```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confu
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns


from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/MyDrive/UNSW_NB15_training-set.csv"

# ------------------ Load Dataset ------------------
data = pd.read_csv(path)
data.columns = data.columns.str.strip()

print("Dataset Shape:", data.shape)
print("Columns:", data.columns.tolist())

# ------------------ Identify Target Column ------------------
target_col = 'label'
print("Using Target Column:", target_col)

# ------------------ Handle Categorical Features ------------------
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
categorical_cols = [col for col in categorical_cols if col != target_col]

label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])

# ------------------ Split Features and Target ------------------
X = data.drop(target_col, axis=1)
y = data[target_col]

if y.dtype == 'object':
    y = LabelEncoder().fit_transform(y)

# ------------------ Handle infinite and NaN values ------------------
X = X.replace([np.inf, -np.inf], np.nan)
for col in X.columns:
    if X[col].isnull().any():
        X[col] = X[col].fillna(X[col].mean())

# ------------------ Train-Test Split ------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ------------------ Feature Scaling ------------------
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# ------------------ MODEL — RANDOM FOREST ------------------
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
pred_rf = rf.predict(X_test)

# ------------------ Evaluation ------------------
print("\nRandom Forest Results:")
print("Accuracy:", accuracy_score(y_test, pred_rf))
print("Precision (macro):", precision_score(y_test, pred_rf, average='macro'))
print("Recall (macro):", recall_score(y_test, pred_rf, average='macro'))
print("F1 Score (macro):", f1_score(y_test, pred_rf, average='macro'))
print("\nClassification Report:\n", classification_report(y_test, pred_rf))

# ------------------ Feature Importance Visualization ------------------
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
features = X.columns

plt.figure(figsize=(12,6))
plt.title("Feature Importances - Random Forest", fontsize=16)
plt.bar(range(len(importances)), importances[indices], align='center')
plt.xticks(range(len(importances)), [features[i] for i in indices], rotation=90)
plt.tight_layout()
plt.show()
```

```
# ------------------ Confusion Matrix ------------------
cm = confusion_matrix(y_test, pred_rf)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
Dataset Shape: (175341, 45)
Columns: ['id', 'dur', 'proto', 'service', 'state', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl',
Using Target Column: label

Random Forest Results:
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
from tensorflow import keras
from tensorflow.keras import layers

# ====================================================
# LOAD DATASET FROM GOOGLE DRIVE
path = "/content/drive/MyDrive/UNSW_NB15_training-set.csv"
data = pd.read_csv(path)
data.columns = data.columns.str.strip().str.replace('\ufeff','')

print("Dataset Shape:", data.shape)
print("Columns:", data.columns.tolist())

# ====================================================
# IDENTIFY TARGET COLUMN
#   UNSW-NB15 target column = 'label'
# ====================================================
target_col = 'label'
if target_col not in data.columns:
    raise ValueError(f"Target column '{target_col}' not found. Available: {data.columns}")

print("Using Target Column:", target_col)

# ====================================================
# HANDLE CATEGORICAL FEATURES (AUTO-DETECT)
# ====================================================
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()
for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])

# ====================================================
# SPLIT FEATURES AND TARGET
# ====================================================
X = data.drop(target_col, axis=1)
y = data[target_col]

# Encode target (0 = normal, 1 = attack)
y = LabelEncoder().fit_transform(y)

# ====================================================
# HANDLE NaN & INF VALUES
# ====================================================
X = X.replace([np.inf, -np.inf], np.nan)
X = X.fillna(X.mean())

# ====================================================
# TRAIN-TEST SPLIT
# ====================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ====================================================
# FEATURE SCALING
# ====================================================
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# ====================================================
# ANN MODEL
# ====================================================
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')    # Binary output
])
```

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

print(model.summary())

# ====================================================
# TRAIN THE MODEL
# ====================================================
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=64,
    validation_split=0.2,
    verbose=1
)

# ====================================================
# PREDICTIONS
# ====================================================
pred_ann = (model.predict(X_test) > 0.5).astype("int32").flatten()

# ====================================================
# EVALUATION
# ====================================================
print("\nANN Results:")
print("Accuracy:", accuracy_score(y_test, pred_ann))
print("Precision:", precision_score(y_test, pred_ann))
print("Recall:", recall_score(y_test, pred_ann))
print("F1 Score:", f1_score(y_test, pred_ann))
print("\nClassification Report:\n", classification_report(y_test, pred_ann))
```

