PRÁCTICA 2

En primer lugar, como es común en Jupyter Notebook, se procede a importar todas las librerias y que se usarán durante la práctica

In [1]:

```
import pandas
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm, shapiro, mannwhitneyu
from sklearn import preprocessing, metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from scipy import stats
import re
import statistics as st
```

1. Descripción del dataset

Carga del DataSet

A continuación, procedemos a cargar el dataset extraído de Kaggle, del link: https://www.kaggle.com/ionaskel/laptop-prices (https://www.kaggle.com/ionaskel/laptop-prices). Se ha creado un enlace al link de la práctica para proceder a la carga del archivo.

In [2]:

```
# Se carga el archivo con referencia al gitHub donde se encuentra el mismo
laptops_file = 'https://raw.githubusercontent.com/carlosalloUOC/PRA2-Limpieza-Analisis/main/csv/laptops_inicial.csv'

# Se lee el fichero anterior, para transformarlo en un dataframe indicándole que en la primera línea se encuentran las cabecera
laptops_initial = pandas.read_csv(laptops_file, header=0, encoding='latin-1')

# Comprobamos que tiene las dimensiones correctas
print(laptops_initial.shape)
```

```
(1303, 13)
```

Se aprecia como se ha importado un dataset de 1303 filas y 13 columnas. Se puede comprabar que este hecho es correcto, ya que en la web mencionada donde se han descargado los datos se proporciona esta información. A continuación, se procede a imprimir las 5 primeras filas del dataframe, para entender el tipo de datos con el que se va a trabajar:

In [3]:

```
laptops_initial.head(5)
```

Out[3]:

	Unnamed: 0	Company	Product	TypeName	Inches	ScreenResolution
0	1	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600
1	2	Apple	Macbook Air	Ultrabook	13.3	1440x900
2	3	HP	250 G6	Notebook	15.6	Full HD 1920x1080
3	4	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800
4	5	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600

Descripción de variables

Se aprecia que se cuentan con las siguientes columnas:

- Unnamed: 0
- Company
- Product
- TypeName
- Inches
- ScreenResolution
- Cpu
- Ram
- Memory
- Gpu
- OpSys
- Weight
- Price_euros

Aunque su nombre es bastante descriptivo, en la documentación se muestra la siguente información sobre estas columnas:

- Unnamed: 0 --> No da información, pero por su estructura en los datos, se aprecia que tiene tiene la mera función de ID_descendente para cada una de las columnas
- Company --> Company Name
- Product --> Product Name
- TypeName --> Laptop Type
- Inches --> Screen Inches
- ScreenResolution --> Screen Resolution
- Cpu --> CPU Model
- Ram --> RAM Characteristics
- Memory --> Memory
- Gpu --> GPU Characteristics
- OpSys --> Operating System
- Weight --> Laptop's Weight
- Price_euros --> Laptop's Price

Además se aprecia que, para valores numéricos, tan solo se presenta sin unidades de medida el caso del precio (donde se indica que son euros en el nombre de la variable (Price_euros)), al igual que ocurre con el tamaño que ya se da en pulgadas. Para otros casos, como el caso del peso, la memoria o la RAM, no ocurre este hecho al proporcionarse la unidad de medida junto con el valor. Esto, como se verá en pasos

posteriores, será objeto de tratamiento para poder hacer de esta manera el estudio de homegeneidad, realizar representaciones o por si se quieren construir modelos a partir de los datos.

En función del perfil del comprador, el desconocimiento de la gama de productos y sus características puede que los lleve a tomar una decisión de compra poco adecuada o ajustada en precio. Por este motivo, sería interesante conocer qué variables o características son más influyentes en el precio a la hora de compra de un ordenador, para que de esta manera, el comprador:

- Pueda hacerse una idea del presupuesto aproximado que tendrá el ordenador que desea en base a las características técnicas deseadas.
- Pueda conocer si es verdad que algunas marcas, como Apple, tienen un precio algo superior al resto de las marcas.
- Pueda conocer qué características son las más influyentes en el precio para en base a estas, poder centrarse en lo que realmente necesita para incrementar o decrementar su presupuesto.

2. Integración y selección

En esta sección, los datos con los que se van a trabajar vienen todos en un mismo dataSet, por lo que no requiere ninguna integración adicional a la importación realizada. Adicionalmente, cabe decir que si se quisieran usar dos dataSet diferentes con las mismas características, se debería de realizar una integración de los mismos en donde habría que tener en cuenta posibles repeticiones de objetos, que las propiedades se presenten en las mismas unidades... Este proceso se realizó en la práctica 1, en el momento en el que se integraban dos dataSets diferentes (uno de cada web en donde se realizó WebScraping), en uno sólo.

Respecto a la selección de los datos, nos quedaremos con todas las filas, ya que cada una de ellas corresponde a un ordenador diferente y aporta información al estudio que se está realizando. Ademas, nos encontramos ante un número de ordenadores (1003) no tan grande como para tener que hacer reducción de la cantidad. Sin embargo, para el caso de tener que realizar el mismo, consideramos que las dos mejores formas de hacerlo serían el método de muestra aleatoria simple sin sustitución (para no tener repeticiones de ordenadores en el dataSet resultante), o muestra de clústeres, en donde cada cluster podría estar correspondido por la marca o por intervalos de precio para asegurar que tenemos muestras de todos los precios.

Si que en este apartado sería interesante hacer una división de los datos en muestra y test para por si posteriormente se usaran datos para realizar modelos. Sin embargo, este paso se realizará más adelante cuando se haya hecho el proceso de limpieza completo de los datos, ya que sino habría que realizar el proceso dos veces.

Sobre las columnas, encontramos la primera de ellas que no nos da ninguna información útil para el estudio y ser un simple id ascendente, con lo cual la eliminaremos. Haremos lo mismo con la columna que se refiere al producto, pues el objetivo en todo momento es realizar una comparación en base a características o marcas, pero no en base al modelo del portátil directamente.

In [4]:

```
# Eliminamos la columna Unnamed: 0
del laptops_initial["Unnamed: 0"]

# Eliminamos la columna Product
del laptops_initial["Product"]

#Imprimimos los datos para comprobar la correcta eliminación
laptops_initial.head(5)
```

Out[4]:

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram	Mem
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128 S
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128 Fl Stor
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256 S
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512 S
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256 S

3. Limpieza de los datos

Elementos vacíos I

En este apartado se prestará atención a los valores que se presentan como nulos o incluso a los que, sin ser nulos, presentan valores anormales que indicarían que nos encontramos ante un caso de elemento vacío o perdido (por ejemplo, el hecho de que en pulgadas se encuentre un 0)

Se comenzará con los valores nulos del dataFrame. Para ello, en primer lugar se hará un resumen de valores nulos por columna:

In [5]:

```
laptops_initial.isnull().sum()
```

Out[5]:

```
Company             0
TypeName            0
Inches              0
ScreenResolution    0
Cpu                 0
Ram                 0
Memory              0
Gpu                 0
OpSys               0
Weight              0
Price_euros         0
dtype: int64
```

Vemos como no hay ningún valor nulo. Para datasets con más columnas es más útil ver si hay algún valor nulo por medio del booleano, para que si este resulta verdadero, mirar dónde está el mismo

In [6]:

```
laptops_initial.isnull().values.any()
```

Out[6]:

```
False
```

Aunque no se han encontrado datos vacíos, se ha de prestar atención a que no se presenten en forma de string del tipo "No available". Esto se realizará en el apartado "Elementos vacíos II", una vez los datos hayan sido sido preparados para el estudio

Preparación de datos

Es interesante realizar una limpieza de las variables para su posterior tratamiento.

En primer lugar, centraremos la atención para que las variables que puedan ser consideradas como numéricas (Inches, Ram, Weight o Price_euros), sean tratadas así. Para el caso de Inches o Price_euros esto ya ocurre, pero para Ram o Weight no ocurre al aparecer sus unidades de medida como son GB o Kg.

Aunque parece que en el dataSet todas las medidas están dadas en GB y Kg, lo primero que se realizará será ver si para todas las columnas se cumple este hecho. Comenzamos por la Ram:

In [7]:

```
#Se realiza un splitado en base a 'GB'
split_ram_value = laptops_initial['Ram'].str.split('GB', 0, expand=True
)
#Se comprueba que todos los valores son numericos (no lo serían si hubi
era otra medida)
split_ram_value[0].str.isnumeric().unique()
```

Out[7]:

```
array([ True])
```

Se aprecia que todos los valores que se encuentran en la variable RAM vienen dados en GB, ya que al realizar el splitado se ha comprobado que en la primera posición del resultado se encuentran todo numeros enteros, por lo que será usaran estos valores numéricos posteriormente (almacenados en split_ram_value[0]).

De forma similar, se realizará el mismo estudio para la columna peso, peso con su unidad de Kg

In [8]:

```
# Se realiza un splitado en base a 'kg'
split_weight_value = laptops_initial['Weight'].str.split('kg', 0, expan
d=True)

# Se comprueba que todos los valores son enteros como en el caso anteri
or o float. Para ello, se define la función is_valid_decimal que
# indicará si un numero es dicimal (True) o no lo es:
def is_valid_decimal(string):
    try:
        float(string)
        return True
    except ValueError:
        return False

# Se crea un objeto serie para almacenar los resultados
is_weight_decimal = pandas.Series([])

# Se recorre la serie con los posibles números resultantes del splitad
o, y se van añadiendo a la serie creada
for index, value in split_weight_value[0].items():
    is_weight_decimal[index] = is_valid_decimal(value)

# Vemos los diferentes valores que contiene le vector (si todos son tru
e, todos serán valores float por lo que todos serán kg)
is_weight_decimal.unique()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
py:14: DeprecationWarning: The default dtype for empty Ser
ies will be 'object' instead of 'float64' in a future vers
ion. Specify a dtype explicitly to silence this warning.
```

Out[8]:

```
array([ True])
```

Por lo tanto, una vez se ha comprobado que ambas columnas eliminando la medida tiene un valor numérico, se procede a transformar estas columnas a numericas. Para ello, la unidad de medida, al ser en todas ellas la misma se incluirá en la columna a la que corresponde:

In [9]:

```
# Se renombran las columnas, añadiendo los GB a la RAM y los KG al peso
laptops_initial = laptops_initial.rename(columns={'Ram': 'Ram(GB)', 'We
ight': 'Weight(Kg)'})

# Se hace el camnio en estas columnas por los números extraídos
laptops_initial["Ram(GB)"] = split_ram_value[0]
laptops_initial["Weight(Kg)"] = split_weight_value[0]

# Se imprime la cabecera
laptops_initial.head(5)
```

Out[9]:

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	M
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	

Una vez se tienen estos datos correctamente transformados, se ponen de tipo numérico las columnas mencionadas

In [10]:

```
# Transformamos esta columnas al tipo numérico
laptops_initial["Ram(GB)"] = pandas.to_numeric(laptops_initial["Ram(GB)"])
laptops_initial["Weight(Kg)"] = pandas.to_numeric(laptops_initial["Weight(Kg)"])
laptops_initial["Inches"] = pandas.to_numeric(laptops_initial["Inches"])
laptops_initial["Price_euros"] = pandas.to_numeric(laptops_initial["Price_euros"])
laptops_initial.head(5)
```

Out[10]:

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	M
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	

Prestando atención a la variable Memoria, vemos que hay dos partes bastante diferenciadas. Aparentemente, en un primer lugar encontramos la unidad de memoria (que sería un caso parecido al atratado con la RAM), y por otro, el tipo de memoria. Por lo tanto, en un primer lugar se analizarán los tipos de memoria de los que disponemos, haciendo el splitado en base al primer espacio:

In [11]:

```
# Se realiza un splitado en base tan solo al primer espacio , ya que es
lo aparentemente separa la cantidad de memoria y el tipo
split_memory_value = laptops_initial['Memory'].str.split(' ', 1, expand
=True)

# Imprimimos las segundas partes del splitado
split_memory_value[1].unique()
```

Out[11]:

```
array(['SSD', 'Flash Storage', 'HDD', 'SSD +  1TB HDD',
       'SSD +  256GB SSD', 'SSD +  2TB HDD', 'Hybrid', 'SS
D +  500GB HDD',
       'SSD +  512GB SSD', 'Flash Storage +  1TB HDD', 'HD
D +  1TB HDD',
       'SSD +  1.0TB Hybrid'], dtype=object)
```

Vemos que nos encontramos ante un problema imprevisto. Al ser tantos datos, se acaba de comprobar que la estructura de cantidad de almacenamiento + tipo no siempre es así, pudiendo a veces combinarse combinaciones de varios tipos de memoria. Además, también con respecto a las unidades, parece que no todas son GB como en el caso anterior. Encontramos en este punto unidades tanto en GB como en TB.

Haremos un primer estudio sobre a cuántos datos afecta este hecho de estar formado por más de un tipo de memoria.

In [12]:

```
laptops_initial[laptops_initial['Memory'].str.contains(' + ')]
```

Out[12]:

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GE
21	Lenovo	Gaming	15.6	IPS Panel Full HD 1920x1080	Intel Core i5 7300HQ 2.5GHz	
28	Dell	Ultrabook	15.6	Full HD 1920x1080	Intel Core i7 8650U 1.9GHz	
37	Dell	Notebook	17.3	IPS Panel Full HD 1920x1080	Intel Core i5 8250U 1.6GHz	
41	Dell	Gaming	15.6	IPS Panel Full HD 1920x1080	Intel Core i7 7700HQ 2.8GHz	1
47	Asus	Gaming	17.3	Full HD 1920x1080	AMD Ryzen 1700 3GHz	
...	...	...	...	...	...	.
1238	MSI	Gaming	15.6	Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	
1245	Asus	Gaming	15.6	IPS Panel Full HD 1920x1080	Intel Core i7 7700HQ 2.8GHz	1
1247	Asus	Gaming	15.6	IPS Panel Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	1
1256	Asus	Gaming	17.3	IPS Panel Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	1
1259	MSI	Gaming	15.6	Full HD 1920x1080	Intel Core i7 6700HQ 2.6GHz	

208 rows × 11 columns

Contamos con 4 tipos de memoria:

- SSD
- Flash Storage
- HDD
- Hybrid

Que se presentan de forma única o con las siguientes combinaciones:

- SSD + SSD
- HDD + HDD
- SSD + HDD
- Flash Storage + HDD
- SSD + Hybrid

Teniendo estas en cuenta, se elaborará una serie para cada una de ellas, con el fin de tener controlado tanto el tipo de memoria como su valor. Se había pensado también en juntar los valores de ambas memorias en tan solo una columna, indicando en el tipo de memoria el conjunto de memorias a las que pertenece. Sin embargo, al realizar esto, se perdería información relevante, ya que no es lo mismo una memoria SSD y una HDD!

Adicionalmente, se escogerá la unidad de medida de GB, ya que es la que más presente está. Se deberá de tener en cuenta que 1TB serán 1024GB, ya que son las dos medidas que aparecen.

