A Feed-forward Neural Network, or multi-layer perceptron, is a type of artificial neural network where information flows in one direction, from the input layer through one or more hidden layers to the output layer, without forming cycles. The input layer receives data, the hidden layers perform computations using weighted sums and activation functions, and the output layer produces the final result. Neurons, the basic units of an FNN, process inputs by applying weights, adding biases, and using activation functions like Sigmoid, Tanh, or ReLU to introduce non-linearity. The network is trained using forward propagation to compute outputs and measure error against actual targets, and backpropagation to adjust weights and biases by minimizing a loss function, typically optimized via algorithms like Stochastic Gradient Descent or Adam. FNNs can approximate any continuous function given sufficient neurons and layers, making them powerful tools for various applications such as classification, regression, and pattern recognition. They can be computationally intensive to train, risk overfitting, and require careful architecture design, including the selection of layers, neurons, and activation functions. Despite these challenges, FNNs remain widely used in machine learning for their ability to model complex relationships in data.

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X, y = make_classification(n_samples=10000, n_features=20, n_informative=15, n_redundant=5,
                           n_classes=2, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential([
    Dense(32, input_dim=X_train.shape[1], activation='relu'),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')

y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))
```