Universidad del Valle de Guatemala

Facultad de Ingeniería

Departamento de Ciencias de la Computación

CC3066 – Data Science

Cristian Eduardo Aguirre Duarte - 20231

Parte 1

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
```

Parte 1.1: Cargue los datos en un DataFrame de Pandas

In [ ]:
```
data = pd.read_csv('UK_foods.csv')
# Transponer el DataFrame para tener países como filas y alimentos como columnas
data = data.set_index('Unnamed: 0').T
```

Parte 1.2: Realice una exploración básica de los datos (estadísticas descriptivas)

In [ ]:
```
data.head()
```

Out[ ]:

Unnamed: 0	Cheese	Carcass_meat	Other_meat	Fish	Fats_and_oils	Sugars	Fresh_potatoes
England	105	245	685	147	193	156	720
Wales	103	227	803	160	235	175	874
Scotland	103	242	750	122	184	147	566
N.Ireland	66	267	586	93	209	139	1033

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, England to N.Ireland
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Cheese               4 non-null      int64
 1   Carcass_meat         4 non-null      int64
 2   Other_meat           4 non-null      int64
 3   Fish                 4 non-null      int64
 4   Fats_and_oils        4 non-null      int64
 5   Sugars               4 non-null      int64
 6   Fresh_potatoes       4 non-null      int64
 7   Fresh_Veg            4 non-null      int64
 8   Other_Veg            4 non-null      int64
 9   Processed_potatoes   4 non-null      int64
 10  Processed_Veg        4 non-null      int64
 11  Fresh_fruit          4 non-null      int64
 12  Cereals              4 non-null      int64
 13  Beverages            4 non-null      int64
 14  Soft_drinks          4 non-null      int64
 15  Alcoholic_drinks     4 non-null      int64
 16  Confectionery        4 non-null      int64
dtypes: int64(17)
memory usage: 576.0+ bytes
```

In [ ]:
```
data.describe()
```

Out[ ]:

	Unnamed: 0	Cheese	Carcass_meat	Other_meat	Fish	Fats_and_oils	Sugars	Fre
count	4.000000	4.00	4.000000	4.000000	4.000000	4.00000		
mean	94.250000	245.25	706.000000	130.500000	205.250000	154.25000		
std	18.856917	16.50	93.427334	29.557853	22.366269	15.47848		
min	66.000000	227.00	586.000000	93.000000	184.000000	139.00000		
25%	93.750000	238.25	660.250000	114.750000	190.750000	145.00000		
50%	103.000000	243.50	717.500000	134.500000	201.000000	151.50000		
75%	103.500000	250.50	763.250000	150.250000	215.500000	160.75000		
max	105.000000	267.00	803.000000	160.000000	235.000000	175.00000		

Parte 1.3: Normalice los datos

In [ ]:
```
normalizador = MinMaxScaler()
normalized_data = normalizador.fit_transform(data)
```