In [13]:

```
# Se crea un objeto serie para almacenar cada tipo de memoria
ssd_value = pandas.Series([])
flash_value = pandas.Series([])
hdd_value = pandas.Series([])
hybrid_value = pandas.Series([])

# Devuelve los GB a partir de la unidad de memoria proporcionada
def convert_GB(memoryNumber):
  if "GB" in memoryNumber:
    return (float(memoryNumber.split('GB')[0]))
  else:
    return (float(memoryNumber.split('TB')[0])*1024)

# Devuelve el tipo de memoria principal (la que aparece en primer luga
r)
def get_primary_type(secondaryMemory):
  secondaryMemory_split_plus = secondaryMemory.split(' +  ')
  return secondaryMemory_split_plus[0]

# Devuelve el tipo de memoria secundaria (la que aparece en segundo lug
ar)
def get_secondary_type(secondaryMemory):
  secondaryMemory_split_plus = secondaryMemory.split(' +  ')
  secondaryMemory_split_value =secondaryMemory_split_plus[1].split(' ')
  return secondaryMemory_split_value[1]

# Devuelve el valor de memoria secundaria (la que aparece en segundo lu
gar)
def get_secondary_value(secondaryMemory):
  secondaryMemory_split_plus = secondaryMemory.split(' +  ')
  secondaryMemory_split_value =secondaryMemory_split_plus[1].split(' ')
  return secondaryMemory_split_value[0]


# Se recorre la serie con los posibles números resultantes del splitad
o, y se van añadiendo a la serie creada
for index, value in split_memory_value[0].items():
  # El primer caso será el que incluya un + en el string, hecho que ind
ica que esta compuesto por dos tipos de memoria
  if " + " in split_memory_value[1][index]:
    # Se extrae el valor y el tipo secundario
    primary_type = get_primary_type(split_memory_value[1][index])
    secondary_value = get_secondary_value(split_memory_value[1][index])
    secondary_type = get_secondary_type(split_memory_value[1][index])
```

```
    # Se comprueban que si ambos son del mismo tipo (ocurre con SSD o H
DD), en cuyo caso habrá que sumar estos valores
    if (primary_type == secondary_type):
      if (primary_type == 'HDD'):
        hdd_value[index] = convert_GB(split_memory_value[0][index]) + c
onvert_GB(secondary_value)
        ssd_value[index] = 0
        hybrid_value[index] = 0
        flash_value[index] = 0
      else:
        ssd_value[index] = convert_GB(split_memory_value[0][index]) + c
onvert_GB(secondary_value)
        hdd_value[index] = 0
        hybrid_value[index] = 0
        flash_value[index] = 0
    # En el caso de que no sean igual, se trata de dos tipos de memoria
s diferentes, en donde teniendo en cuentas las combinaciones posibles e
n base a las combinaciones vistas anteriormente
    else:
      if (primary_type == 'SSD' and secondary_type == 'HDD'):
        ssd_value[index] = convert_GB(split_memory_value[0][index])
        hdd_value[index] = convert_GB(secondary_value)
        hybrid_value[index] = 0
        flash_value[index] = 0
      elif (primary_type == 'Flash Storage' and secondary_type == 'HDD'
):
        ssd_value[index] = 0
        hdd_value[index] = convert_GB(secondary_value)
        hybrid_value[index] = 0
        flash_value[index] = convert_GB(split_memory_value[0][index])
      elif (primary_type == 'SSD' and secondary_type == 'Hybrid'):
        ssd_value[index] = convert_GB(split_memory_value[0][index])
        hdd_value[index] = 0
        hybrid_value[index] = convert_GB(secondary_value)
        flash_value[index] = 0

  # En el caso de que no se encuentre la particula +, será porque sólo
 tiene un tipo de memoria, por lo que buscaremos cual es y lo introduci
remos a la serie, dejando el resto de valores
  # de la memoria que no corresponda a 0
  else:
    # Si es SSD, rellenamos la serie de SSD, mientras que en las demás
 introdimos un 0, al no contar con esa memoria
    if split_memory_value[1][index] == 'SSD':
      ssd_value[index] = convert_GB(split_memory_value[0][index])
      hdd_value[index] = 0
      hybrid_value[index] = 0
```

```
    flash_value[index] = 0
  elif split_memory_value[1][index] == 'HDD':
    hdd_value[index] = convert_GB(split_memory_value[0][index])
    ssd_value[index] = 0
    hybrid_value[index] = 0
    flash_value[index] = 0
  elif split_memory_value[1][index] == 'Hybrid':
    hybrid_value[index] = convert_GB(split_memory_value[0][index])
    ssd_value[index] = 0
    hdd_value[index] = 0
    flash_value[index] = 0
  elif split_memory_value[1][index] == 'Flash Storage':
    flash_value[index] = convert_GB(split_memory_value[0][index])
    ssd_value[index] = 0
    hdd_value[index] = 0
    hybrid_value[index] = 0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
py:2: DeprecationWarning: The default dtype for empty Seri
es will be 'object' instead of 'float64' in a future versi
on. Specify a dtype explicitly to silence this warning.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
py:3: DeprecationWarning: The default dtype for empty Seri
es will be 'object' instead of 'float64' in a future versi
on. Specify a dtype explicitly to silence this warning.
  This is separate from the ipykernel package so we can av
oid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
py:4: DeprecationWarning: The default dtype for empty Seri
es will be 'object' instead of 'float64' in a future versi
on. Specify a dtype explicitly to silence this warning.
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
py:5: DeprecationWarning: The default dtype for empty Seri
es will be 'object' instead of 'float64' in a future versi
on. Specify a dtype explicitly to silence this warning.
  """
```

Para comprobar que se ha hecho bien el cambio, se imprimirá uno de los casos especiales (que incluyen dos tipos de memoria), y que se ha visto en el printeo de los mismos. Este será la muestra 21, que tiene segun vemos en la tabla: 128GB SSD + 1TB HDD

In [14]:

```
print('Valor SSD en GB: ' + str(ssd_value[21]))
print('Valor HDD en GB: ' + str(hdd_value[21]))
```

```
Valor SSD en GB: 128.0
Valor HDD en GB: 1024.0
```

Una vez hecho este proceso, se colocarán las 4 nuevas columnas con cada tipo de memoria, siendo este un valor numérico:

In [15]:

```
# Introducimos las nuevas columnas
laptops_initial["MemorySSD(GB)"] = ssd_value
laptops_initial["MemoryHDD(GB)"] = hdd_value
laptops_initial["MemoryFlash(GB)"] = flash_value
laptops_initial["MemoryHybrid(GB)"] = hybrid_value

# Las transformamos a numéricas
laptops_initial["MemorySSD(GB)"] = pandas.to_numeric(laptops_initial["M
emorySSD(GB)"])
laptops_initial["MemoryHDD(GB)"] = pandas.to_numeric(laptops_initial["M
emoryHDD(GB)"])
laptops_initial["MemoryFlash(GB)"] = pandas.to_numeric(laptops_initial[
"MemoryFlash(GB)"])
laptops_initial["MemoryHybrid(GB)"] = pandas.to_numeric(laptops_initial
["MemoryHybrid(GB)"])

# Eliminamos la columna inicial que tenia los datos de la memoria sin l
impiar
del laptops_initial["Memory"]

# Verificamos la correcta creación
laptops_initial.head(5)
```

Out[15]:

	Company	TypeName	Inches	ScreenResolution	Cpu	Ram(GB)	
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	(
1	Apple	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	(
2	HP	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	(
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	(

Se procederá ahora a realizar una sepación de la CPU. Aparentemente, parece que en una primera partaparece la empresa que la proporciona, en segundo la versión que se proporciona, y por último, la velocidad del mismo. Como siempre, se comprueba gracias a la programación que para todos ocurre esto (ya que al contar con 1303 muestras, no nos podemos fijar tan solo en las 5 primeras)

In [16]:

```
# Se realiza un splitado en base al primer espacio, dejando en la prime
ra parte la supuesta marca, mientras que en la segunda el resto
split_cpu_company_value = laptops_initial['Cpu'].str.split(' ', 1, expa
nd=True)

# Almacenamos las marcas de cpu
company_cpu = split_cpu_company_value[0]

# Comprobamos que efectivamente todas ellas corresponden a marcas
company_cpu.unique()
```

Out[16]:

```
array(['Intel', 'AMD', 'Samsung'], dtype=object)
```

Se aprecia como en esta ocasión, se está en lo cierto y tan solo hay 3 marcas. Procederemos ahora a ver los tipos de productos trabajando con la parte del splitado anterior no usada

In [17]:

```
# Se realiza un splitado de la segunda parte, en donde el mismo se real
izará en base al último espacio, al ser la estructura VERSION + ' ' + V
ELOCIDAD
split_cpu_version_and_speed_value = split_cpu_company_value[1].str.rspl
it(' ', 1, expand=True)

# Sacamos las versiones, que estarán en la primera parte del splitado
version_cpu=split_cpu_version_and_speed_value[0]

# Comprobamos que todas ellas son versiones
version_cpu.unique()
```

Out[17]:

```
array(['Core i5', 'Core i5 7200U', 'Core i7', 'A9-Series 9
420',
       'Core i7 8550U', 'Core i5 8250U', 'Core i3 6006U',
'Core M m3',
       'Core i7 7500U', 'Core i3 7100U', 'Atom x5-Z8350',
       'Core i5 7300HQ', 'E-Series E2-9000e', 'Core i7 865
0U',
       'Atom x5-Z8300', 'E-Series E2-6110', 'A6-Series 922
0',
       'Celeron Dual Core N3350', 'Core i3 7130U', 'Core i
7 7700HQ',
       'Ryzen 1700', 'Pentium Quad Core N4200', 'Atom x5-Z
8550',
       'Celeron Dual Core N3060', 'FX 9830P', 'Core i7 756
0U',
       'E-Series 6110', 'Core i5 6200U', 'Core M 6Y75', 'C
ore i5 7500U',
       'Core i7 6920HQ', 'Core i5 7Y54', 'Core i7 7820HK',
       'Xeon E3-1505M V6', 'Core i7 6500U', 'E-Series 9000
e',
       'A10-Series A10-9620P', 'A6-Series A6-9220', 'Core
i7 6600U',
       'Celeron Dual Core 3205U', 'Core i7 7820HQ', 'A10-S
eries 9600P',
       'Core i7 7600U', 'A8-Series 7410', 'Celeron Dual Co
re 3855U',
       'Pentium Quad Core N3710', 'A12-Series 9720P', 'Cor
e i5 7300U',
       'Celeron Quad Core N3450', 'Core i5 6440HQ', 'Core
i7 6820HQ',
       'Ryzen 1600', 'Core i7 7Y75', 'Core i5 7440HQ', 'Co
re i7 7660U',
       'Core M m3-7Y30', 'Core i5 7Y57', 'Core i7 6700HQ',
       'Core i3 6100U', 'A10-Series 9620P', 'E-Series 711
0',
       'A9-Series A9-9420', 'Core i7 6820HK', 'Core M 7Y3
0',
       'Xeon E3-1535M v6', 'Celeron Quad Core N3160', 'Cor
e i5 6300U',
       'E-Series E2-9000', 'Celeron Dual Core N3050', 'Cor
e M M3-6Y30',
       'Core i5 6300HQ', 'A6-Series 7310', 'Atom Z8350',
       'Xeon E3-1535M v5', 'Core i5 6260U', 'Pentium Dual
Core N4200',
       'Celeron Quad Core N3710', 'Core M', 'A12-Series 97
```

```
00P',
       'Pentium Dual Core 4405U', 'A4-Series 7210', 'Core
i7 6560U',
       'Core M m7-6Y75', 'FX 8800P', 'Core M M7-6Y75', 'At
om X5-Z8350',
       'Pentium Dual Core 4405Y', 'Pentium Quad Core N370
0',
       'Core M 6Y54', 'Cortex A72&A53', 'E-Series 9000',
'Core M 6Y30',
       'A9-Series 9410'], dtype=object)
```

En este caso tenemos más versiones que marcas, pero vemos como todas ellas son versiones de los diferentes proveedores, por lo que tan solo quedará comprobar que, efectivamente, el final de este string coincide en todos casos con velocidades

In [18]:

```
# Sacamos las velocidades, que estarán en la seguna parte del splitado
 anterior
speed_cpu=split_cpu_version_and_speed_value[1]
speed_cpu.unique()
```

Out[18]:

```
array(['2.3GHz', '1.8GHz', '2.5GHz', '2.7GHz', '3.1GHz',
'3GHz', '2.2GHz',
       '1.6GHz', '2GHz', '2.8GHz', '1.2GHz', '2.9GHz', '2.
4GHz',
       '1.44GHz', '1.5GHz', '1.9GHz', '1.1GHz', '2.0GHz',
'1.3GHz',
       '2.6GHz', '3.6GHz', '1.60GHz', '3.2GHz', '1.0GHz',
'2.1GHz',
       '0.9GHz', '1.92GHz', '2.50GHz', '2.70GHz'], dtype=o
bject)
```

También se aprecia que todas son frecuencias y además en la misma medida, por lo que parece ser que estábamos en lo cierto en base a que lo que ocurre con las 5 primeras muestras, se replica para el resto de ellas. Por tanto, se dividirá esta columna en 3, ya que será más eficiente a la hora de hacer análisis y gráficas. Para le valor numérico, como en casos anteriores, se eliminará la unidad de medida de las muestras poniendo esta en la cabecera

In [19]:

```
# Se realiza un splitado en base a 'GHz'
speed_cpu_split = speed_cpu.str.split('GHz', 0, expand=True)
# Se crea la serie sin medidas
speed_cpu_GHz = speed_cpu_split[0]
```

Finalmente, se crean estas columnas y se le da el tipo numérico a la última de ellas

In [20]:

```
# Introducimos las nuevas columnas
laptops_initial["CPU_Company"] = company_cpu
laptops_initial["CPU_Version"] = version_cpu
laptops_initial["CPU_Speed(GHz)"] = speed_cpu_GHz

# La transformamos a numéricas
laptops_initial["CPU_Speed(GHz)"] = pandas.to_numeric(laptops_initial[
"CPU_Speed(GHz)"])

# Eliminamos la columna inicial que tenia los datos de la CPU sin limpi
ar
del laptops_initial["Cpu"]

# Verificamos la correcta creación
laptops_initial.head(5)
```

Out[20]:

	Company	TypeName	Inches	ScreenResolution	Ram(GB)	Gpu
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	8	Intel Iris Plus Graphics 640
1	Apple	Ultrabook	13.3	1440x900	8	Intel HD Graphics 6000
2	HP	Notebook	15.6	Full HD 1920x1080	8	Intel HD Graphics 620
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	16	AMD Radeon Pro 455
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	8	Intel Iris Plus Graphics 650

Se aprecia que para la GPU parece seguir una estructura similar: VENDEDOR + VERSION. Siguiendo el mismo proceso que el anterior, nos encontraríamos:

In [21]:

```
# Se realiza un splitado en base al primer espacio, dejando en la prime
ra parte la supuesta marca, mientras que en la segunda la versión
split_gpu_company_and_model = laptops_initial['Gpu'].str.split(' ', 1,
expand=True)

# Almacenamos las marcas de Gpu
company_gpu = split_gpu_company_and_model[0]

# Comprobamos que efectivamente todas ellas corresponden a marcas
company_gpu.unique()
```

Out[21]:

```
array(['Intel', 'AMD', 'Nvidia', 'ARM'], dtype=object)
```

In [22]:

```
# Almacenamos las versiones de Gpu
version_gpu = split_gpu_company_and_model[1]

# Comprobamos que efectivamente todas ellas corresponden a marcas
version_gpu.unique()
```

Out[22]:

```
array(['Iris Plus Graphics 640', 'HD Graphics 6000', 'HD G
raphics 620',
       'Radeon Pro 455', 'Iris Plus Graphics 650', 'Radeon
R5',
       'Iris Pro Graphics', 'GeForce MX150', 'UHD Graphics
620',
       'HD Graphics 520', 'Radeon Pro 555', 'Radeon R5 M43
0',
       'HD Graphics 615', 'Radeon Pro 560', 'GeForce 940M
X',
       'HD Graphics 400', 'GeForce GTX 1050', 'Radeon R2',
'Radeon 530',
       'GeForce 930MX', 'HD Graphics', 'HD Graphics 500',
       'GeForce 930MX ', 'GeForce GTX 1060', 'GeForce 150M
X',
       'Iris Graphics 540', 'Radeon RX 580', 'GeForce 920M
X',
       'Radeon R4 Graphics', 'Radeon 520', 'GeForce GTX 10
70',
       'GeForce GTX 1050 Ti', 'GeForce MX130', 'R4 Graphic
s',
       'GeForce GTX 940MX', 'Radeon RX 560', 'GeForce 920
M',
       'Radeon R7 M445', 'Radeon RX 550', 'GeForce GTX 105
0M',
       'HD Graphics 515', 'Radeon R5 M420', 'HD Graphics 5
05',
       'GTX 980 SLI', 'R17M-M1-70', 'GeForce GTX 1080', 'Q
uadro M1200',
       'GeForce 920MX ', 'GeForce GTX 950M', 'FirePro W419
0M ',
       'GeForce GTX 980M', 'Iris Graphics 550', 'GeForce 9
30M',
       'HD Graphics 630', 'Radeon R5 430', 'GeForce GTX 94
0M',
       'HD Graphics 510', 'HD Graphics 405', 'Radeon RX 54
0',
       'GeForce GT 940MX', 'FirePro W5130M', 'Quadro M2200
M', 'Radeon R4',
       'Quadro M620', 'Radeon R7 M460', 'HD Graphics 530',
       'GeForce GTX 965M', 'GeForce GTX1080', 'GeForce GTX
1050 Ti',
       'GeForce GTX 960M', 'Radeon R2 Graphics', 'Quadro M
620M',
       'GeForce GTX 970M', 'GeForce GTX 960<U+039C>', 'Gra
```

```
phics 620',
       'GeForce GTX 960', 'Radeon R5 520', 'Radeon R7 M44
0', 'Radeon R7',
       'Quadro M520M', 'Quadro M2200', 'Quadro M2000M', 'H
D Graphics 540',
       'Quadro M1000M', 'Radeon 540', 'GeForce GTX 1070M',
       'GeForce GTX1060', 'HD Graphics 5300', 'Radeon R5 M
420X',
       'Radeon R7 Graphics', 'GeForce 920', 'GeForce 940
M',
       'GeForce GTX 930MX', 'Radeon R7 M465', 'Radeon R3',
       'GeForce GTX 1050Ti', 'Radeon R7 M365X', 'Radeon R9
M385',
       'HD Graphics 620 ', 'Quadro 3000M', 'GeForce GTX 98
0 ',
       'Radeon R5 M330', 'FirePro W4190M', 'FirePro W6150
M',
       'Radeon R5 M315', 'Quadro M500M', 'Radeon R7 M360',
       'Quadro M3000M', 'GeForce 960M', 'Mali T860 MP4'],
dtype=object)
```