```
Dataset Shape: (175341, 45)
Columns: ['id', 'dur', 'proto', 'service', 'state', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl',
Using Target Column: label
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_sha
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	5,760
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

```
 Total params: 16,129 (63.00 KB)
 Trainable params: 16,129 (63.00 KB)
 Non-trainable params: 0 (0.00 B)
None
Epoch 1/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9551 - loss: 0.1166 - val_accuracy: 0.9996 - val_loss:
Epoch 2/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9982 - loss: 0.0094 - val_accuracy: 0.9998 - val_loss: 8
Epoch 3/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9995 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 7
Epoch 4/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9996 - loss: 0.0014 - val_accuracy: 1.0000 - val_loss: 6
Epoch 5/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 8.2063e-04 - val_accuracy: 1.0000 - val_los
Epoch 6/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9997 - loss: 0.0015 - val_accuracy: 1.0000 - val_loss: 4
Epoch 7/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 2
Epoch 8/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9995 - loss: 0.0015 - val_accuracy: 1.0000 - val_loss: 5
Epoch 9/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9999 - loss: 2.9809e-04 - val_accuracy: 1.0000 - val_lo
Epoch 10/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9997 - loss: 0.0010 - val_accuracy: 1.0000 - val_loss: 2
Epoch 11/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 8.4525e-04 - val_accuracy: 1.0000 - val_los
Epoch 12/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9999 - loss: 5.7267e-04 - val_accuracy: 1.0000 - val_los
Epoch 13/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9997 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 8
Epoch 14/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 1.0000 - loss: 2.4837e-04 - val_accuracy: 1.0000 - val_los
Epoch 15/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9998 - loss: 5.1383e-04 - val_accuracy: 1.0000 - val_los
Epoch 16/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9999 - loss: 5.7565e-04 - val_accuracy: 1.0000 - val_los
Epoch 17/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 1.0000 - loss: 6.9271e-05 - val_accuracy: 1.0000 - val_los
Epoch 18/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 1.0000 - loss: 6.9167e-05 - val_accuracy: 1.0000 - val_los
Epoch 19/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9999 - loss: 4.7485e-04 - val_accuracy: 1.0000 - val_los
Epoch 20/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9999 - loss: 5.7312e-04 - val_accuracy: 1.0000 - val_los
1096/1096 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

ANN Results:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11200
           1       1.00      1.00      1.00     23869

    accuracy                           1.00     35069
   macro avg       1.00      1.00      1.00     35069
weighted avg       1.00      1.00      1.00     35069
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
```

```
from sklearn.ensemble import RandomForestClassifier
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# ====================================================
# LOAD DATASET FROM GOOGLE DRIVE
# Replace FILE_ID with your Google Drive file ID
# ====================================================
path = "/content/drive/MyDrive/UNSW_NB15_training-set.csv"

data = pd.read_csv(path)

# Clean column names
data.columns = data.columns.str.strip().str.replace('\ufeff','')

print("Dataset Shape:", data.shape)
print("Columns:", data.columns.tolist())

# ====================================================
# IDENTIFY TARGET COLUMN
# UNSW-NB15 uses "label" (0 = normal, 1 = attack)
# ====================================================
target_col = 'label'
if target_col not in data.columns:
    raise ValueError(f"Target column '{target_col}' not found. Columns = {data.columns}")

print("Using Target Column:", target_col)

# ====================================================
# HANDLE CATEGORICAL COLUMNS (AUTO-DETECT)
# ====================================================
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
label_encoder = LabelEncoder()

for col in categorical_cols:
    data[col] = label_encoder.fit_transform(data[col])

# ====================================================
# SPLIT FEATURES & TARGET
# ====================================================
X = data.drop(target_col, axis=1)
y = data[target_col]

y = LabelEncoder().fit_transform(y)   # 0/1

# ====================================================
# CLEAN NaN / INF
# ====================================================
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X.fillna(X.mean(), inplace=True)

# ====================================================
# TRAIN-TEST SPLIT
# ====================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ====================================================
# FEATURE SCALING
# RF does not require scaling, ANN requires scaling
# ====================================================
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ====================================================
