```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
!wget -O diamonds_prices2022.csv "https://drive.google.com/uc?id=1f7l-r51EO6DhMbMotrijyke6xIty0AdX"
```

```
--2023-06-22 16:18:55--  https://drive.google.com/uc?id=1f7l-r51EO6DhMbMotrijyke6xIty0AdX
Resolving drive.google.com (drive.google.com)... 74.125.20.100, 74.125.20.139, 74.125.20.113, ...
Connecting to drive.google.com (drive.google.com)|74.125.20.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0c-bo-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/mm6ocfdcj3uo37g14i3eq00keo2185k8/1
Warning: wildcards not supported in HTTP.
--2023-06-22 16:18:56--  https://doc-0c-bo-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/mm6ocfdcj3uo37g14i3
Resolving doc-0c-bo-docs.googleusercontent.com (doc-0c-bo-docs.googleusercontent.com)... 142.250.107.132, 2607:f8b0:400e:c0d::84
Connecting to doc-0c-bo-docs.googleusercontent.com (doc-0c-bo-docs.googleusercontent.com)|142.250.107.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2815122 (2.7M) [text/csv]
Saving to: 'diamonds_prices2022.csv'

diamonds_prices2022 100%[===================>]   2.68M  --.-KB/s    in 0.02s

2023-06-22 16:18:56 (175 MB/s) - 'diamonds_prices2022.csv' saved [2815122/2815122]
```

```python
dataset_original = pd.read_csv('diamonds_prices2022.csv')
```

```python
dataset_original
```

| | Unnamed: 0 | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55.0 | 326 | 3.95 | 3.98 | 2.43 |
| **1** | 2 | 0.21 | Premium | E | SI1 | 59.8 | 61.0 | 326 | 3.89 | 3.84 | 2.31 |
| **2** | 3 | 0.23 | Good | E | VS1 | 56.9 | 65.0 | 327 | 4.05 | 4.07 | 2.31 |
| **3** | 4 | 0.29 | Premium | I | VS2 | 62.4 | 58.0 | 334 | 4.20 | 4.23 | 2.63 |
| **4** | 5 | 0.31 | Good | J | SI2 | 63.3 | 58.0 | 335 | 4.34 | 4.35 | 2.75 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **53938** | 53939 | 0.86 | Premium | H | SI2 | 61.0 | 58.0 | 2757 | 6.15 | 6.12 | 3.74 |
| **53939** | 53940 | 0.75 | Ideal | D | SI2 | 62.2 | 55.0 | 2757 | 5.83 | 5.87 | 3.64 |
| **53940** | 53941 | 0.71 | Premium | E | SI1 | 60.5 | 55.0 | 2756 | 5.79 | 5.74 | 3.49 |
| **53941** | 53942 | 0.71 | Premium | F | SI1 | 59.8 | 62.0 | 2756 | 5.74 | 5.73 | 3.43 |
| **53942** | 53943 | 0.70 | Very Good | E | VS2 | 60.5 | 59.0 | 2757 | 5.71 | 5.76 | 3.47 |

53943 rows × 11 columns

```python
dataset_original.shape
```

```
(53943, 11)
```

```python
dataset_original.describe()
```

| Unnamed: 0 | carat | depth | table | price | x | y |
|---|---|---|---|---|---|---|

```
dataset_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  53943 non-null  int64  
 1   carat       53943 non-null  float64
 2   cut         53943 non-null  object 
 3   color       53943 non-null  object 
 4   clarity     53943 non-null  object 
 5   depth       53943 non-null  float64
 6   table       53943 non-null  float64
 7   price       53943 non-null  int64  
 8   x           53943 non-null  float64
 9   y           53943 non-null  float64
 10  z           53943 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

```
dataset_copia = dataset_original.head(25000).copy()
columns_to_drop = ['Unnamed: 0']
dataset_copia.drop(columns=columns_to_drop, inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for columna in ["cut", "color", "clarity"]:
    dataset_copia[columna + "_numerica"] = le.fit_transform(dataset_copia[columna])
```

```
print(dataset_copia)
print("Dimensiones del dataset_copia:", dataset_copia.shape)
print("Dimensiones del dataset_original:", dataset_original.shape)
print(dataset_copia.info())
```