Vemos como de esta manera facilitamos análisis posteriores, sobre todo en lo referente a la marca, en donde podremos estudiar si la misma influye en el precio. Añadiendo estas columnas a nuestro dataFrame

In [23]:

```
# Introducimos las nuevas columnas
laptops_initial["GPU_Company"] = company_gpu
laptops_initial["GPU_Version"] = version_gpu

# Eliminamos la columna inicial que tenia los datos de la GPU sin limpi
ar
del laptops_initial["Gpu"]

# Verificamos la correcta creación
laptops_initial.head(5)
```

Out[23]:

	Company	TypeName	Inches	ScreenResolution	Ram(GB)	OpSys	V
0	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	8	macOS	
1	Apple	Ultrabook	13.3	1440x900	8	macOS	
2	HP	Notebook	15.6	Full HD 1920x1080	8	No OS	
3	Apple	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	16	macOS	
4	Apple	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	8	macOS	

Finalmente, para la resolución, la divideremos en dos partes. La primera de ellas, de extraer la resolución en píxeles por una expresión regex:

In [24]:

```
# Limpieza columna ScreenResolution (se dejan los valores que indican ú
nicamente la resolución en píxeles)
pixels_ScreenResolution = laptops_initial.ScreenResolution.str.extract(
"(\d+x\d+)", expand=False)

# Se comprueba que no hay nulos (que para todos los valores se ha conse
guido obtener los pixeles)
pixels_ScreenResolution.isnull().sum()
```

Out[24]:

```
0
```

Para poder tratarlas numéricamente, se extraerá el la parte de los píxeles a lo alto y a lo ancho, separados en todas las medidas por una x

In [25]:

```
# Se realiza un splitado en base a la x
split_pixels_value = pixels_ScreenResolution.str.split('x', 0, expand=T
rue)

# Se separan en ancho (width) y alto (high)
split_pixels_width = split_pixels_value[0]
split_pixels_high = split_pixels_value[1]
```

Una vez realizado este hecho, se extraerá también tipo de resolución que presenta el ordenador. Mientras que todas las muestras traían los pixeles, en este caso no ocurrirá lo mismo, ya que se ve en las 5 primeras muestras como en la segunda (id 1), no se presenta el tipo.

In [26]:

```
# Se realiza un splitado en base al último espacio, ya que si tiene tip
o, en la primera variable de este splitado estará el mismo
# y si no lo tiene, estarán los píxeles sacados anteriormente
split_pixels_by_last_space = laptops_initial.ScreenResolution.str.rspli
t(' ', 1, expand=True)

# Se crea un objeto serie para almacenar los tipos
type_pixels = pandas.Series([])

# Comprobamos uno a uno si son igualas a los píxeles (no tiene tipo) o
 si no lo son (tiene el tipo)
for index, value in split_pixels_by_last_space[0].items():
  if (split_pixels_by_last_space[0][index] == pixels_ScreenResolution[i
ndex]):
    type_pixels[index]=np.NaN
  else:
    type_pixels[index]=split_pixels_by_last_space[0][index]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
py:6: DeprecationWarning: The default dtype for empty Seri
es will be 'object' instead of 'float64' in a future versi
on. Specify a dtype explicitly to silence this warning.
```

En este caso, como es de esperar aparecerán nulos que serán tratados en la sección Elementos vacíos II. Introducimos los nuevos valores en el dataframe

In [27]:

```
# Introducimos las nuevas columnas
laptops_initial["ScreenResolution_Type"] = type_pixels
laptops_initial["ScreenResolution_Width"] = split_pixels_width
laptops_initial["ScreenResolution_High"] = split_pixels_high

# La transformamos a numéricas las dos últimas
laptops_initial["ScreenResolution_Width"] = pandas.to_numeric(laptops_i
nitial["ScreenResolution_Width"])
laptops_initial["ScreenResolution_High"] = pandas.to_numeric(laptops_in
itial["ScreenResolution_High"])

# Eliminamos la columna inicial que tenia los datos de ScreenResolution
sin limpiar
del laptops_initial["ScreenResolution"]

#Verificamos la correcta creación
laptops_initial.head(5)
```

Out[27]:

	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Price_e
0	Apple	Ultrabook	13.3	8	macOS	1.37	133
1	Apple	Ultrabook	13.3	8	macOS	1.34	89
2	HP	Notebook	15.6	8	No OS	1.86	57
3	Apple	Ultrabook	15.4	16	macOS	1.83	253
4	Apple	Ultrabook	13.3	8	macOS	1.37	180

Elementos vacíos II

En este momento, se comprobará si se han introducido valores nulos en el dataFrame. Tan sólo sería esperado en el apartado de tipo de resolución de pantalla, en donde se ha visto que algunas no disponían de esta información

In [28]:

```
laptops_initial.isnull().sum()
```

Out[28]:

```
Company                     0
TypeName                    0
Inches                      0
Ram(GB)                     0
OpSys                       0
Weight(Kg)                  0
Price_euros                 0
MemorySSD(GB)               0
MemoryHDD(GB)               0
MemoryFlash(GB)             0
MemoryHybrid(GB)            0
CPU_Company                 0
CPU_Version                 0
CPU_Speed(GHz)              0
GPU_Company                 0
GPU_Version                 0
ScreenResolution_Type     314
ScreenResolution_Width      0
ScreenResolution_High       0
dtype: int64
```

Efectivamente, se aprecia como para este caso se encuentran solo valores vacíos en la variable ScreenResolution_Type. Para su tratamiento, se realizará el método de indiciar los valores perdidos como la sustitución por una misma constante o etiqueta, en este caso, "Unknown". Esto se hace ya que falta un gran número de registro (un 33%), y el uso de otras técnicas como la moda harían que tuvieramos muchísimos datos 'no reales'. Otras técnicas basadas en modelos que lo predicen, tampoco las vemos oportunadas, ya que siendo un atributo como es el tipo de resolución, podría darse el caso de que dos ordenadores tuvieran las mismas características, y sin embargo, un tipo de resolución diferencia. Por tanto, aplicando esto al dataFrame:

In [29]:

```
# Cambiamos los valores nulos por el literal Unknow
laptops_initial["ScreenResolution_Type"].fillna("Unknown", inplace = Tr
ue)

# Verificamos que no queda ningún valor nulo
laptops_initial.ScreenResolution_Type.isnull().sum()
```

Out[29]:

```
0
```

Finalemente, para las variables con las que se han tratado, se ha visto como no hay valores que indiquen valores nulos o vacíos. Faltaría tratar ver los valores de las columnas que no se han tratado todavía:

- Company
- TypeName
- OpSys

Para comprobar que no hay valores nulos en estas variables, se imprimirán los diferentes valores que contiene cada una

In [30]:

```
laptops_initial['Company'].unique()
```

Out[30]:

```
array(['Apple', 'HP', 'Acer', 'Asus', 'Dell', 'Lenovo', 'C
huwi', 'MSI',
       'Microsoft', 'Toshiba', 'Huawei', 'Xiaomi', 'Vero',
'Razer',
       'Mediacom', 'Samsung', 'Google', 'Fujitsu', 'LG'],
dtype=object)
```

In [31]:

```
laptops_initial['TypeName'].unique()
```

Out[31]:

```
array(['Ultrabook', 'Notebook', 'Netbook', 'Gaming', '2 in
1 Convertible',
       'Workstation'], dtype=object)
```

In [32]:

```
laptops_initial['OpSys'].unique()
```

Out[32]:

```
array(['macOS', 'No OS', 'Windows 10', 'Mac OS X', 'Linu
x', 'Android',
       'Windows 10 S', 'Chrome OS', 'Windows 7'], dtype=ob
ject)
```

Se puede apreciar como para todos los casos se encuentran valores no vacíos. Si bien es verdad que para sistema operativo encontramos 'No OS', que puede dar lugar a confusión. Sin embargo, este valor es válido, ya que algunos ordenadores pueden no tener sistema operativo integradado, y por tanto, posiblemente, abarate el coste del mismo

Valores extremos

Se analizarán a continuación los valores extremos, empezando por la variable más relevante para el estudio: price_euros. En un primer momento, se realizará un análisis descriptivo de esta variable numérica, estudiando sus diferentes percentiles:

In [33]:

```
# Percentiles y resumen estadístico de la columna de precios
laptops_initial["Price_euros"].describe()
```

Out[33]:

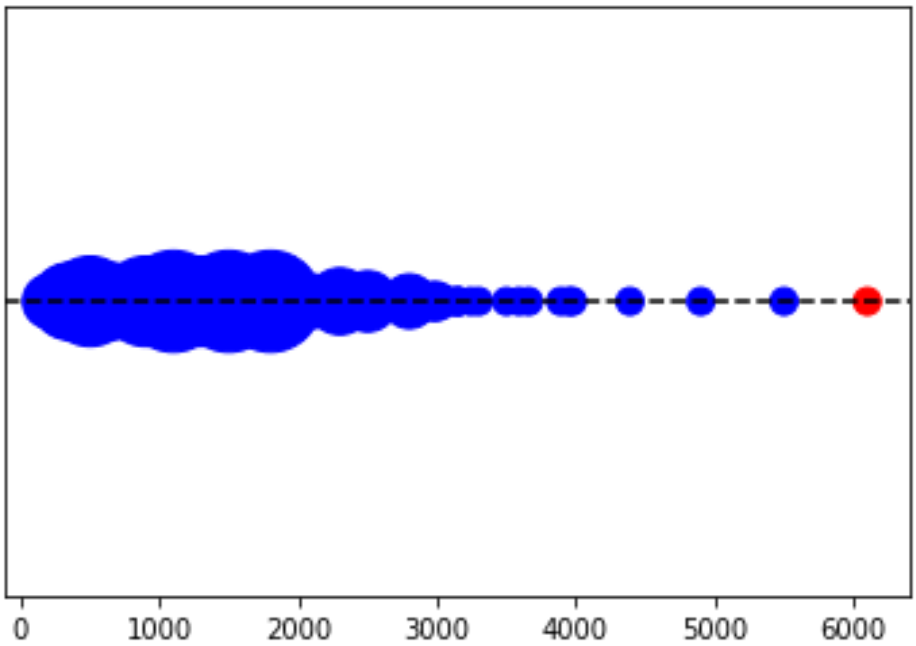
```
count    1303.000000
mean     1123.686992
std       699.009043
min       174.000000
25%       599.000000
50%       977.000000
75%      1487.880000
max      6099.000000
Name: Price_euros, dtype: float64
```

Se aprecia como aparentemente, aunque el percentil 50 está en 977 y su 75 en 1487, la media es de 1123, lo que es indicio de que, tras el percentil 75 encontraremos algún valor más elevado, hasta llegar al máximo de 6099. Estudiaremos gráficamente en una recta la situación de estos datos

In [34]:

```
# Extraemos cada precio y el número de veces que salen
price_unique, counts = np.unique(laptops_initial['Price_euros'], return
_counts=True)

# Representamos de azul los valores en donde se concentran la mayoría d
e las filas
# y en rojo las que se alejan de esta
sizes = counts*100
colors = ['blue']*len(price_unique)
colors[-1] = 'red'
plt.axhline(1, color='k', linestyle='--')
plt.scatter(price_unique, np.ones(len(price_unique)), s=sizes, color=co
lors)
plt.yticks([])
plt.show()
```



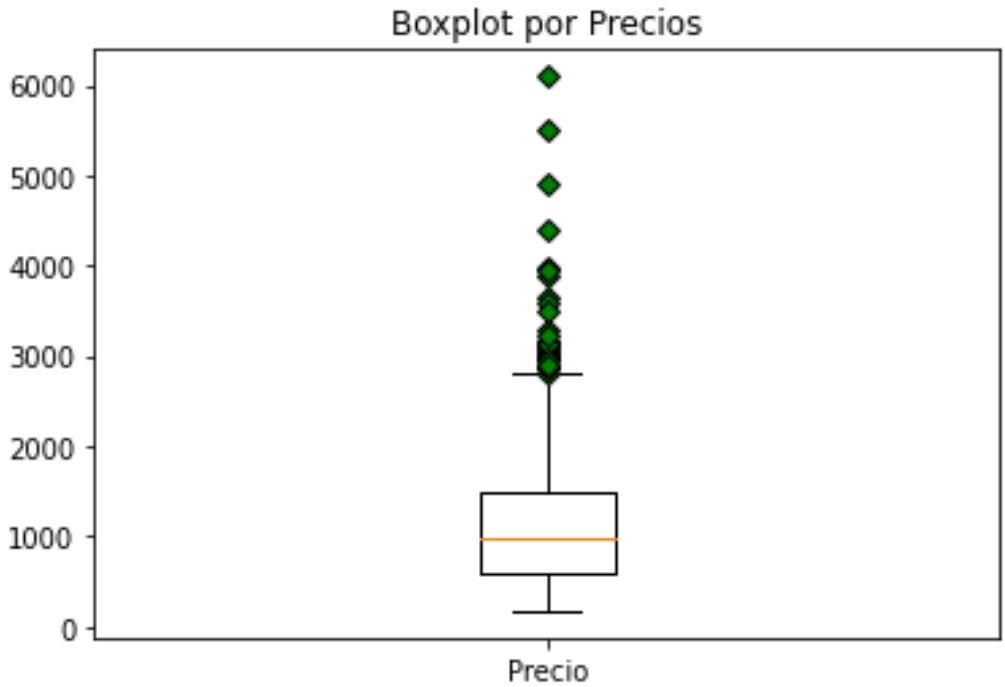
Se aprecia como hay una gran concentración de datos en la zona en torno a 1000 euros, y que va disminuyendo hasta llegar hasta los 3000, en donde los precios a partir de este punto empiezan a encontrarse más distantes hasta llegar a los 6000, que la función cataloga como punto alejado o outlier. Una forma más visual y algo más matemática de verlo sería por medio de los boxplots:

In [35]:

```
# Construción de boxplot
green_diamond = dict(markerfacecolor='g', marker='D')
fig, ax = plt.subplots()
ax.set_title('Boxplot por Precios')
ax.boxplot(laptops_initial['Price_euros'], flierprops=green_diamond, la
bels=["Precio"])
```

Out[35]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x7fbc7b8fb650>],
 'caps': [<matplotlib.lines.Line2D at 0x7fbc7b9016d0>,
  <matplotlib.lines.Line2D at 0x7fbc7b901c10>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fbc7b90a710>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7fbc7b90a1d0>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7fbc7b8fbc10>,
  <matplotlib.lines.Line2D at 0x7fbc7b901190>]}
```



En este, se aprecia información como la que se ha comentado en párrafos anteriores. En esta gráfica se aprecia como, los puntos por encima de la T, son catalogados como extremos o anómalos. Estudiemos en el dataSet estos casos, para ver si efectivamente se tratan de outliers que debemos de eliminar o tratar al ser posibles valores incorrectos o los mismos son puntos de interés de nuestro análisis y han de estar presentes.

In [36]:

