Side Effects	Safety	Drug Disease Interactions
0	Metformin	0.50	0.90	0.60	0.40	B12 deficiency, Cardiovascular risk, Hypoglyce...
1	Simvastatin	0.60	0.60	0.50	0.70	Cognitive impairment, Diabetes, Liver disease,...
2	Levothyroxine	0.12	0.41	0.64	0.36	Adrenal insufficiency, Cardiovascular disease,...
3	Hydrocodone	0.95	0.83	0.86	0.15	Arrhythmias, Biliary spasm, Drug dependence
4	Amoxicillin	0.29	0.44	0.74	0.26	Colitis, Diabetes, Hemodialysis, Mononucleosis

Next steps: Generate code with data View recommended plots

```
y = data['Drugs']
```

```
X = data[["Efficacy","Cost","Side Effects", "Safety"]]
X
```

	Efficacy	Cost	Side Effects	Safety
0	0.50	0.90	0.60	0.40
1	0.60	0.60	0.50	0.70
2	0.12	0.41	0.64	0.36
3	0.95	0.83	0.86	0.15
4	0.29	0.44	0.74	0.26
...	...	...	...	...
4423	0.36	0.96	0.95	0.31
4424	0.37	0.01	0.27	0.47
4425	0.79	0.76	0.52	0.22
4426	0.14	0.83	0.57	0.07
4427	0.91	0.57	0.86	0.62

4428 rows × 4 columns

Next steps: Generate code with X View recommended plots

```
label_encoderY = LabelEncoder()
y_encoded = label_encoderY.fit_transform(y)


X_train, X_temp, y_train, y_temp = train_test_split(X, y_encoded, test_size=0.2, random_state=143)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=143)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)


def get_model():
    model = keras.Sequential()
    model.add(layers.Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(len(label_encoderY.classes_), activation='softmax'))

    # Compile the model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model


%time
model = get_model()
# Train the model
history = model.fit(X_train_scaled, y_train, epochs=30, batch_size=32, validation_data=(X_val_scaled, y_val))
```

```
Epoch 2/30
111/111 [==============================] - 1s 11ms/step - loss: 3.5247 - accuracy: 0.1784 - val_loss: 2.7285 - val_accuracy: 0.3837
Epoch 3/30
111/111 [==============================] - 1s 12ms/step - loss: 2.5661 - accuracy: 0.3097 - val_loss: 1.9606 - val_accuracy: 0.5688
Epoch 4/30
111/111 [==============================] - 1s 9ms/step - loss: 2.1319 - accuracy: 0.3834 - val_loss: 1.5579 - val_accuracy: 0.6185
Epoch 5/30
111/111 [==============================] - 1s 9ms/step - loss: 1.8416 - accuracy: 0.4503 - val_loss: 1.3134 - val_accuracy: 0.6682
Epoch 6/30
111/111 [==============================] - 1s 7ms/step - loss: 1.6582 - accuracy: 0.4831 - val_loss: 1.1078 - val_accuracy: 0.7223
Epoch 7/30
111/111 [==============================] - 1s 6ms/step - loss: 1.4989 - accuracy: 0.5220 - val_loss: 0.9666 - val_accuracy: 0.7856
Epoch 8/30
111/111 [==============================] - 0s 4ms/step - loss: 1.4034 - accuracy: 0.5438 - val_loss: 0.8333 - val_accuracy: 0.7946
Epoch 9/30
111/111 [==============================] - 1s 6ms/step - loss: 1.2878 - accuracy: 0.5858 - val_loss: 0.7524 - val_accuracy: 0.8578
Epoch 10/30
```

```
111/111 [==============================] - 1s 5ms/step - loss: 1.0591 - accuracy: 0.6494 - val_loss: 0.5258 - val_accuracy: 0.8894
Epoch 13/30
111/111 [==============================] - 1s 9ms/step - loss: 0.9788 - accuracy: 0.6858 - val_loss: 0.4765 - val_accuracy: 0.8691
Epoch 14/30
111/111 [==============================] - 1s 10ms/step - loss: 0.9436 - accuracy: 0.6894 - val_loss: 0.4616 - val_accuracy: 0.8826
Epoch 15/30
111/111 [==============================] - 1s 11ms/step - loss: 0.8643 - accuracy: 0.7185 - val_loss: 0.3983 - val_accuracy: 0.9142
Epoch 16/30
111/111 [==============================] - 1s 8ms/step - loss: 0.8303 - accuracy: 0.7219 - val_loss: 0.3588 - val_accuracy: 0.9052
Epoch 17/30
111/111 [==============================] - 1s 8ms/step - loss: 0.8102 - accuracy: 0.7292 - val_loss: 0.3433 - val_accuracy: 0.9391
Epoch 18/30
111/111 [==============================] - 1s 11ms/step - loss: 0.7627 - accuracy: 0.7431 - val_loss: 0.3144 - val_accuracy: 0.9300
Epoch 19/30
111/111 [==============================] - 1s 12ms/step - loss: 0.7116 - accuracy: 0.7578 - val_loss: 0.2746 - val_accuracy: 0.9345
Epoch 20/30
111/111 [==============================] - 1s 10ms/step - loss: 0.6986 - accuracy: 0.7668 - val_loss: 0.2557 - val_accuracy: 0.9481
Epoch 21/30
111/111 [==============================] - 1s 10ms/step - loss: 0.6473 - accuracy: 0.7741 - val_loss: 0.2410 - val_accuracy: 0.9571
Epoch 22/30
111/111 [==============================] - 1s 11ms/step - loss: 0.6154 - accuracy: 0.7874 - val_loss: 0.2121 - val_accuracy: 0.9594
Epoch 23/30
111/111 [==============================] - 1s 6ms/step - loss: 0.6118 - accuracy: 0.7919 - val_loss: 0.2384 - val_accuracy: 0.9300
Epoch 24/30
111/111 [==============================] - 1s 5ms/step - loss: 0.5864 - accuracy: 0.7967 - val_loss: 0.2180 - val_accuracy: 0.9436
Epoch 25/30
111/111 [==============================] - 1s 5ms/step - loss: 0.5710 - accuracy: 0.8032 - val_loss: 0.2109 - val_accuracy: 0.9391
Epoch 26/30
111/111 [==============================] - 1s 6ms/step - loss: 0.5441 - accuracy: 0.8145 - val_loss: 0.1996 - val_accuracy: 0.9458
Epoch 27/30
111/111 [==============================] - 1s 8ms/step - loss: 0.5424 - accuracy: 0.8139 - val_loss: 0.1583 - val_accuracy: 0.9684
Epoch 28/30
111/111 [==============================] - 1s 8ms/step - loss: 0.5248 - accuracy: 0.8111 - val_loss: 0.1646 - val_accuracy: 0.9594
Epoch 29/30
111/111 [==============================] - 1s 8ms/step - loss: 0.5200 - accuracy: 0.8210 - val_loss: 0.1633 - val_accuracy: 0.9616
Epoch 30/30
111/111 [==============================] - 1s 6ms/step - loss: 0.4766 - accuracy: 0.8377 - val_loss: 0.1311 - val_accuracy: 0.9774
```

```
test_loss, test_accuracy = model.evaluate(X_val_scaled, y_val)
print('val Loss:', test_loss * 100)
print('val Accuracy:', test_accuracy * 100)
```

```
14/14 [==============================] - 0s 5ms/step - loss: 0.1311 - accuracy: 0.9774
val Loss: 13.112227618694305
val Accuracy: 97.74266481399536
```