Parte 2

In [ ]:
```
# Construir el autocodificador
codificador = Sequential()
codificador.add(Dense(units=2, activation='relu', input_shape=[17]))
decodificador = Sequential()
decodificador.add(Dense(units=17, activation='relu', input_shape=[2]))
autocodificador = Sequential([codificador, decodificador])

# Compilar el autocodificador
autocodificador.compile(loss='mse', optimizer=SGD(lr=1.5))

# Entrenar el autocodificador
autocodificador.fit(normalized_data, normalized_data, epochs=100)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or us
e the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
```

```
Epoch 1/100
1/1 [==============================] - 0s 215ms/step - loss: 0.3210
Epoch 2/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3204
Epoch 3/100
1/1 [==============================] - 0s 4ms/step - loss: 0.3198
Epoch 4/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3192
Epoch 5/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3186
Epoch 6/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3180
Epoch 7/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3174
Epoch 8/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3168
Epoch 9/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3162
Epoch 10/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3156
Epoch 11/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3150
Epoch 12/100
1/1 [==============================] - 0s 5ms/step - loss: 0.3144
Epoch 13/100
1/1 [==============================] - 0s 5ms/step - loss: 0.3139
Epoch 14/100
1/1 [==============================] - 0s 4ms/step - loss: 0.3133
Epoch 15/100
1/1 [==============================] - 0s 5ms/step - loss: 0.3127
Epoch 16/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3121
Epoch 17/100
1/1 [==============================] - 0s 2ms/step - loss: 0.3116
Epoch 18/100
1/1 [==============================] - 0s 5ms/step - loss: 0.3110
Epoch 19/100
1/1 [==============================] - 0s 4ms/step - loss: 0.3104
Epoch 20/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3099
Epoch 21/100
1/1 [==============================] - 0s 2ms/step - loss: 0.3093
Epoch 22/100
1/1 [==============================] - 0s 5ms/step - loss: 0.3087
Epoch 23/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3082
Epoch 24/100
1/1 [==============================] - 0s 2ms/step - loss: 0.3076
Epoch 25/100
1/1 [==============================] - 0s 4ms/step - loss: 0.3071
Epoch 26/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3065
Epoch 27/100
1/1 [==============================] - 0s 4ms/step - loss: 0.3060
Epoch 28/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3054
```

```
Epoch 29/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3049
Epoch 30/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3044
Epoch 31/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3038
Epoch 32/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3033
Epoch 33/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3028
Epoch 34/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3022
Epoch 35/100
1/1 [==============================] - 0s 4ms/step - loss: 0.3017
Epoch 36/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3012
Epoch 37/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3007
Epoch 38/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3001
Epoch 39/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2996
Epoch 40/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2991
Epoch 41/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2986
Epoch 42/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2981
Epoch 43/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2976
Epoch 44/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2971
Epoch 45/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2966
Epoch 46/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2961
Epoch 47/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2956
Epoch 48/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2951
Epoch 49/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2946
Epoch 50/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2941
Epoch 51/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2935
Epoch 52/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2928
Epoch 53/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2922
Epoch 54/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2915
Epoch 55/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2908
Epoch 56/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2902
```

```
Epoch 57/100
1/1 [==============================] - 0s 5ms/step - loss: 0.2895
Epoch 58/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2889
Epoch 59/100
1/1 [==============================] - 0s 5ms/step - loss: 0.2882
Epoch 60/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2876
Epoch 61/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2869
Epoch 62/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2863
Epoch 63/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2856
Epoch 64/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2850
Epoch 65/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2843
Epoch 66/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2837
Epoch 67/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2831
Epoch 68/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2825
Epoch 69/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2818
Epoch 70/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2812
Epoch 71/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2806
Epoch 72/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2800
Epoch 73/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2794
Epoch 74/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2788
Epoch 75/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2781
Epoch 76/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2775
Epoch 77/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2769
Epoch 78/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2763
Epoch 79/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2758
Epoch 80/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2752
Epoch 81/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2746
Epoch 82/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2740
Epoch 83/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2734
Epoch 84/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2728
```

```
Epoch 85/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2722
Epoch 86/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2717
Epoch 87/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2711
Epoch 88/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2705
Epoch 89/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2699
Epoch 90/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2694
Epoch 91/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2688
Epoch 92/100
1/1 [==============================] - 0s 5ms/step - loss: 0.2683
Epoch 93/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2677
Epoch 94/100
1/1 [==============================] - 0s 5ms/step - loss: 0.2671
Epoch 95/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2666
Epoch 96/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2660
Epoch 97/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2655
Epoch 98/100
1/1 [==============================] - 0s 4ms/step - loss: 0.2650
Epoch 99/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2644
Epoch 100/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2639
```

Out[ ]: <keras.callbacks.History at 0x146df885e70>

In [ ]:
```
# Evaluar el rendimiento del autocodificador utilizando la pérdida de reconstrucció
loss = autocodificador.evaluate(normalized_data, normalized_data)
loss = loss * 100
print(f"Pérdida de reconstrucción: {loss:.2f} %")
```

```
1/1 [==============================] - 0s 60ms/step - loss: 0.2633
Pérdida de reconstrucción: 26.33 %
1/1 [==============================] - 0s 60ms/step - loss: 0.2633
Pérdida de reconstrucción: 26.33 %
```

In [ ]:
```
# Obtener las dos dimensiones reducidas
codificado_2dim = codificador.predict(normalized_data)
```