```
# Ordenar los ordenadores por precio, e imprimir las columnas más caras
laptops_initial.sort_values('Price_euros').tail(10)
```

Out[36]:

	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Pr
1231	Razer	Gaming	14.0	16	Windows 10	1.95	
780	Dell	Gaming	17.3	32	Windows 10	4.42	
723	Dell	Gaming	17.3	32	Windows 10	4.36	
238	Asus	Gaming	17.3	32	Windows 10	4.70	
1136	HP	Workstation	17.3	8	Windows 7	3.00	
1066	Asus	Gaming	17.3	64	Windows 10	3.58	
749	HP	Workstation	17.3	16	Windows 7	3.00	
610	Lenovo	Notebook	15.6	32	Windows 10	2.50	
830	Razer	Gaming	17.3	32	Windows 10	3.49	
196	Razer	Gaming	17.3	32	Windows 10	3.49	

Se aprecia como sin duda, en la gran mayoría encontramos en esta sección ordenadores Gaming, los cuales necesitan una capacidad mayores que los ordenadores estándar para el día a día. Además, vemos por ejemplo que la diferencia entre los dos últimos reside en la diferencia de la memoria SSD, ya que las demás características son iguales. Por lo general, vemos que todos estos ordenadores, presentan una CPU y GPU muy potente, que consultando el precio de estas en el mercado, parece ser un elemento que aumentará bastante el precio. Para confirmar este hecho, vamos a realizar un resumen de estos ordenadores

In [37]:

```
laptops_initial[laptops_initial['TypeName']=='Gaming'].describe()
```

Out[37]:

	Inches	Ram(GB)	Weight(Kg)	Price_euros	MemorySSD(GB
count	205.000000	205.000000	205.000000	205.000000	205.00000
mean	16.345854	14.048780	2.949761	1731.380634	236.01951
std	0.927174	7.234013	0.759205	814.174430	189.47460
min	14.000000	4.000000	1.600000	699.000000	0.00000
25%	15.600000	8.000000	2.430000	1169.000000	128.00000
50%	15.600000	16.000000	2.700000	1492.800000	256.00000
75%	17.300000	16.000000	3.350000	2199.000000	256.00000
max	18.400000	64.000000	4.700000	6099.000000	1024.00000

Se aprecia que aunque encontramos también algún ordenador barato de estas características (699), la mayoría, incrementan todos los percentiles vistos anteriormente. Además, con el resto de característimas de rendimiento como la CPU, presentan valores altos en donde desde el percentil 50 toma el valor de 2.8, llegando a tomar 3.2 en su máximo.

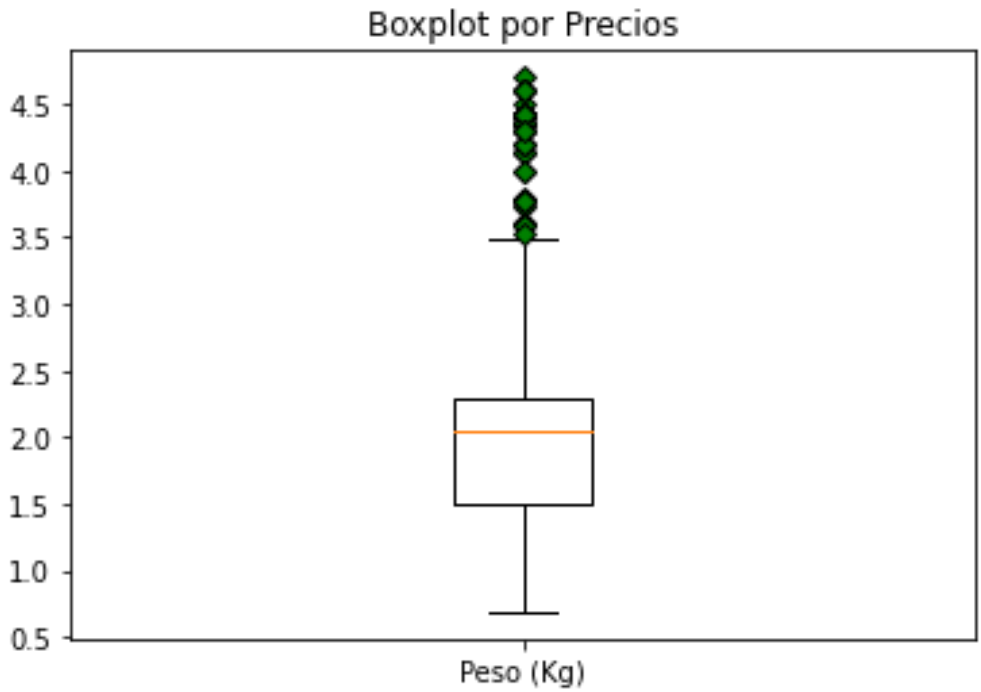
Centrando ahora el estudio para el caso del peso:

In [38]:

```
# Construción de boxplot
green_diamond = dict(markerfacecolor='g', marker='D')
fig, ax = plt.subplots()
ax.set_title('Boxplot por Precios')
ax.boxplot(laptops_initial['Weight(Kg)'], flierprops=green_diamond, lab
els=["Peso (Kg)"])
```

Out[38]:

```
{'boxes': [<matplotlib.lines.Line2D at 0x7fbc7bdd8610>],
 'caps': [<matplotlib.lines.Line2D at 0x7fbc7be726d0>,
  <matplotlib.lines.Line2D at 0x7fbc7be81150>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fbc7bde8350>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x7fbc7b939150>],
 'whiskers': [<matplotlib.lines.Line2D at 0x7fbc7fe18450>,
  <matplotlib.lines.Line2D at 0x7fbc7be72a90>]}
```



Se aprecia como se vuelven a tener varios registros que se alejan "por arriba" del conjunto de los demás datos. Para ello, volveremos a ver de que ordenadores se trata:

In [39]:

```
laptops_initial[laptops_initial['Weight(Kg)']>3.5]
```

Out[39]:

	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Pri
177	MSI	Gaming	18.4	32	Windows 10	4.40	
224	Dell	Gaming	17.3	16	Windows 10	4.42	
238	Asus	Gaming	17.3	32	Windows 10	4.70	
247	Asus	Gaming	17.3	16	Windows 10	3.60	
251	Asus	Gaming	17.3	16	Windows 10	4.30	
258	MSI	Gaming	17.3	16	Windows 10	4.14	
424	Dell	Gaming	17.3	16	Windows 10	4.42	
456	Dell	Notebook	17.3	8	Windows 10	4.42	
468	HP	Gaming	17.3	12	Windows 10	3.78	
494	Asus	Gaming	17.3	24	Windows 10	4.33	
530	Dell	Gaming	17.3	16	Windows 10	4.42	
552	Dell	Gaming	17.3	16	Windows 10	4.42	
577	Lenovo	Gaming	17.3	32	Windows 10	4.60	
578	MSI	Gaming	17.3	16	Windows 10	4.14	
585	MSI	Gaming	17.3	16	Windows 10	4.50	
586	HP	Gaming	17.3	16	Windows 10	3.78	
603	MSI	Gaming	17.3	16	Windows 10	4.14	
650	Asus	Gaming	17.3	32	Windows 10	3.80	

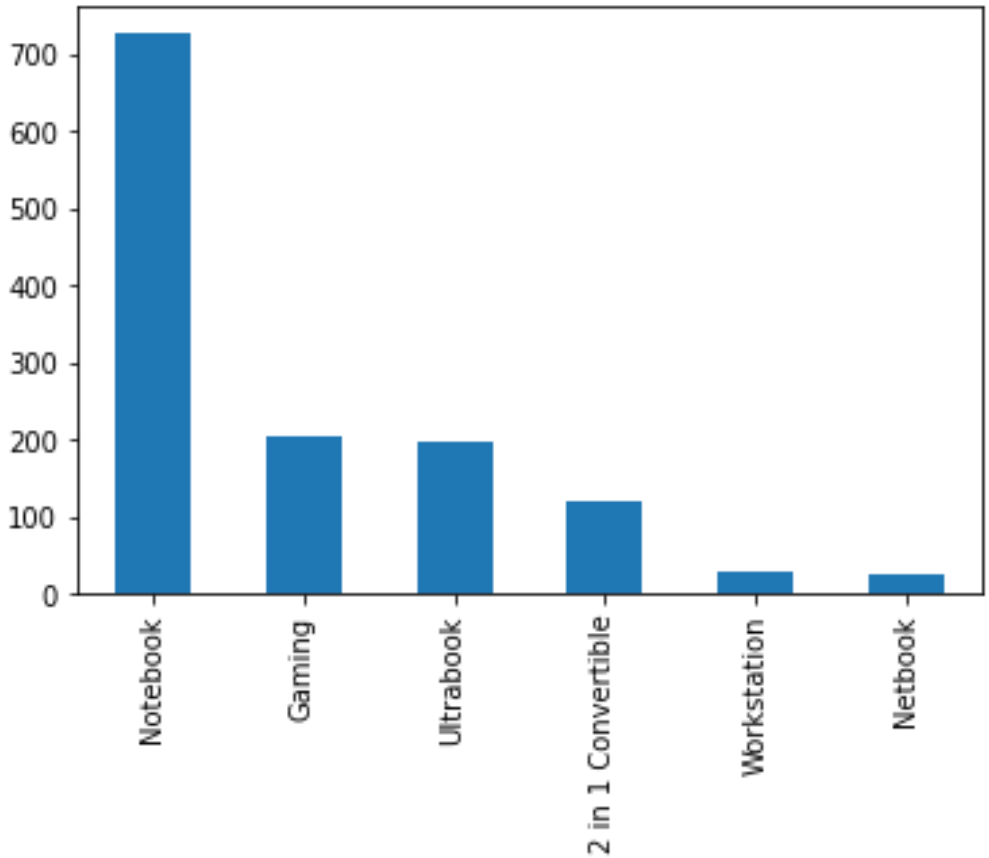
	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Pri
659	Dell	Gaming	17.3	32	Windows 10	4.42	
723	Dell	Gaming	17.3	32	Windows 10	4.36	
730	Acer	Gaming	17.3	16	Windows 10	4.20	
758	Dell	Gaming	15.6	16	Windows 10	4.42	
780	Dell	Gaming	17.3	32	Windows 10	4.42	
781	Asus	Gaming	17.3	16	Windows 10	4.30	
788	Acer	Gaming	17.3	16	Windows 10	4.20	
810	HP	Gaming	17.3	12	Windows 10	3.74	
818	Dell	Gaming	17.3	16	Windows 10	4.36	
841	Dell	Gaming	17.3	32	Windows 10	4.42	
901	Asus	Gaming	17.3	32	Windows 10	3.58	
939	Dell	Gaming	17.3	16	Windows 10	4.36	
955	Dell	Gaming	17.3	16	Windows 10	4.36	
968	Dell	Gaming	17.3	32	Windows 10	4.42	
972	Dell	Gaming	17.3	32	Windows 10	4.42	
1047	MSI	Gaming	17.3	16	Windows 10	3.78	
1048	Lenovo	Gaming	17.3	16	Windows 10	4.60	
1061	Asus	Gaming	17.3	16	No OS	4.00	
1066	Asus	Gaming	17.3	64	Windows 10	3.58	

	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Pri
1077	MSI	Gaming	17.3	16	Windows 10	3.78	
1081	Lenovo	Gaming	17.3	32	Windows 10	4.60	
1099	Asus	Gaming	17.3	16	Windows 10	4.30	
1116	Lenovo	Gaming	17.3	16	Windows 10	4.60	
1119	Asus	Gaming	17.3	8	Windows 10	3.52	
1138	MSI	Gaming	17.3	16	Windows 10	3.78	
1189	Acer	Gaming	17.3	16	Windows 10	4.20	
1197	Asus	Gaming	17.3	16	Windows 10	4.30	
1256	Asus	Gaming	17.3	16	Windows 10	4.00	

Se aprecia como, de nuevo, vuelve a tratarse de ordenadores Gaming, los cuales parece ser que pesan más que los "normales", posiblemente, debido a que presentan componentes más pesados y pesados, siendo ordenadores centrados en el rendimiento y no tanto en la portabilidad. Para comprobar cómo estos ordenadores no son mayoría, se realizará un gráfico de barras en función del tipo de ordeandor:

In [40]:

```
laptops_initial['TypeName'].value_counts().plot.bar()
plt.show()
```



Se aprecia como el tipo Notebook es el más presente en el dataSet, y por tanto, el que concetrará más ordeandores de las mismas características. El resto de ordenadores, aunque tienen diferentes funcionalidades, tienen un uso parecido entre ellos (notebook, ultrabook o 2 en 1), en donde sus diferencias en términos de características son pequeñas (que la pantalla sea táctil, que sea mas transportable...). Sin embargo, los gaming, suelen ser ordenadores más potentes, y habitualmente, más grandes, alejándose del resto de los ordenadores que hacen que se presente esta diferencia, y no por ello son datos inválidos. Serán simplemente, como el nombre del apartado indica, valores extremos.

Para el resto de variables numéricas, no tiene tanto sentido elaborar este estudio, ya que aunque son valores numéricos, son valores que no son continuos como tal, ya que las medidas dadas en GB son medidas específicas (8-16-64...), al igual que ocurre con los tamaños de pantalla, en donde suelen ser tamaños estandarizados (alto-ancho) y no llegan a ser continuos como tal. Además, estos valores se han visto en la preparación de datos anteriores, y no ha apreciado ningún valor fuera de lo común o medida desmesurada para un ordenador.

Una vez tenemos los datos limpios, los exportamos para añadirlos al git para por si en futuro alguien desea reutilizarlos:

In [41]:

```
#Se exportan los datos limpios
laptops_initial.to_csv('laptops_procesados.csv')
```

4. Análisis de los datos

La variable objetivo de este conjunto de datos es la variable de Price_euros, debido a que en apartados posteriores, realizaremos un contraste de medias y un modelo de regresión lineal.

En el contraste de medias, estudiaremos si se puede afirmar que es estadísticamente significativo que los productos de Apple tengan un coste más elevado que el resto de marcas.

En la regresión lineal, se realizará un modelo que trate de predecir el precio de un dispositivo teniendo en cuenta sus características técnicas. Para ello, se hará uso de las variables independientes que muestren tener más impacto en el cálculo final del producto, realizando un análisis de sus distribuciones y correlación.

Resumen del dataSet

En este apartado se realizará un resumen del dataSet con el que se va a trabajar, para que de un vistazo, se pueda extraer información útil del mismo.

In [42]:

```
#Se realiza un pequeño resumen estadístico de las variables numéricas
 laptops_initial.describe()
```

Out[42]:

	Inches	Ram(GB)	Weight(Kg)	Price_euros	MemorySSD
count	1303.000000	1303.000000	1303.000000	1303.000000	1303.00
mean	15.017191	8.382195	2.038734	1123.686992	184.02
std	1.426304	5.084665	0.665475	699.009043	188.26
min	10.100000	2.000000	0.690000	174.000000	0.00
25%	14.000000	4.000000	1.500000	599.000000	0.00
50%	15.600000	8.000000	2.040000	977.000000	256.00
75%	15.600000	8.000000	2.300000	1487.880000	256.00
max	18.400000	64.000000	4.700000	6099.000000	1024.00

Con lo que se ha visto hasta el momento, de esta tabla resumen lo primero que llama la atención es que se puede confirmar la afirmación mostrada anteriormente. Todos los valores presentan medias y percentiles menores que los que se han dado en los ordenadores gaming, menos en la memoria Flash, ya que los ordenadores de estas características priorizan los otros tipos de memoria, en especial la SSD. Por tanto, se aprecia como lo descrito anteriormente es cierto.

Por otro lado, si ignoramos las variables de memoria, no se producen desviaciones estándar muy elevadas, excepto en el precio (que tiene sentido al tener ordenadores de diversas marcas, características) y en la resolución de pantalla, que tiene sentido también ya que son números más elevados y es lógico que de un notebook a un ordenador gamer se encuentre esta diferencia en la pantalla.

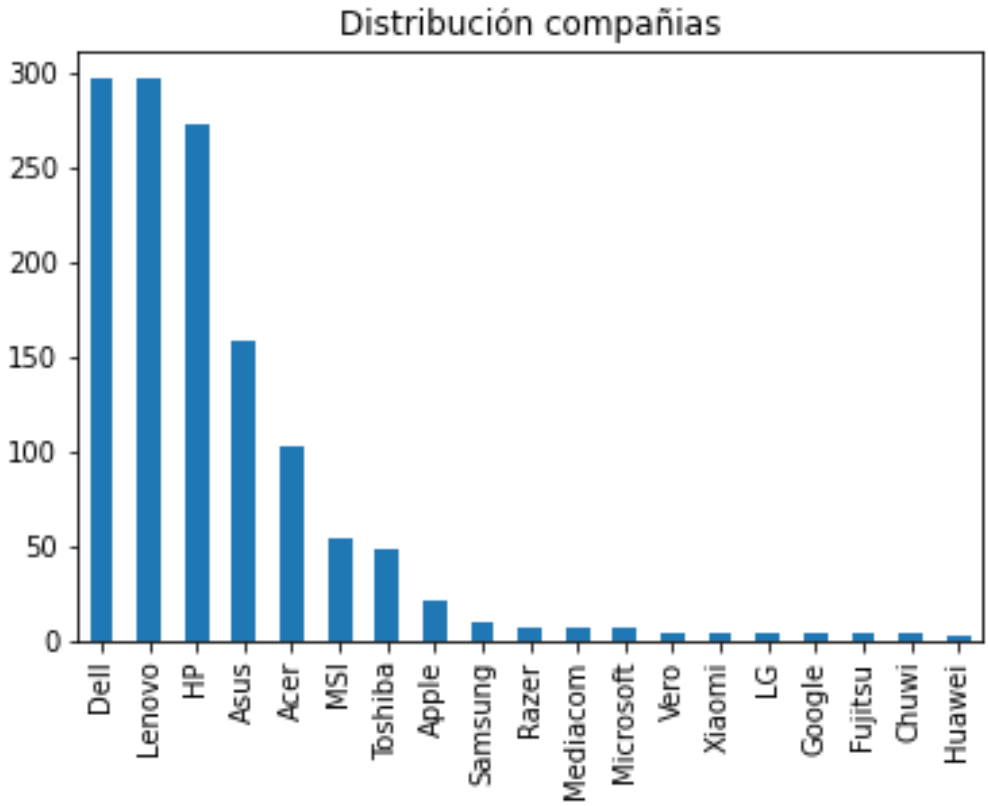
En las memorias vemos como las desviaciones son mayores (teniendo en cuenta su unidad de medida, medias y límites), y esto es así por el hecho de que algunos ordenadores poseen un tipo de memoria y otros dos. Este hecho, por la decisión que se ha tomado antes, hace que si no se dispone de un tipo de memoria, se ponga el valor de 0 para ese caso, por lo que en una columna se pueden apreciar datos algo diversos por el hecho de que un ordenador tenga una memoria de un tipo pero de otra no. Debido a este hecho, y a que estas columnas se podrían llegar a considerar discretas (las memorias pueden ser de unos valores en concreto), se harán una representación gráfica de las mismas como con las categóricas.

Para las variables categóricas, siempre que no nos encontremos con una cantidad de posibles valores desorbitada, se realizarán gráficos de barra que permiten ver de una forma más visual el tipo de datos con los que se trabaja. En caso de que se encuentren muchas más categorías, se realizará una tabla de frecuencia relativa a esa variable, hecho que ocurre tanto para las versiones de GPU como de CPU como para los tipo de resolución .

Por tanto, calculando las mismas:

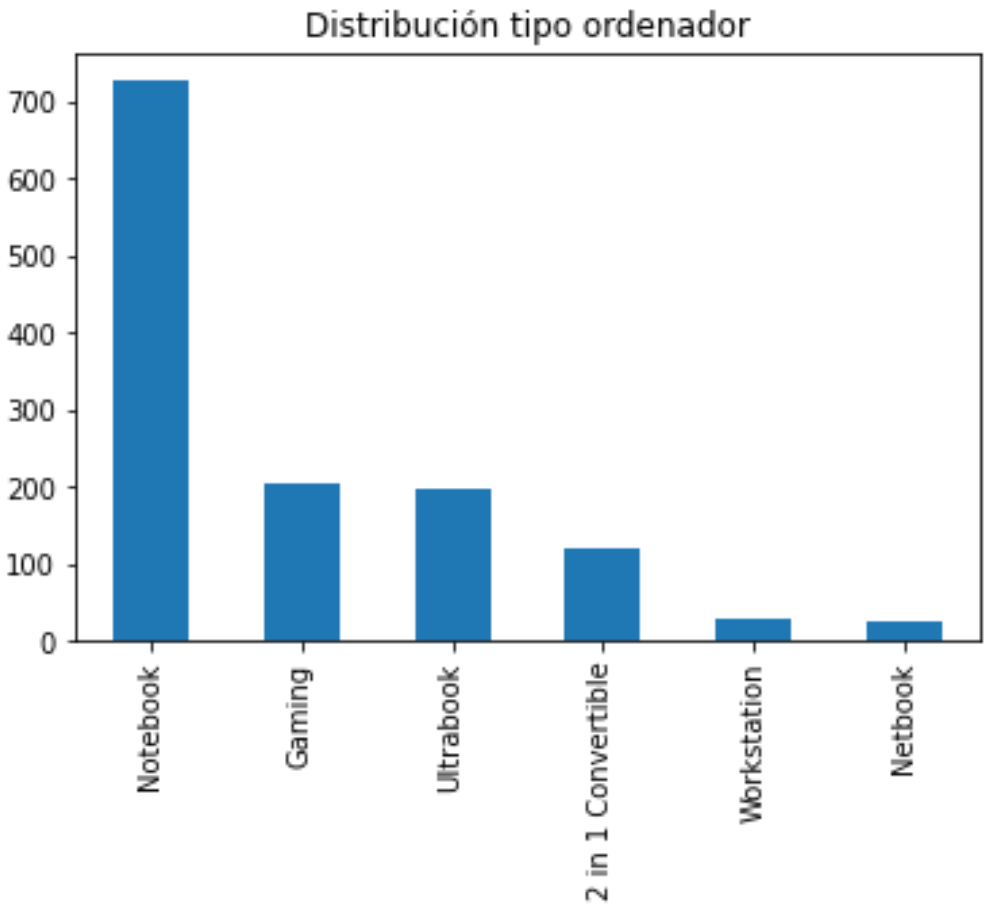
In [43]:

```
laptops_initial['Company'].value_counts().plot.bar()
plt.title("Distribución compañias")
plt.show()
```



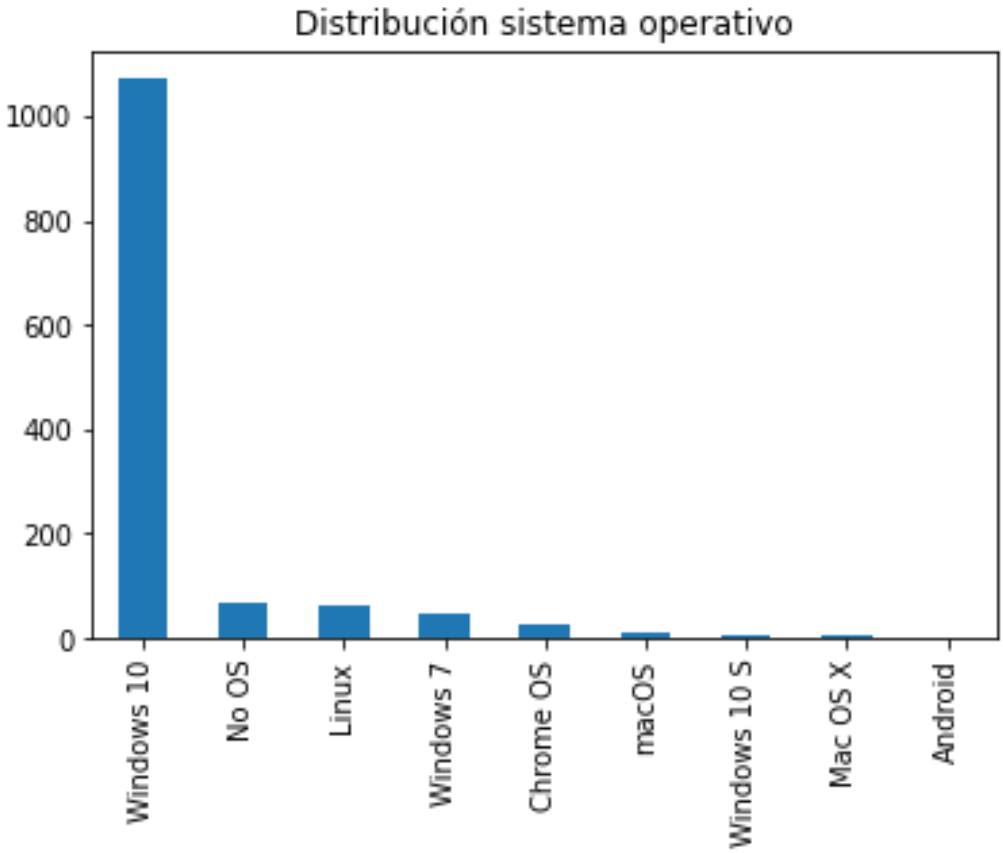
In [44]:

```
laptops_initial['TypeName'].value_counts().plot.bar()
plt.title("Distribución tipo ordenador")
plt.show()
```



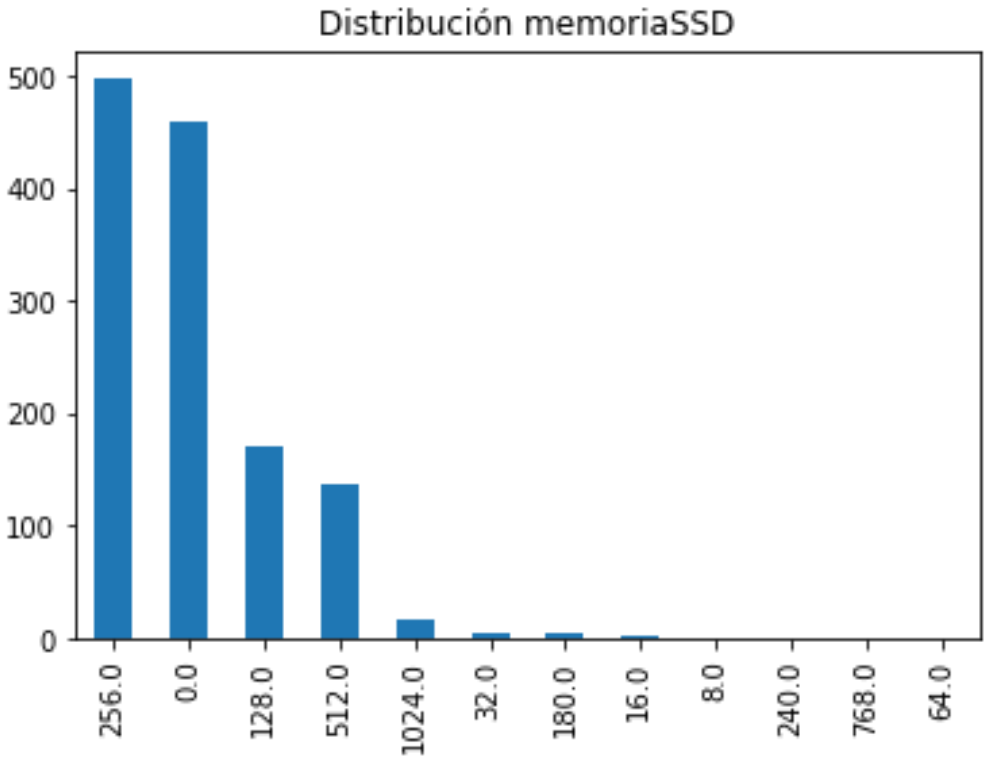
In [45]:

```
laptops_initial['OpSys'].value_counts().plot.bar()
plt.title("Distribución sistema operativo")
plt.show()
```



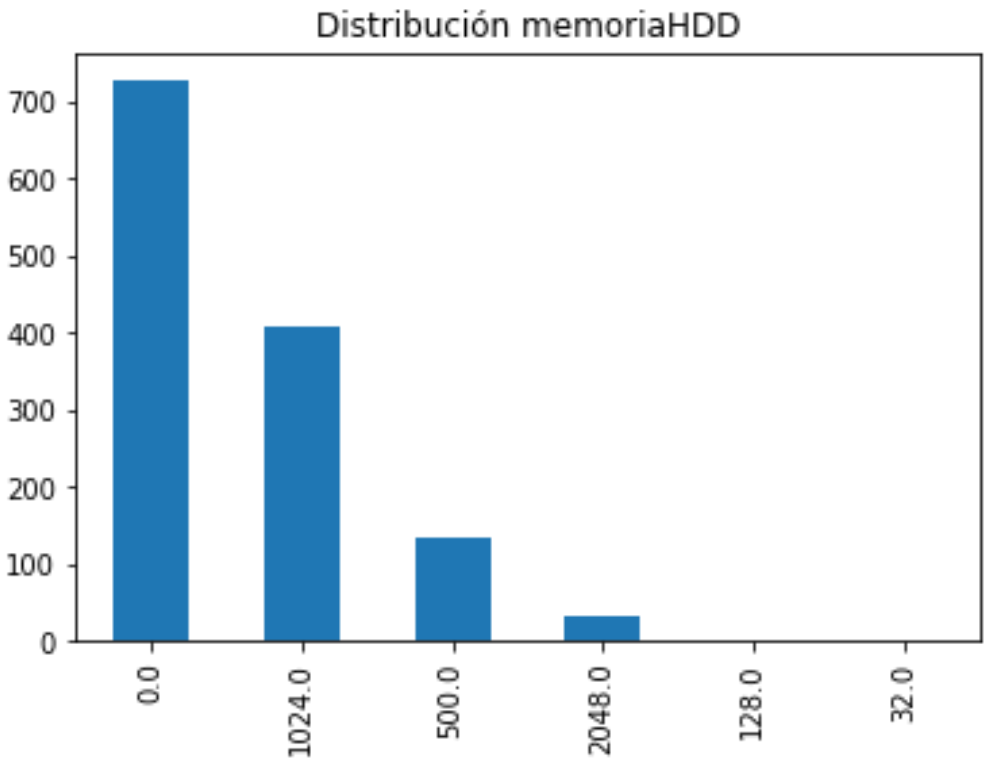
In [46]:

```
laptops_initial['MemorySSD(GB)'].value_counts().plot.bar()
plt.title("Distribución memoriaSSD")
plt.show()
```



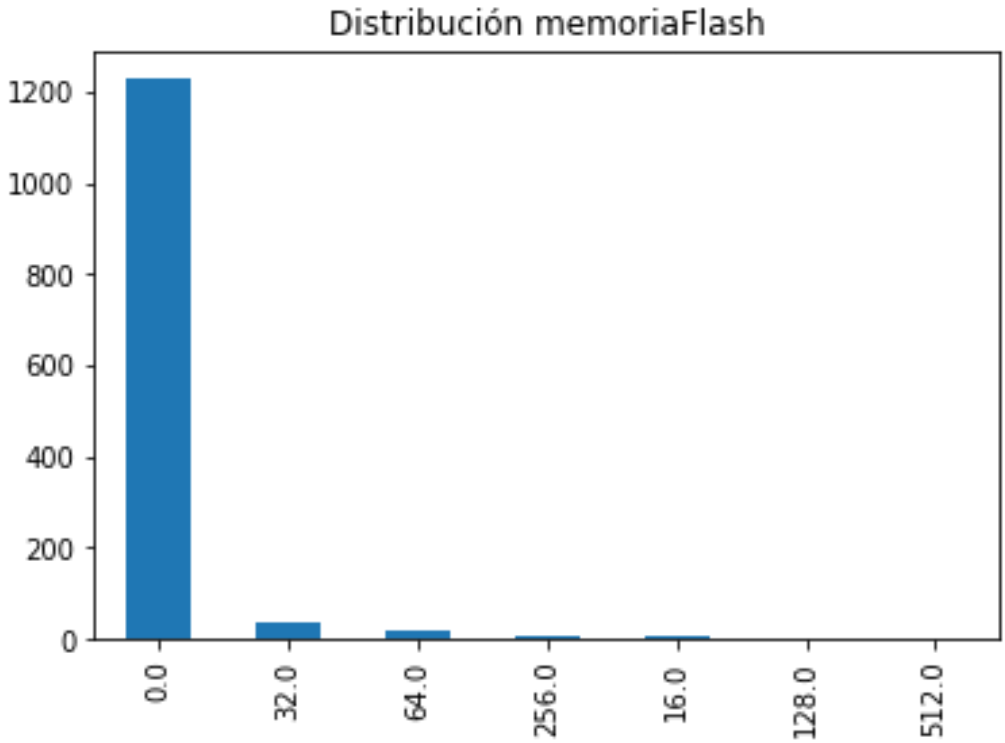
In [47]:

```
laptops_initial['MemoryHDD(GB)'].value_counts().plot.bar()
plt.title("Distribución memoriaHDD")
plt.show()
```



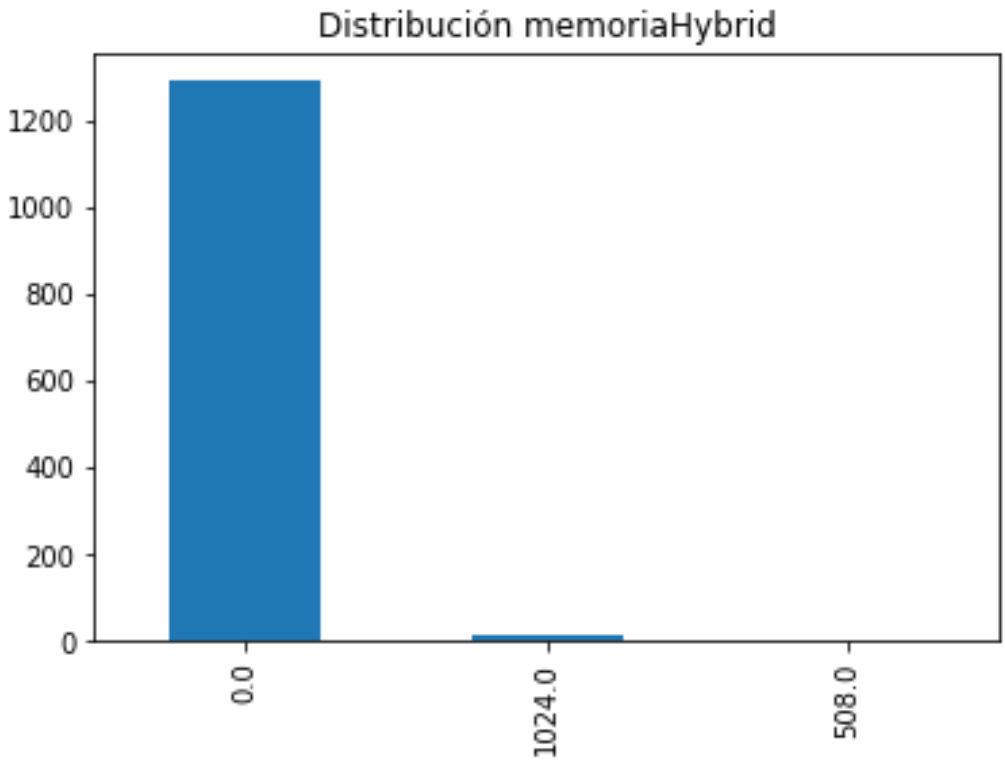
In [48]:

```
laptops_initial['MemoryFlash(GB)'].value_counts().plot.bar()
plt.title("Distribución memoriaFlash")
plt.show()
```



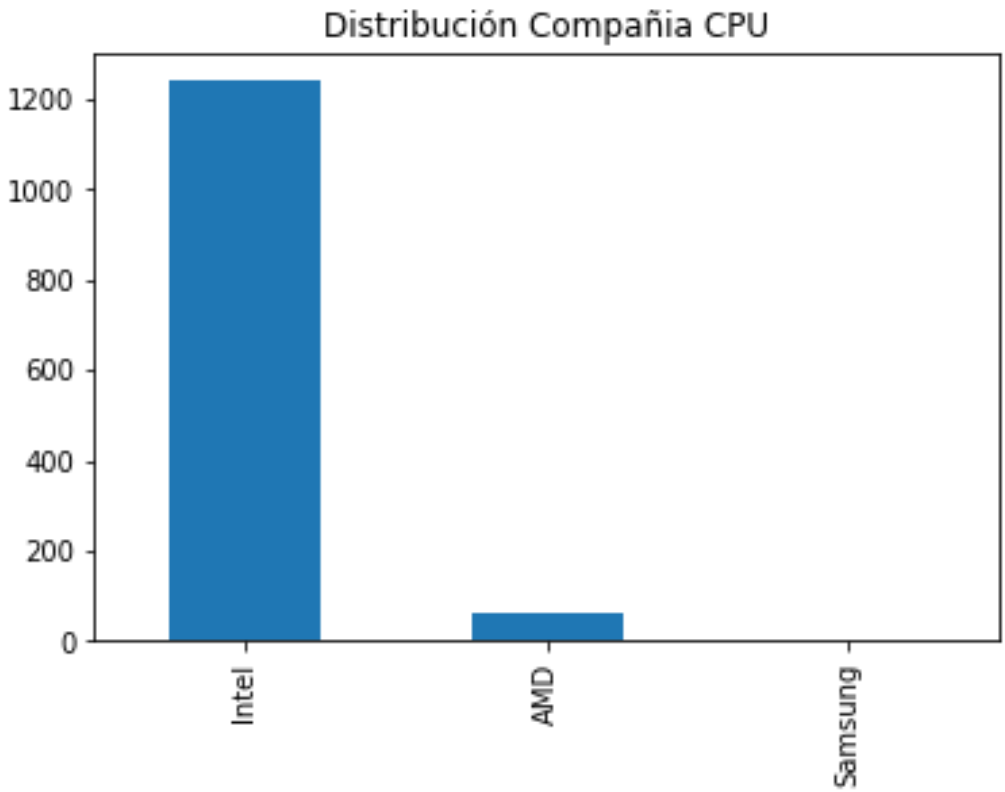
In [49]:

```
laptops_initial['MemoryHybrid(GB)'].value_counts().plot.bar()
plt.title("Distribución memoriaHybrid")
plt.show()
```



In [50]:

```
laptops_initial['CPU_Company'].value_counts().plot.bar()
plt.title("Distribución Compañia CPU")
plt.show()
```



In [51]:

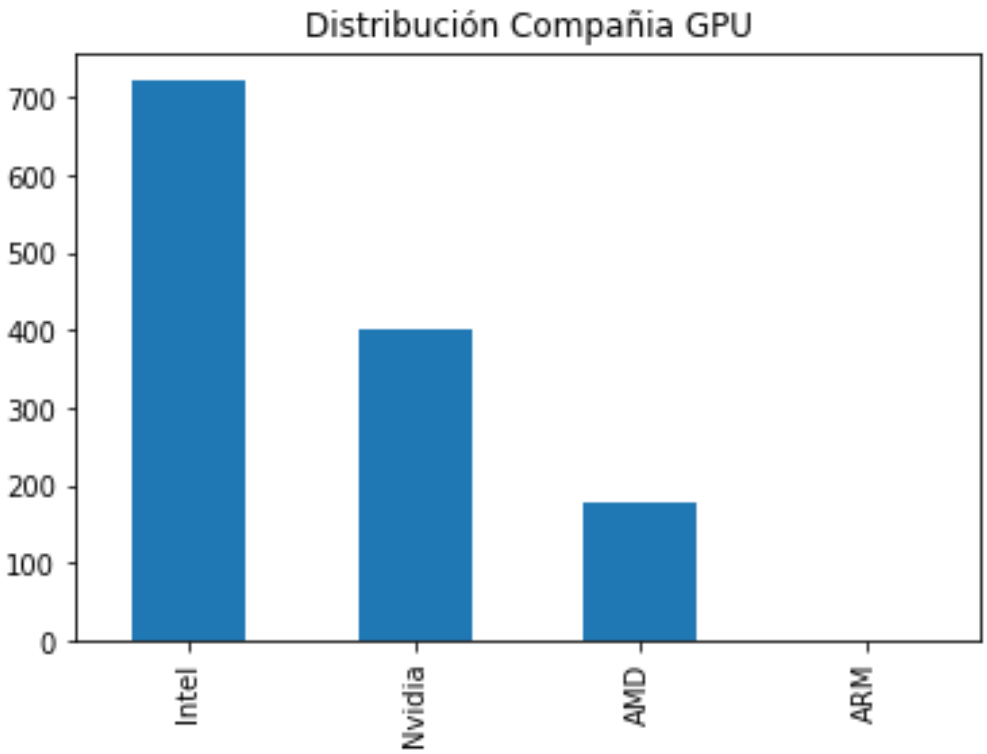
```
#laptops_initial['CPU_Version'].value_counts().plot.bar()
#plt.title("Distribución version CPU")
#plt.show()
# tabla de frecuencia
100 * laptops_initial['CPU_Version'].value_counts() / len(laptops_initi
al['CPU_Version'])
```

Out[51]:

```
Core i5 7200U               14.811972
Core i7 7700HQ              11.281658
Core i7 7500U               10.360706
Core i3 6006U                6.216424
Core i7 8550U                5.602456
                                ...
Atom Z8350                   0.076746
Core i7 6920HQ               0.076746
Celeron Quad Core N3710      0.076746
Pentium Dual Core 4405U      0.076746
Pentium Dual Core N4200      0.076746
Name: CPU_Version, Length: 93, dtype: float64
```

In [52]:

```
laptops_initial['GPU_Company'].value_counts().plot.bar()
plt.title("Distribución Compañia GPU")
plt.show()
```



In [53]:

```
#laptops_initial['GPU_Version'].value_counts().plot.bar()
#plt.title("Distribución version CPU")
#plt.show()
# tabla de frecuencia
100 * laptops_initial['GPU_Version'].value_counts() / len(laptops_initi
al['GPU_Version'])
```

Out[53]:

```
HD Graphics 620       21.565618
HD Graphics 520       14.198005
UHD Graphics 620       5.218726
GeForce GTX 1050       5.065234
GeForce GTX 1060       3.683807
                        ...
GeForce 960M           0.076746
GeForce GTX1050 Ti     0.076746
Radeon Pro 560         0.076746
Mali T860 MP4          0.076746
Radeon R7 M465         0.076746
Name: GPU_Version, Length: 110, dtype: float64
```

In [54]:

```
#laptops_initial['ScreenResolution_Type'].value_counts().plot.bar()
#plt.title("Distribución tipo de resolución")
#plt.show()
# tabla de frecuencia
100 * laptops_initial['ScreenResolution_Type'].value_counts() / len(lap
tops_initial['ScreenResolution_Type'])
```

Out[54]:

```
Full HD                                38.910207
Unknown                                24.098235
IPS Panel Full HD                      18.035303
IPS Panel Full HD / Touchscreen         4.067536
Full HD / Touchscreen                   3.607061
Touchscreen                             2.455871
IPS Panel Retina Display                1.304682
Quad HD+ / Touchscreen                  1.151190
IPS Panel Touchscreen                   0.997698
IPS Panel 4K Ultra HD                   0.920952
IPS Panel 4K Ultra HD / Touchscreen     0.844206
IPS Panel                               0.844206
4K Ultra HD / Touchscreen               0.767460
4K Ultra HD                             0.537222
IPS Panel Quad HD+ / Touchscreen        0.460476
IPS Panel Quad HD+                      0.383730
Quad HD+                                0.230238
IPS Panel Touchscreen / 4K Ultra HD     0.153492
Touchscreen / Full HD                   0.076746
Touchscreen / 4K Ultra HD               0.076746
Touchscreen / Quad HD+                  0.076746
Name: ScreenResolution_Type, dtype: float64
```

De las gráficas de barras se pueden extraer de primera mano varias conclusiones, como son:

- El dataset cuenta con gran cantidad de marcas de ordenadores diferentes, pero se observa una predominancia de Dell, Lenovo y HP. Posiblemente sea porque son los que más modelos tengan.
- Los ordenadores de tipo Notebok (al ser los más comunes en el día a día), son los que más porción ocupan del dataSet.
- El sistema operativo por excelencia es Windows 10, seguido muy de lejos por ordenadores libres y a continuación Linux.
- En el caso de contar memoria SSD (que más de la mitad cuentan con este tipo de memoria), la misma es de 256 GB. En el caso de tener HDD, es 1024GB, aunque predominan los ordenadores sin este tipo de memoria (habitualmente es porque los ordenadores, como se ha podido ver en la limpieza, es más habitualmente que posean memoria SSD que HDD al ser esta más rápida). Para los casos de flash e Hybrid, no la gran mayoría no disponen de esta memoria.
- Se puede afirmar, que Intel es la marca líder en el mercado en “darle vida al ordanador”. Esta marca es la líder en tanto en GPU como en CPU. En GPU, es seguida por Nvidia, sistema operativo presente en la gran mayoría de ordenadores Gaming como se puede apreciar en la tabla del apartado anterior.

Respecto a estas tablas de frecuencias, se aprecia como ha gran cantidad de versiones tanto de CPU como de GPU. Estos son principalmente el core i5 7200U, que se lleva casi el 15% de ordenadores (este hecho puede ser debido a que es una versión de CPU de calidad baja válida para trabajos diarios), seguidos por CPU de calidad algo mayores al hablar de core i7. Para el cado de las GPU, predominan los HD Graphics 620 y 520, llevándose estas el 35% de los ordenadores. Finalmente, con respecto al tipo de resolución predomina la HD en su mayor parte, aunque en un 25% de las ocasiones, no se ha indicado el tipo de la misma como se ha analizado en el apartado de limpieza.

Contraste de medias

Tal y como se ha indicado anteriormente, procedemos en este apartado a realizar un constraste de medias que nos indicará si los productos de Apple son significativamente más caros, de media, que el resto de productos de otras marcas.

Para ello, separaremos primero el dataset en dos de tal forma que contengan respectivamente los precios de los productos de Apple y del resto de marcas.

In [55]:

```
apple_prices = laptops_initial[laptops_initial["Company"] == "Apple"][
"Price_euros"]
other_prices = laptops_initial[laptops_initial["Company"] != "Apple"][
"Price_euros"]
print(apple_prices.head(5))
print(other_prices.head(5))

print(f'Precio medio de los productos de Apple: {round(st.mean(apple_pr
ices), 2)}.')
print(f'Precio medio de los productos de otras marcas: {round(st.mean(o
ther_prices), 2)}.')
```