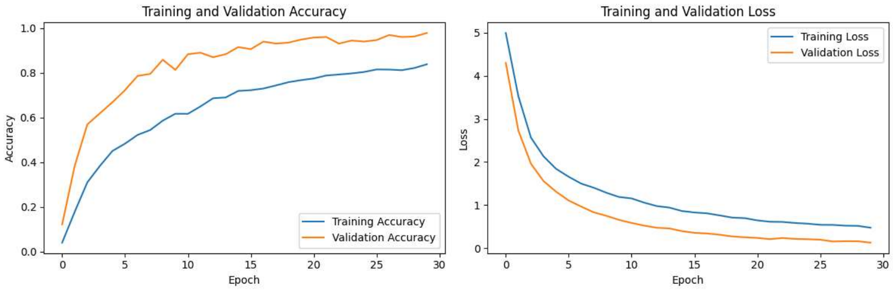
```
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_accuracy)
```

```
14/14 [==============================] - 0s 6ms/step - loss: 0.1680 - accuracy: 0.9684
Test Loss: 0.16797326505184174
Test Accuracy: 0.968397319316864
```

```
# Plot accuracy and loss curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



```
def test(data):
    test_case = np.array(data).reshape(1, -1)
    test_case_scaled = scaler.transform(test_case)
    #print(np.argmax(model.predict(test_case_scaled)))
    return label_encoderY.inverse_transform(np.array([np.argmax(model.predict(test_case_scaled))]))[0]
```

```
"Output: " + test([0.12, 0.41, 0.64, 0.36])
```

```
1/1 [==============================] - 0s 118ms/step
'Output: Levothyroxine'
```

Calculating F1-Score, Precision, Recal

```
y_true = list(label_encoderY.inverse_transform(y_test))
```

```
y_pred_lst = model.predict(X_test)
y_pred = []

for i in y_pred_lst:
    y_pred.append(label_encoderY.inverse_transform([np.argmax(i)])[0])
```

```
14/14 [                              ] - 0s 2ms/step
```