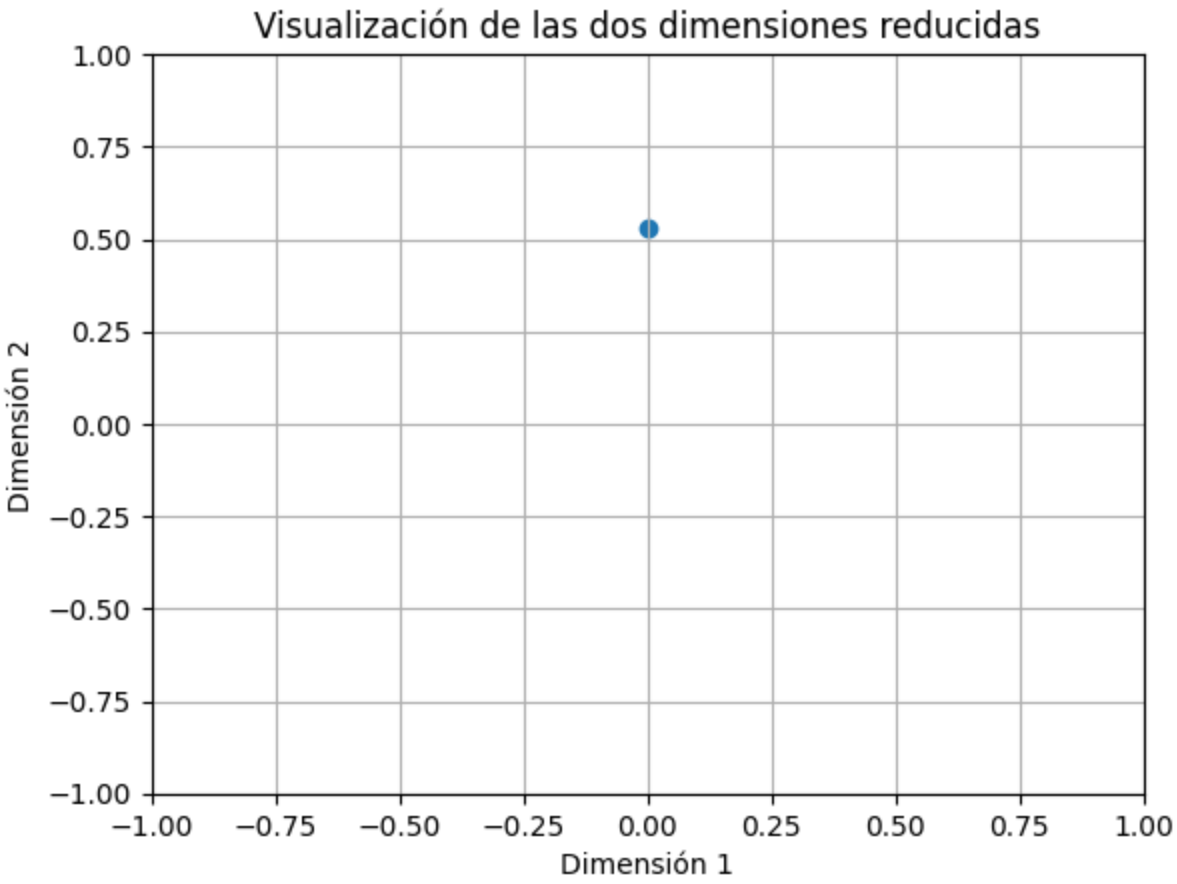
```
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```

Parte 3

In [ ]:
```
# Visualizar las dos dimensiones reducidas
plt.scatter(codificado_2dim[:, 0], codificado_2dim[:, 1])
plt.xlabel('Dimensión 1')
```

```
plt.ylabel('Dimensión 2')
plt.xlim(-1, 1)  # Ajusta los límites del eje X
plt.ylim(-1, 1)  # Ajusta los límites del eje Y
plt.title('Visualización de las dos dimensiones reducidas')
plt.grid(True)
plt.show()

# También puedes usar Plotly Express para una visualización interactiva
fig = px.scatter(x=codificado_2dim[:, 0], y=codificado_2dim[:, 1], width=700, heigh
fig.show()
```



Parte 4

4.1 ¿Qué podemos aprender sobre los patrones de consumo de alimentos en los cuatro países del Reino Unido?

El primer punto (0, 0.528782) indica que un país (probablemente Escocia) tiene una coordenada positiva en la Dimensión 1 y una coordenada positiva en la Dimensión 2. Esto podría sugerir que este país tiene un patrón de consumo de alimentos diferente en comparación con los otros.

El segundo punto (1.591624, 0.1067735) indica que otro país (posiblemente Gales) tiene una coordenada positiva en la Dimensión 1 y una coordenada cercana a cero en la Dimensión 2. Esto podría sugerir que este país también tiene un patrón de consumo de alimentos distinto.

El tercer punto (1.682579, 0) muestra un país (quizás Inglaterra) con una coordenada positiva en la Dimensión 1 y una coordenada igual a cero en la Dimensión 2. Esto podría indicar que este país tiene un patrón de consumo particular en la Dimensión 1, pero similar a otros en la Dimensión 2.

El cuarto punto (1.95713, 0) representa otro país (posiblemente Irlanda del Norte) con coordenadas positivas en la Dimensión 1 y Dimensión 2 igual a cero. Esto podría indicar un patrón de consumo único en la Dimensión 1 pero similar a otros en la Dimensión 2.

4.2 ¿El autocodificador hizo un buen trabajo al reducir la dimensionalidad? ¿Qué métricas de rendimiento apoyan su afirmación?

El autocodificador ha reducido la dimensionalidad de manera efectiva a 2 dimensiones, por otra parte este modelo fue evaluado haciendo uso de la pérdida de reconstrucción, la cual mostró un resultado del 26.33 % que indica que hay cierto nivel de error en la reconstrucción, pero lo cual no indica necesariamente un mal rendimiento. Además con las visualizaciones podemos ver las representaciones de los 4 paises mostrados en el conjunto de datos de los cuales podemos determinar ciertas tendencias en los patrones de consumo de cada uno de ellos, por lo cual también es un indicador de la efectividad del modelo.