```
0    1339.69
1     898.94
3    2537.45
4    1803.60
6    2139.97
Name: Price_euros, dtype: float64
2      575.0
5      400.0
8     1495.0
9      770.0
10     393.9
Name: Price_euros, dtype: float64
Precio medio de los productos de Apple: 1564.2.
Precio medio de los productos de otras marcas: 1116.47.
```

Una vez separadas las muestras, estudiamos la normalidad de cada una de ellas:

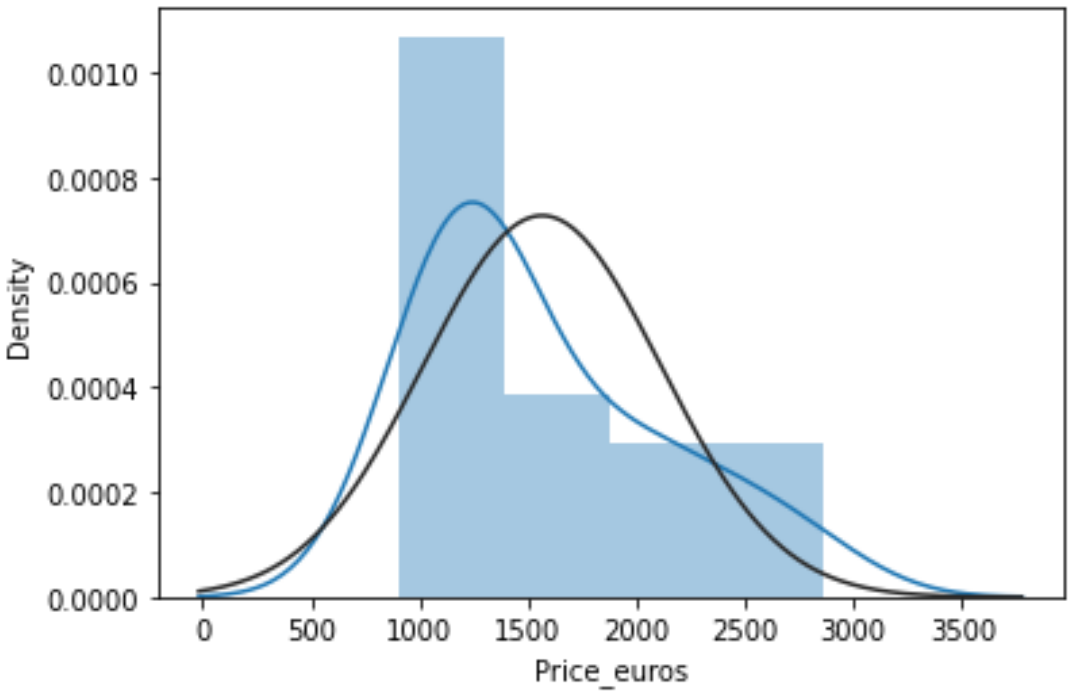
In [56]:

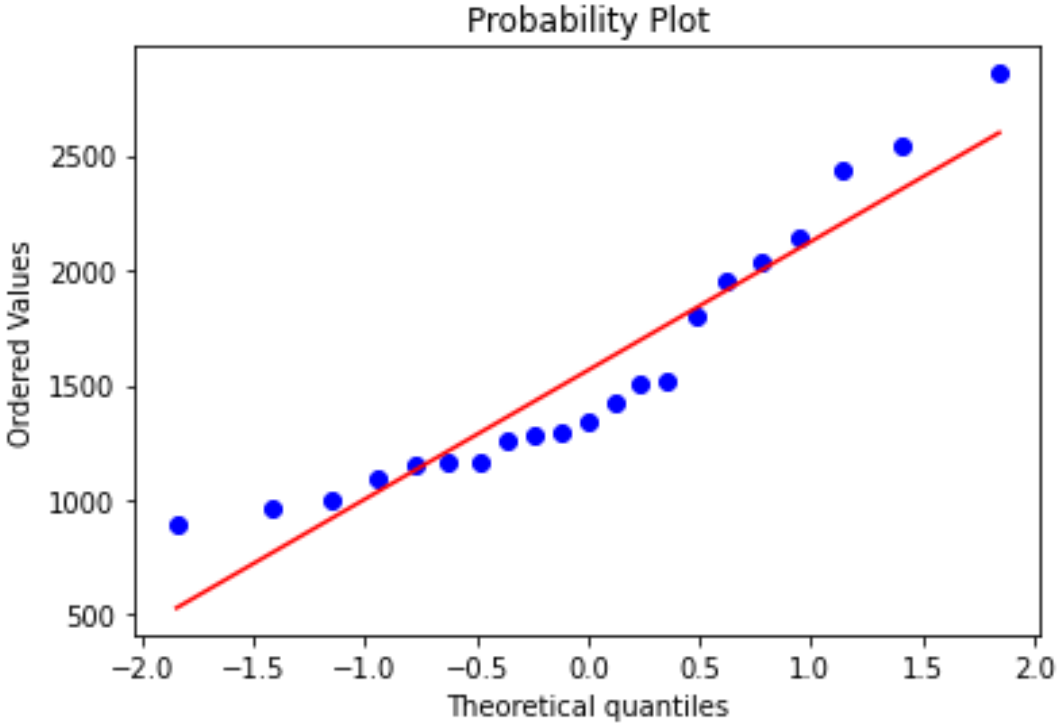
```
# Histograma y gráfico de probabilidad normal de los precios de Apple:
sns.distplot(apple_prices, fit = norm);
fig = plt.figure()
res = stats.probplot(apple_prices, plot = plt)

# Prueba de Shapiro-Wilk
stat, p = shapiro(apple_prices)
print(f"Stat: {round(stat,3)}")
print(f"p-value: {round(p,3)}")
```

```
Stat: 0.893
p-value: 0.026

/usr/local/lib/python3.7/dist-packages/seaborn/distributio
ns.py:2557: FutureWarning: `distplot` is a deprecated func
tion and will be removed in a future version. Please adapt
your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```




Probability Plot
Ordered Values
2500
2000
1500
1000
500
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Theoretical quantiles

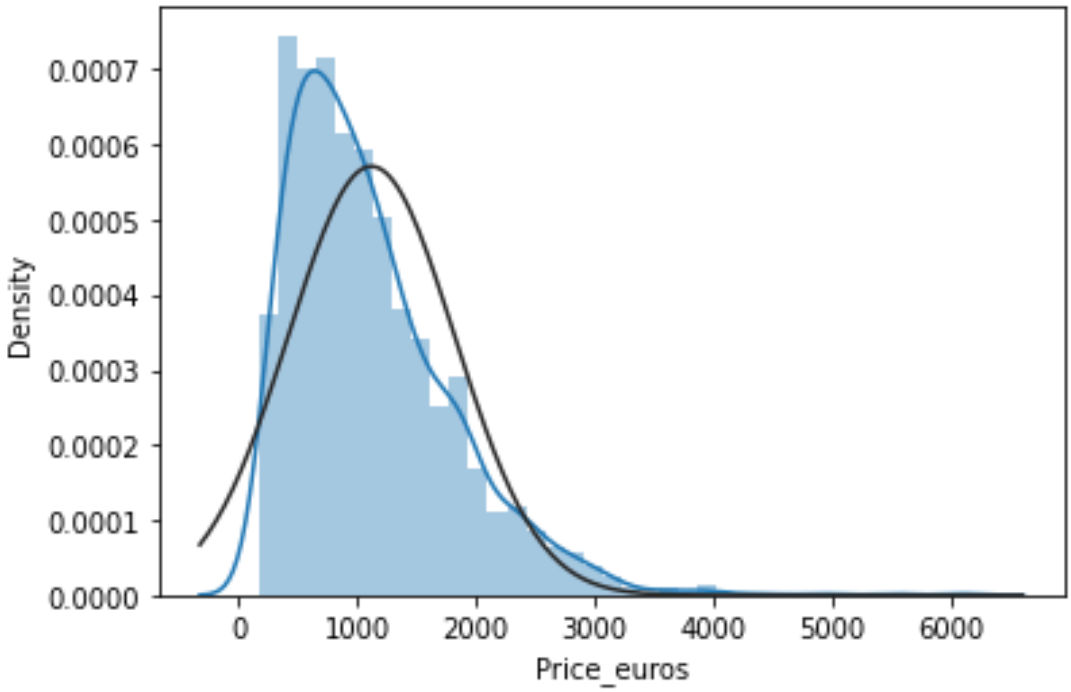
In [57]:

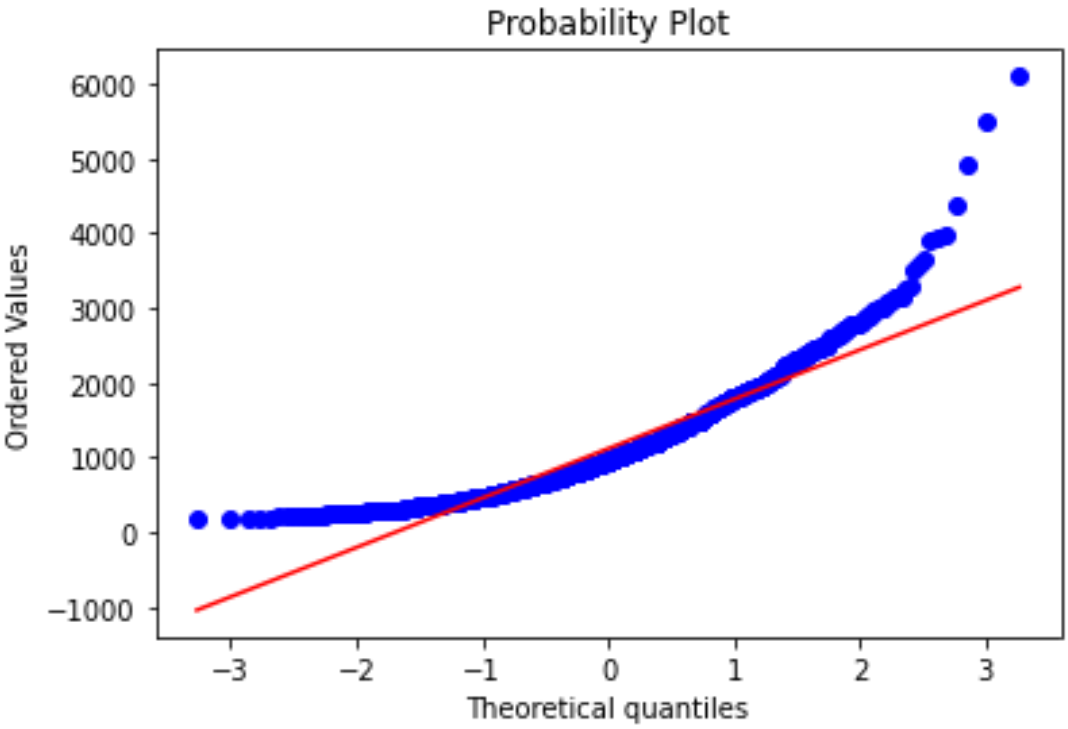
```
# Histograma y gráfico de probabilidad normal de los precios del resto
 de marcas:
sns.distplot(other_prices, fit = norm);
fig = plt.figure()
res = stats.probplot(other_prices, plot = plt)

# Prueba de Shapiro-Wilk
stat, p = shapiro(other_prices)
print(f"Stat: {round(stat,3)}")
print(f"p-value: {round(p,3)}")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributio
ns.py:2557: FutureWarning: `distplot` is a deprecated func
tion and will be removed in a future version. Please adapt
your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)

Stat: 0.891
p-value: 0.0
```





En base a los resultados obtenidos, no es posible garantizar la normalidad en la distribución de las muestras de precios tanto de apple como de otras marcas dado que el test de Shapiro rechaza la hipótesis nula de normalidad. Asimismo, en el caso del subset de Apple, no sería posible tampoco aplicar el teorema del límite central al no superar las 30 muestras.

En este sentido, para poder confirmar si podemos asumir homocedasticidad (igualdad de varianzas entre muestras) debemos aplicar el test de Fligner-Killeen (no paramétrico) al no poder suponer normalidad:

In [58]:

```
# Fligner-Killeen test
fligner_test = stats.fligner(apple_prices, other_prices, center='media
n')
fligner_test
```

Out[58]:

```
FlignerResult(statistic=0.9821213321141975, pvalue=0.32167
564402079707)
```

A raíz de los resultados (p value >> 0.05) no podemos descartar la hipótesis nula, por lo que se confirma la homocedasticidad. No obstante, debido a que no hemos podido afirmar que sigan distribuciones normales, no podremos aplicar un contraste de muestras paramétrico (t-Student), si no que tendremos que aplicar uno no paramétrico (Mann-Whitney):

In [59]:

```
# Mann-Whitney test
mannwhitneyu_test = stats.mannwhitneyu(apple_prices, other_prices, alte
rnative="greater")
mannwhitneyu_test
```

Out[59]:

```
MannwhitneyuResult(statistic=19689.0, pvalue=0.00013581790
526573893)
```

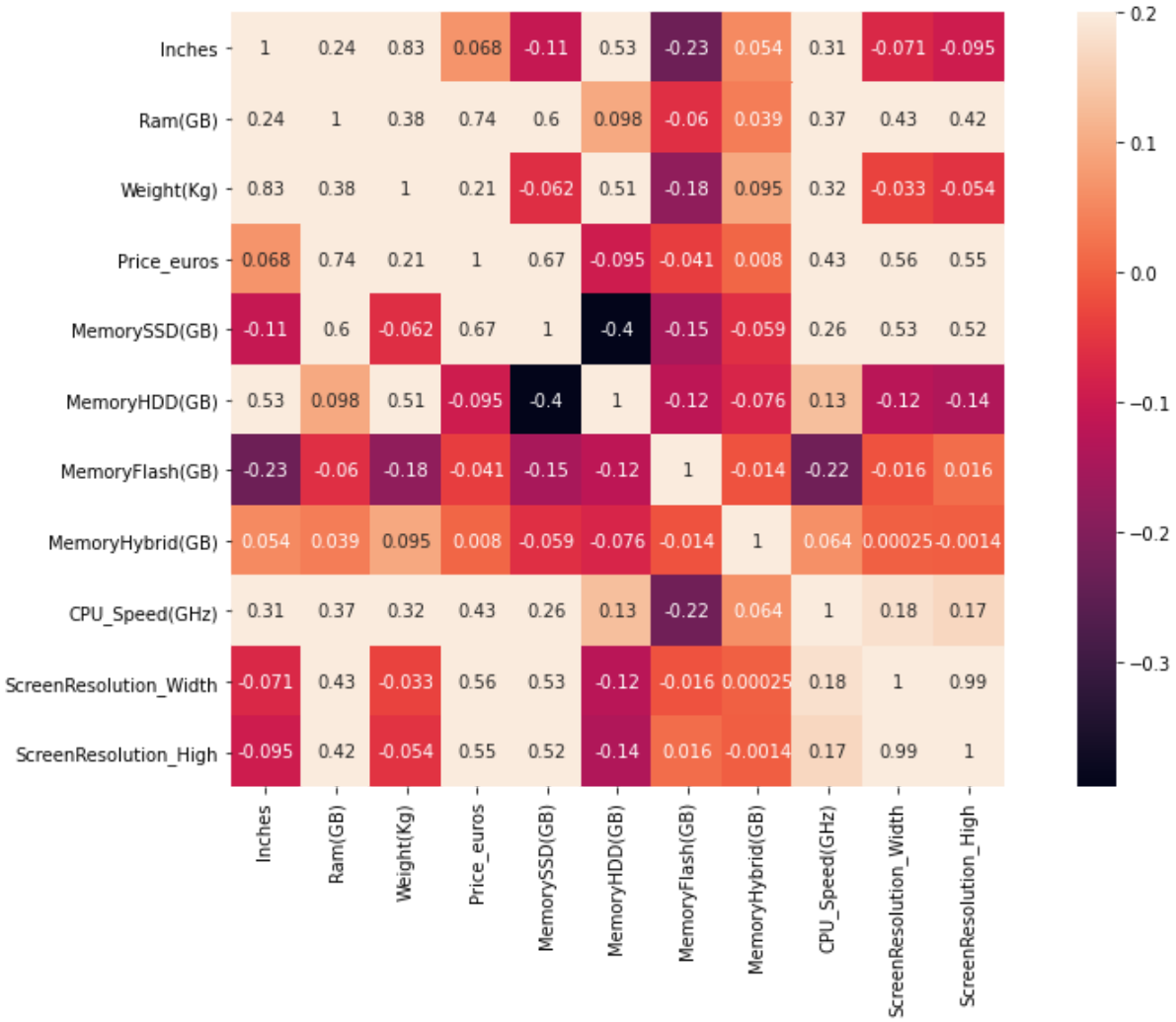
En base a los resultados obtenidos del test de Mann-Whitney (p value << 0.05), podemos rechazar la hipótesis nula en favor de la hipótesis alternativa que, en este caso, correspondía con que el precio de los productos de Apple es superior de media que para el resto de marcas.

Estudio de variables numéricas

Realizamos primero una matriz de correlación, que nos podrá indicar con qué variables tiene más relación el valor del precio:

In [60]:

```
# Matriz de correlación:
corrmat = laptops_initial.corr()
f, ax = plt.subplots(figsize=(15, 8))
sns.heatmap(corrmat, annot=True, vmax=.2, square=True);
```



Tal y como se puede apreciar en el mapa de calor, la variable numérica con la que tiene más correlación el precio es con la cantidad de RAM que tenga el dispositivo, seguido de la cantidad de memoria SSD, la resolución de pantalla y la CPU que monte el dispositivo. Todas ellas, serán candidatas a formar parte del modelo de predicción del precio de un dispositivo.

Al contrario, las variables que menos influyen en la variabilidad de precio son el resto de tipos de almacenamiento (HDD, Flash e Híbrido), así como las pulgadas del portátil. El peso, pese a que influye en la valoración del portátil, lo hace de forma muy débil.

Dado que se identifica que la resolución de pantalla a lo ancho y alto están completamente correladas entre sí, se elimina la columna ScreenResolution_High para evitar redundancias.