```
       carat        cut color clarity  depth  table  price     x     y     z  \
0       0.23      Ideal     E     SI2   61.5   55.0    326  3.95  3.98  2.43   
1       0.21    Premium     E     SI1   59.8   61.0    326  3.89  3.84  2.31   
2       0.23       Good     E     VS1   56.9   65.0    327  4.05  4.07  2.31   
3       0.29    Premium     I     VS2   62.4   58.0    334  4.20  4.23  2.63   
4       0.31       Good     J     SI2   63.3   58.0    335  4.34  4.35  2.75   
...      ...        ...   ...     ...    ...    ...    ...   ...   ...   ...   
24995   1.54      Ideal     G     VS2   60.2   59.0  13508  7.47  7.52  4.51   
24996   2.20      Ideal     I     SI2   62.2   56.0  13512  8.33  8.29  5.17   
24997   1.50    Premium     E     VS1   60.1   60.0  13513  7.42  7.48  4.48   
24998   1.51       Good     G     VS1   62.4   59.0  13515  7.25  7.28  4.53   
24999   1.50  Very Good     G     VS2   61.1   60.0  13528  7.40  7.30  4.49   

       cut_numerica  color_numerica  clarity_numerica  
0                 2               1                 3  
1                 3               1                 2  
2                 1               1                 4  
3                 3               5                 5  
4                 1               6                 3  
...             ...             ...               ...  
24995             2               3                 5  
24996             2               5                 3  
24997             3               1                 4  
24998             1               3                 4  
24999             4               3                 5  

[25000 rows x 13 columns]
Dimensiones del dataset_copia: (25000, 13)
Dimensiones del dataset_original: (53943, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   carat             25000 non-null  float64
 1   cut               25000 non-null  object 
 2   color             25000 non-null  object 
 3   clarity           25000 non-null  object 
 4   depth             25000 non-null  float64
 5   table             25000 non-null  float64
 6   price             25000 non-null  int64  
 7   x                 25000 non-null  float64
 8   y                 25000 non-null  float64
```

```
 9   z                 25000 non-null  float64
 10  cut_numerica      25000 non-null  int64
 11  color_numerica    25000 non-null  int64
 12  clarity_numerica  25000 non-null  int64
dtypes: float64(6), int64(4), object(3)
memory usage: 2.5+ MB
None
```

```
1 vu1 = dataset_copia[["cut", "cut_numerica"]].drop_duplicates()
2 vu2 = dataset_copia[["color", "color_numerica"]].drop_duplicates()
3 vu3 = dataset_copia[["clarity", "clarity_numerica"]].drop_duplicates()
4 vu1
```

| | cut | cut_numerica |
|---|---|---|
| 0 | Ideal | 2 |
| 1 | Premium | 3 |
| 2 | Good | 1 |
| 5 | Very Good | 4 |
| 8 | Fair | 0 |

```
1 vu2
```

| | color | color_numerica |
|---|---|---|
| 0 | E | 1 |
| 3 | I | 5 |
| 4 | J | 6 |
| 7 | H | 4 |
| 12 | F | 2 |
| 25 | G | 3 |
| 28 | D | 0 |

```
1 vu3
```

| | clarity | clarity_numerica |
|---|---|---|
| 0 | SI2 | 3 |
| 1 | SI1 | 2 |
| 2 | VS1 | 4 |
| 3 | VS2 | 5 |
| 5 | VVS2 | 7 |
| 6 | VVS1 | 6 |
| 15 | I1 | 0 |
| 229 | IF | 1 |

```
1 columns_to_drop = ['cut','color','clarity']
2 dataset_copia.drop(columns=columns_to_drop, inplace=True)
```

```
1 dataset_copia.sample(10)
```

| | carat | depth | table | price | x | y | z | cut_numerica | color_numerica | clarity_numerica |
|---|---|---|---|---|---|---|---|---|---|---|
| **18899** | 1.03 | 59.0 | 55.0 | 7752 | 6.67 | 6.62 | 3.92 | 2 | 2 | 4 |
| **7104** | 1.08 | 63.3 | 59.0 | 4167 | 6.48 | 6.44 | 4.09 | 4 | 2 | 3 |
| **14849** | 1.25 | 62.1 | 56.0 | 5980 | 6.81 | 6.84 | 4.24 | 2 | 6 | 7 |

```
# Calcular el rango intercuartílico (IQR)
Q1 = dataset_copia.quantile(0.25)
Q3 = dataset_copia.quantile(0.75)
IQR = Q3 - Q1

