Ejercicio01, esta realizado sobre la creación de una red neuronal que predice qué plato del menú de un restaurante debería recibir una promoción para aumentar las ventas. Este ejercicio utiliza TensorFlow y Keras.

```
# Importar bibliotecas necesarias
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Generar datos simulados para el restaurante
np.random.seed(42)

# Datos del restaurante
platos = ['Ceviche', 'Lomo Saltado', 'Aji de Gallina', 'Pollo a la Brasa']
dias_semana = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo']

data = {
  'DiaSemana': np.random.choice(dias_semana, 200),
  'Clientes': np.random.randint(10, 100, 200),
  'PromocionActual': np.random.choice([0, 1], 200),  # 0: Sin promoción, 1: Con promoción
  'Plato': np.random.choice(platos, 200)
}

df = pd.DataFrame(data)

# Generar variable objetivo (Ventas Simuladas)
df['Ventas'] = df['Clientes'] * (np.random.uniform(0.8, 1.2, 200)) + df['PromocionActual'] * 10

# Codificar variables categóricas
label_encoder_dia = LabelEncoder()
df['DiaSemana'] = label_encoder_dia.fit_transform(df['DiaSemana'])

label_encoder_plato = LabelEncoder()
df['Plato'] = label_encoder_plato.fit_transform(df['Plato'])

# Dividir en características y variable objetivo
X = df[['DiaSemana', 'Clientes', 'PromocionActual']]
y = df['Plato']

# Normalizar los datos
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear el modelo de red neuronal
model = Sequential([
  Dense(16, input_dim=X_train.shape[1], activation='relu'),
  Dense(8, activation='relu'),
```

```
  Dense(4, activation='softmax')  # 4 salidas porque hay 4 platos
])

# Compilar el modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=10, validation_data=(X_test, y_test))

# Evaluar el modelo
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Pérdida: {loss:.4f}, Precisión: {accuracy:.4f}")

# Hacer predicciones
predicciones = model.predict(X_test)
predicciones_clases = np.argmax(predicciones, axis=1)

# Mostrar resultados
resultados = pd.DataFrame({
  'Real': label_encoder_plato.inverse_transform(y_test),
  'Predicción': label_encoder_plato.inverse_transform(predicciones_clases)
})
print(resultados.head())

# Gráficas de desempeño
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# Precisión del entrenamiento y validación
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Precisión del modelo')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()

# Pérdida del entrenamiento y validación
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida del modelo')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()

plt.show()
```