# MODEL 1 — RANDOM FOREST
# ====================================================
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)   # RF uses unscaled data
pred_rf = rf.predict(X_test)

print("\nRandom Forest Results:")
print("Accuracy:", accuracy_score(y_test, pred_rf))
print("Precision:", precision_score(y_test, pred_rf))
print("Recall:", recall_score(y_test, pred_rf))
print("F1 Score:", f1_score(y_test, pred_rf))
print("\nClassification Report:\n", classification_report(y_test, pred_rf))

# ====================================================
# MODEL 2 — ANN
```

```
# ======================================================
ann_model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

ann_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = ann_model.fit(
    X_train_scaled, y_train,
    epochs=20,
    batch_size=64,
    validation_split=0.2,
    verbose=1
)

pred_ann = (ann_model.predict(X_test_scaled) > 0.5).astype("int32").flatten()

print("\nANN Results:")
print("Accuracy:", accuracy_score(y_test, pred_ann))
print("Precision:", precision_score(y_test, pred_ann))
print("Recall:", recall_score(y_test, pred_ann))
print("F1 Score:", f1_score(y_test, pred_ann))
print("\nClassification Report:\n", classification_report(y_test, pred_ann))

# ======================================================
# MODEL 3 — HYBRID (RF + ANN)
# Simple voting-based ensemble
# ======================================================
hybrid_pred = np.round((pred_rf + pred_ann) / 2).astype(int)

print("\nHybrid Model Results:")
print("Accuracy:", accuracy_score(y_test, hybrid_pred))
print("Precision:", precision_score(y_test, hybrid_pred))
print("Recall:", recall_score(y_test, hybrid_pred))
print("F1 Score:", f1_score(y_test, hybrid_pred))
print("\nClassification Report:\n", classification_report(y_test, hybrid_pred))

# ======================================================
# ACCURACY COMPARISON PLOT
# ======================================================
scores = [
    accuracy_score(y_test, pred_rf),
    accuracy_score(y_test, pred_ann),
    accuracy_score(y_test, hybrid_pred)
]

names = ["Random Forest", "ANN", "Hybrid"]

plt.figure(figsize=(7,4))
plt.bar(names, scores)
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()
```

```
Dataset Shape: (175341, 45)
Columns: ['id', 'dur', 'proto', 'service', 'state', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl', 'dttl
Using Target Column: label

Random Forest Results:
Accuracy: 0.9999714847871339
Precision: 0.9999581064097193
Recall: 1.0
F1 Score: 0.9999790527660822

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11200
           1       1.00      1.00      1.00     23869

    accuracy                           1.00     35069
   macro avg       1.00      1.00      1.00     35069
weighted avg       1.00      1.00      1.00     35069

Epoch 1/20
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_sh
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9559 - loss: 0.1114 - val_accuracy: 0.9996 - val_loss:
Epoch 2/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9986 - loss: 0.0061 - val_accuracy: 0.9999 - val_loss:
Epoch 3/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9996 - loss: 0.0018 - val_accuracy: 0.9999 - val_loss:
Epoch 4/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9996 - loss: 0.0015 - val_accuracy: 1.0000 - val_loss:
Epoch 5/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9998 - loss: 0.0010 - val_accuracy: 1.0000 - val_loss:
Epoch 6/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss:
Epoch 7/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9999 - loss: 5.1531e-04 - val_accuracy: 1.0000 - val_lo
Epoch 8/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 7.3317e-04 - val_accuracy: 1.0000 - val_lo
Epoch 9/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 6.5007e-04 - val_accuracy: 1.0000 - val_lo
Epoch 10/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9999 - loss: 9.6325e-04 - val_accuracy: 1.0000 - val_lo
Epoch 11/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 13s 5ms/step - accuracy: 0.9998 - loss: 6.4142e-04 - val_accuracy: 1.0000 - val_
Epoch 12/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9999 - loss: 5.9324e-04 - val_accuracy: 0.9999 - val_lo
Epoch 13/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9999 - loss: 8.4932e-04 - val_accuracy: 1.0000 - val_lo
Epoch 14/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9998 - loss: 7.8010e-04 - val_accuracy: 1.0000 - val_lo
Epoch 15/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 1.0000 - loss: 3.9257e-04 - val_accuracy: 1.0000 - val_lo
Epoch 16/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9998 - loss: 9.1966e-04 - val_accuracy: 0.9996 - val_lo
Epoch 17/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9998 - loss: 6.8817e-04 - val_accuracy: 1.0000 - val_lo
Epoch 18/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 1.0000 - loss: 2.2901e-04 - val_accuracy: 1.0000 - val_lo
Epoch 19/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9999 - loss: 2.9959e-04 - val_accuracy: 1.0000 - val_lo
Epoch 20/20
1754/1754 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 1.0000 - loss: 1.7184e-04 - val_accuracy: 1.0000 - val_lo
1096/1096 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

ANN Results:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Classification Report:
              precision    recall  f1-score   support
```