# Definir los límites superior e inferior
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR

# Eliminar los valores atípicos
df_filtrado = dataset_copia[((dataset_copia>= limite_inferior) & (dataset_copia<= limite_superior)).all(axis=1)]

# Imprimir el DataFrame filtrado
print(df_filtrado)
```

```
       carat  depth  table  price     x     y     z  cut_numerica  \
90      0.70   62.5   57.0   2757  5.70  5.72  3.57             2
92      0.70   61.6   56.0   2757  5.70  5.67  3.50             2
93      0.71   62.4   57.0   2759  5.68  5.73  3.56             4
94      0.78   63.8   56.0   2759  5.81  5.85  3.72             4
96      0.70   59.4   62.0   2759  5.71  5.76  3.40             1
...      ...    ...    ...    ...   ...   ...   ...           ...
24595   1.53   62.8   57.0  12907  7.48  7.43  4.63             2
24597   1.52   62.0   56.0  12907  7.36  7.34  4.56             2
24599   1.63   61.8   57.0  12910  7.50  7.54  4.64             2
24600   1.19   62.0   57.0  12912  6.86  6.76  4.22             2
24602   1.52   59.3   59.0  12916  7.48  7.55  4.46             4

       color_numerica  clarity_numerica
90                  1                 2
92                  3                 5
93                  1                 5
94                  3                 3
96                  2                 4
...               ...               ...
24595               3                 5
24597               1                 2
24599               4                 2
24600               0                 7
24602               3                 4

[19820 rows x 10 columns]
```

```
# Dividir el conjunto de datos en conjuntos de entrenamiento y prueba
train_dataset = df_filtrado.sample(frac=0.8, random_state=0)
test_dataset = df_filtrado.drop(train_dataset.index)
```

```
# Separe la variable objetivo (precio) de las características
train_features = train_dataset.drop("price", axis=1)
train_labels = train_dataset["price"]
test_features = test_dataset.drop("price", axis=1)
test_labels = test_dataset["price"]
```

```
train_features
```

| | carat | depth | table | x | y | z | cut_numerica | color_numerica | clarity_numerica |
|---|---|---|---|---|---|---|---|---|---|
| **19328** | 1.53 | 62.4 | 58.0 | 7.32 | 7.38 | 4.59 | 2 | 4 | 3 |
| **22095** | 1.17 | 61.7 | 59.0 | 6.77 | 6.72 | 4.16 | 3 | 0 | 4 |
| **22837** | 1.05 | 61.6 | 55.0 | 6.57 | 6.53 | 4.04 | 2 | 2 | 6 |
| **21748** | 1.10 | 61.2 | 56.0 | 6.68 | 6.65 | 4.08 | 2 | 2 | 7 |

```
#from tensorflow.keras.layers import Normalization
#normalizar = Normalization(axis=-1)
#normalizar.adapt(np.array(train_features))
# Normalizamos the features
train_mean = train_features.mean()
train_std = train_features.std()
train_features = (train_features - train_mean) / train_std
test_features = (test_features - train_mean) / train_std
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **5428** | 0.94 | 60.4 | 59.0 | 6.34 | 6.37 | 3.84 | 3 | 1 | 3 |

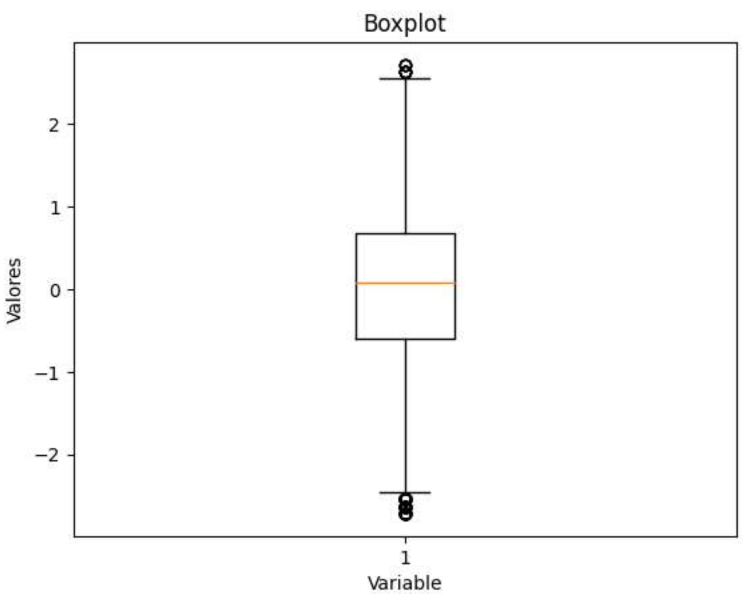
```
train_features['depth'].min()
```

```
-2.7136591012932323
```

```
# Crear el boxplot
plt.boxplot(train_features['depth'])

# Agregar etiquetas y título
plt.xlabel("Variable")
plt.ylabel("Valores")
plt.title("Boxplot")

# Mostrar el gráfico
plt.show()
```




```
# Define the neural network model
model = keras.Sequential([
    layers.Dense(1024, activation="relu", input_shape=[len(train_features.keys())]),
    layers.Dense(512, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1)
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1024)              10240
```

```
 dense_1 (Dense)             (None, 512)               524800

 dense_2 (Dense)             (None, 128)               65664

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 611,073
Trainable params: 611,073
Non-trainable params: 0
_________________________________________________________________
```

```python
# Compile the model
model.compile(loss="mse", optimizer='adam', metrics=["mae", "mse"])
```

```python
# Train the model
history = model.fit(train_features, train_labels, epochs=500, validation_data=(test_features, test_labels))
```

```
Epoch 1/500
496/496 [==============================] - 3s 6ms/step - loss: 134468.0469 - mae: 264.4326 - mse: 134468.0469 - val_loss: 343705.6562
Epoch 2/500
496/496 [==============================] - 2s 5ms/step - loss: 131925.6406 - mae: 262.8116 - mse: 131925.6406 - val_loss: 346217.2812
Epoch 3/500
496/496 [==============================] - 2s 5ms/step - loss: 131113.0938 - mae: 261.2518 - mse: 131113.0938 - val_loss: 343563.8438
Epoch 4/500
496/496 [==============================] - 2s 5ms/step - loss: 133078.2031 - mae: 264.7411 - mse: 133078.2031 - val_loss: 345997.1875
Epoch 5/500
496/496 [==============================] - 2s 5ms/step - loss: 128812.6719 - mae: 258.5217 - mse: 128812.6719 - val_loss: 354618.9375
Epoch 6/500
496/496 [==============================] - 3s 6ms/step - loss: 133025.0938 - mae: 262.4914 - mse: 133025.0938 - val_loss: 352738.1875
Epoch 7/500
496/496 [==============================] - 2s 5ms/step - loss: 131261.3750 - mae: 260.9868 - mse: 131261.3750 - val_loss: 333681.4375
Epoch 8/500
496/496 [==============================] - 2s 5ms/step - loss: 128403.3125 - mae: 259.0532 - mse: 128403.3125 - val_loss: 348047.5000
Epoch 9/500
496/496 [==============================] - 2s 5ms/step - loss: 130128.1094 - mae: 260.3539 - mse: 130128.1094 - val_loss: 347023.2188
Epoch 10/500
496/496 [==============================] - 2s 5ms/step - loss: 126940.2656 - mae: 256.2032 - mse: 126940.2656 - val_loss: 342735.1250
Epoch 11/500
496/496 [==============================] - 3s 7ms/step - loss: 130488.7656 - mae: 260.9501 - mse: 130488.7656 - val_loss: 342543.2188
Epoch 12/500
496/496 [==============================] - 2s 5ms/step - loss: 126473.4766 - mae: 255.9786 - mse: 126473.4766 - val_loss: 344284.5312
Epoch 13/500
496/496 [==============================] - 2s 5ms/step - loss: 129733.8750 - mae: 261.0691 - mse: 129733.8750 - val_loss: 366165.6875
Epoch 14/500
496/496 [==============================] - 2s 5ms/step - loss: 128634.8125 - mae: 258.9760 - mse: 128634.8125 - val_loss: 377042.0938
Epoch 15/500
496/496 [==============================] - 2s 5ms/step - loss: 131271.2500 - mae: 261.3327 - mse: 131271.2500 - val_loss: 345416.0312
Epoch 16/500
496/496 [==============================] - 3s 6ms/step - loss: 129564.4219 - mae: 260.1589 - mse: 129564.4219 - val_loss: 363260.4375
Epoch 17/500
496/496 [==============================] - 2s 5ms/step - loss: 131995.6875 - mae: 263.6449 - mse: 131995.6875 - val_loss: 357567.7500
Epoch 18/500
496/496 [==============================] - 2s 5ms/step - loss: 128086.7031 - mae: 258.7032 - mse: 128086.7031 - val_loss: 348725.6250
Epoch 19/500
496/496 [==============================] - 2s 5ms/step - loss: 129319.3984 - mae: 258.9720 - mse: 129319.3984 - val_loss: 343971.6875
Epoch 20/500
496/496 [==============================] - 2s 5ms/step - loss: 130234.3281 - mae: 260.8618 - mse: 130234.3281 - val_loss: 347876.7188
Epoch 21/500
496/496 [==============================] - 3s 6ms/step - loss: 128037.4375 - mae: 257.7584 - mse: 128037.4375 - val_loss: 357945.6562
Epoch 22/500
496/496 [==============================] - 2s 5ms/step - loss: 125820.8203 - mae: 255.8233 - mse: 125820.8203 - val_loss: 351307.3750
Epoch 23/500
496/496 [==============================] - 2s 5ms/step - loss: 124873.1484 - mae: 255.1273 - mse: 124873.1484 - val_loss: 356884.9688
Epoch 24/500
496/496 [==============================] - 2s 5ms/step - loss: 127358.7578 - mae: 257.0789 - mse: 127358.7578 - val_loss: 356073.7812
Epoch 25/500
496/496 [==============================] - 2s 5ms/step - loss: 125365.6094 - mae: 255.1174 - mse: 125365.6094 - val_loss: 353425.2500
Epoch 26/500
496/496 [==============================] - 3s 7ms/step - loss: 125677.8594 - mae: 256.7423 - mse: 125677.8594 - val_loss: 364988.7812
Epoch 27/500
496/496 [==============================] - 2s 5ms/step - loss: 126981.3125 - mae: 255.9910 - mse: 126981.3125 - val_loss: 340637.4375
Epoch 28/500
496/496 [==============================] - 2s 5ms/step - loss: 126284.4922 - mae: 255.2047 - mse: 126284.4922 - val_loss: 347311.5312
Epoch 29/500
```

```
# Evaluate the model on the test set
test_loss, test_mae, test_mse = model.evaluate(test_features, test_labels)
```

```
plt.plot(history.history["mse"])
plt.plot(history.history["val_mse"])
plt.title("Model MSE")
plt.ylabel("MSE")
plt.xlabel("Epoch")
plt.legend(["Train", "Validation"], loc="upper left")
plt.show()
```

```
# Make predictions on new data
new_data = np.array([[0.23,61.5,55,3.95,3.98,2.43,2,1,3]])
new_data = (new_data - train_mean.values.reshape(1, -1)) / train_std.values.reshape(1, -1)
prediction = model.predict(new_data)
print("Predicción del precio:", prediction)
```

```
predicciones = model.predict(test_features)
inicio = 50
fin=200
# Crear la gráfica de predicción por muestra
plt.figure(figsize=(10, 6))
a=plt.axes(aspect='equal')
plt.scatter(test_labels,predicciones)
plt.xlabel('Muestra')
plt.ylabel('Valor')
plt.title('Predicción por muestra del conjunto de pruebas')
_=plt.plot(test_labels,test_labels,color='orange')
plt.show()
```

```
plt.figure(figsize=(10, 6))
plt.plot(range(inicio,fin), test_labels[inicio:fin])
plt.plot(range(inicio,fin), predicciones[inicio:fin])
plt.xlabel('Muestra')
plt.ylabel('Valor')
plt.title('Valores de test_labels')
plt.show()
```

```
test_labels
```

```
predicciones
```

```
# Plot each feature along with the predicted values
fig, axs = plt.subplots(3, 4, figsize=(15, 10))
axs = axs.flatten()

for i, column in enumerate(train_features.columns):
    axs[i].scatter(train_features[column], train_labels, s=2, color='black')
    #axs[i].scatter(test_features[column], test_labels, s=2, color='red')
    axs[i].scatter(train_features[column], model.predict(train_features), s=2, color='red')
    #axs[i].plot(test_features[column], model.predict(test_features), linewidth=1)
```