In [61]:

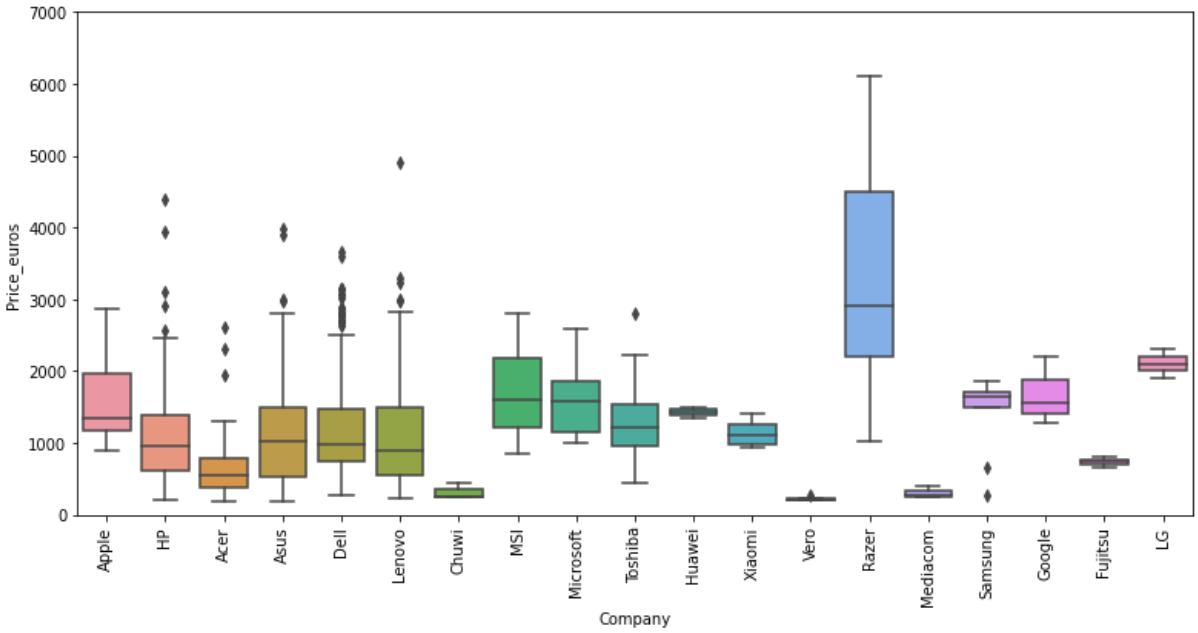
```
# Eliminación de la columna redundante ScreenResolution_High
del laptops_initial['ScreenResolution_High']
```

Estudio de variables categóricas

A continuación, vamos a estudiar con más detalle algunas de las variables categóricas que pensamos pueden afectar más al rendimiento del modelo, para identificar visualmente si vemos conveniente que también formen parte del mismo:

In [62]:

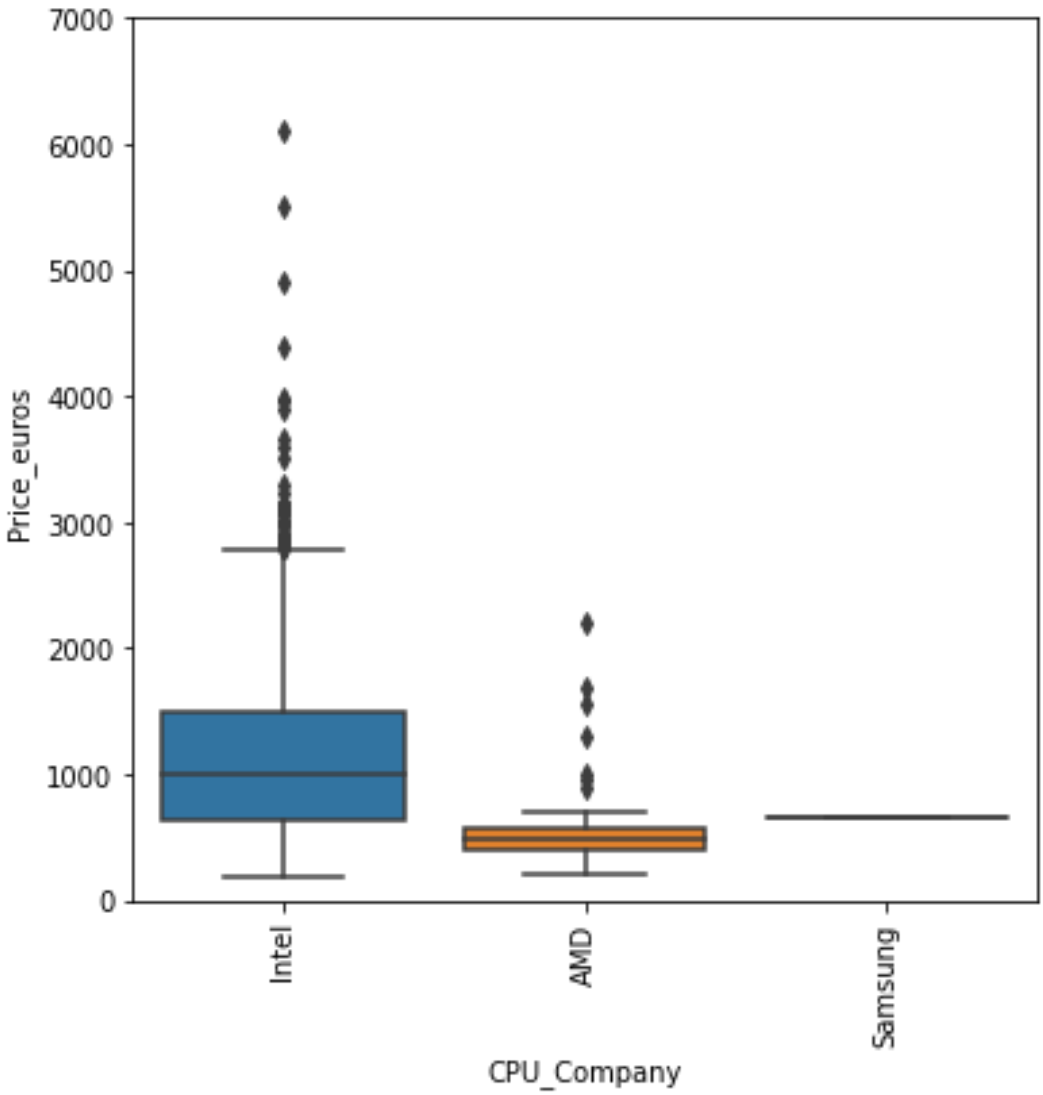
```
# Boxplot Company/Price_euros:
var = 'Company'
data = pandas.concat([laptops_initial['Price_euros'], laptops_initial[v
ar]], axis=1)
f, ax = plt.subplots(figsize=(13, 6))
fig = sns.boxplot(x=var, y="Price_euros", data = data)
fig.axis(ymin=0, ymax=7000);
plt.xticks(rotation=90);
```



Respecto a las marcas, si bien hay algunas con rangos claramente superiores o inferiores, la mayoría de ellas mantienen una media de precios estable entorno a los 1000-2000€, rango que por otra parte se ha comprobado anteriormente como la media del dataset. En este sentido, no haremos uso de esta variable para el modelo.

In [63]:

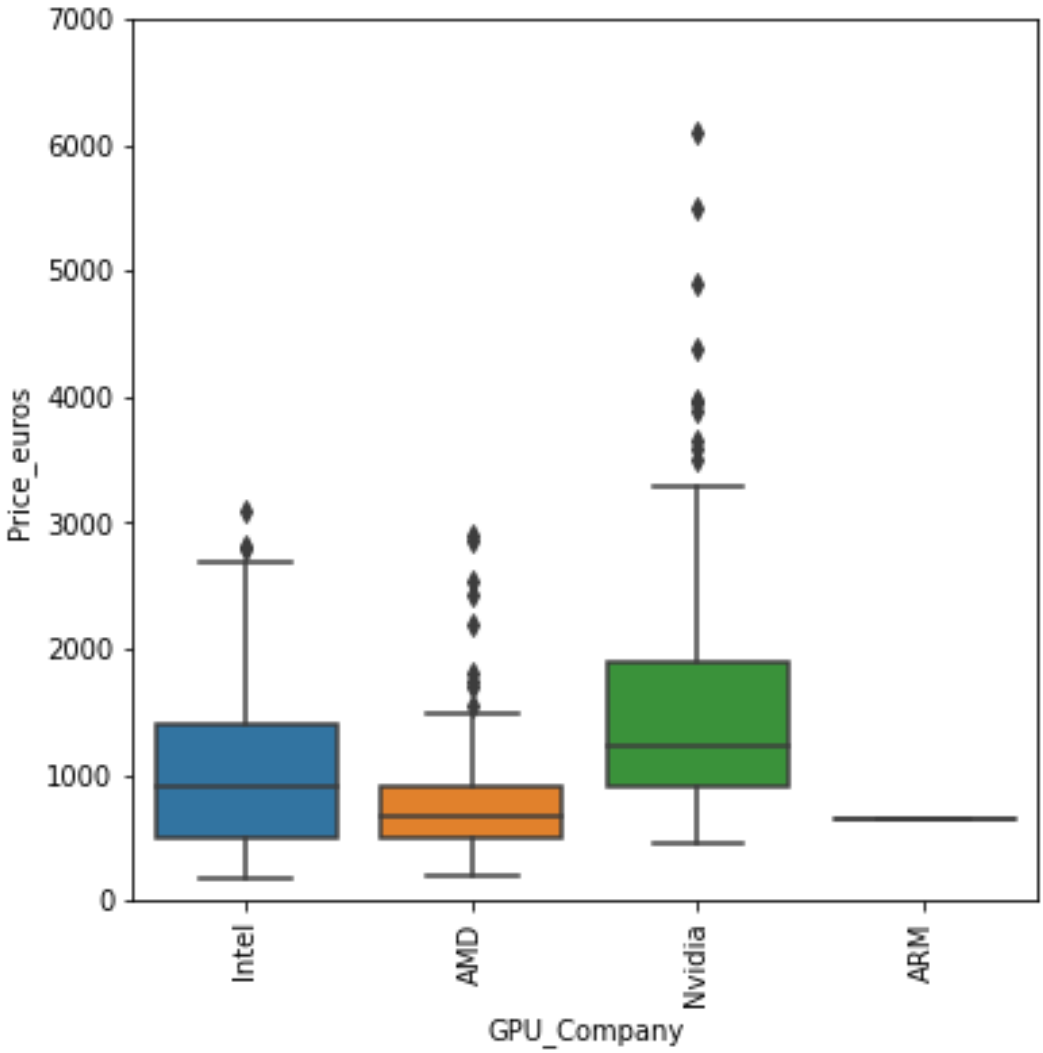
```
# Boxplot CPU_Company/Price_euros:
var = 'CPU_Company'
data = pandas.concat([laptops_initial['Price_euros'], laptops_initial[v
ar]], axis=1)
f, ax = plt.subplots(figsize=(6, 6))
fig = sns.boxplot(x=var, y="Price_euros", data = data)
fig.axis(ymin=0, ymax=7000);
plt.xticks(rotation=90);
```



En este caso, se ve una clara diferencia de medias entre las CPU de Intel y las de AMD, que son las dos que abarcan la mayoría de dispositivos. Vemos conveniente por tanto disponer de dicha información en el modelo.

In [64]:

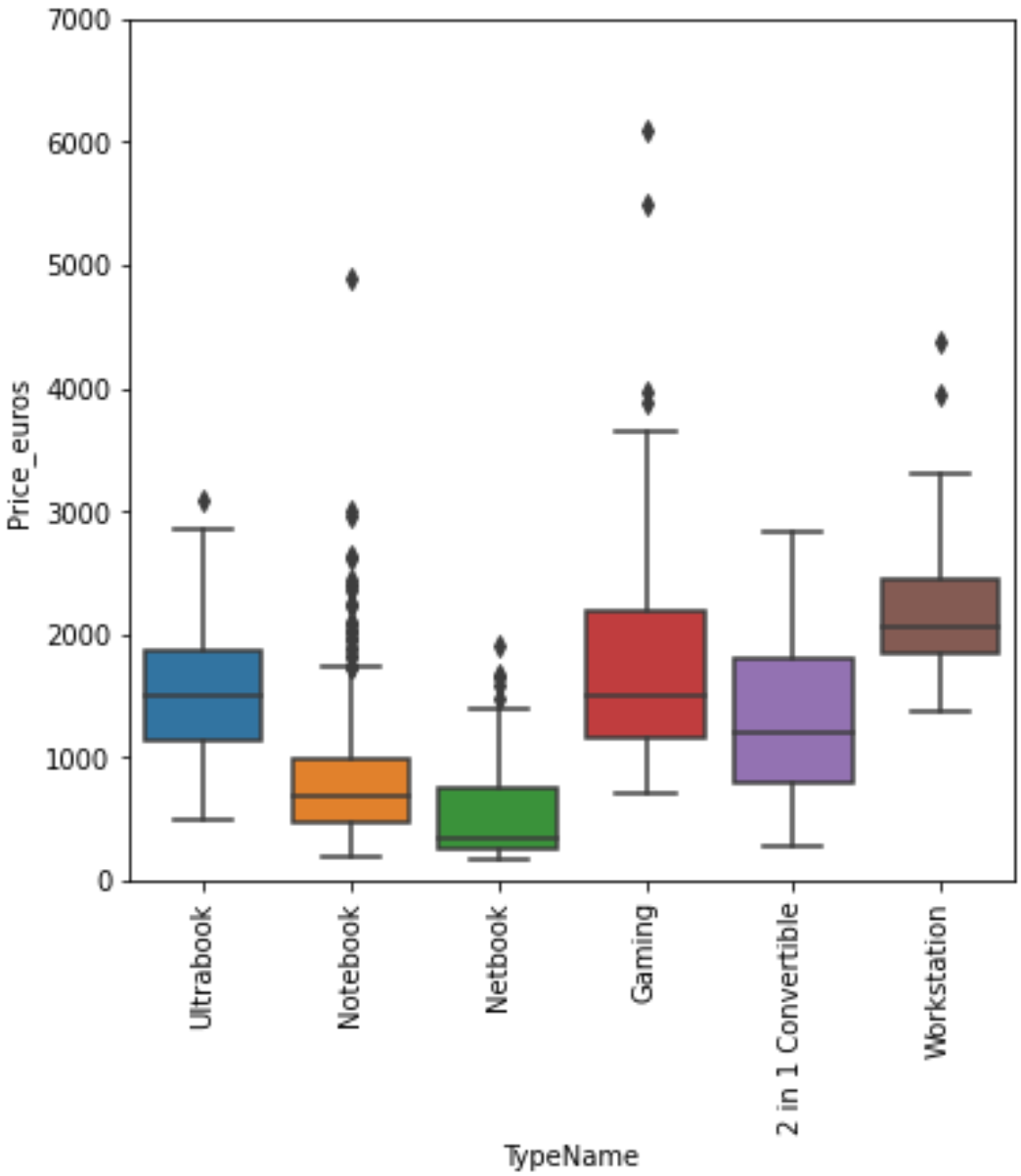
```
# Boxplot GPU_Company/Price_euros:
var = 'GPU_Company'
data = pandas.concat([laptops_initial['Price_euros'], laptops_initial[v
ar]], axis=1)
f, ax = plt.subplots(figsize=(6, 6))
fig = sns.boxplot(x=var, y="Price_euros", data = data)
fig.axis(ymin=0, ymax=7000);
plt.xticks(rotation=90);
```



En este caso, aunque no de forma tan clara como en el de las CPU, se pueden apreciar lo que serían 3 gamas de GPU. De más barata a más cara estarían: AMD, Intel y Nvidia. Por lo tanto, consideramos relevante añadirlo al modelo.

In [65]:

```
# Boxplot TypeName/Price_euros:
var = 'TypeName'
data = pandas.concat([laptops_initial['Price_euros'], laptops_initial[v
ar]], axis=1)
f, ax = plt.subplots(figsize=(6, 6))
fig = sns.boxplot(x=var, y="Price_euros", data = data)
fig.axis(ymin=0, ymax=7000);
plt.xticks(rotation=90);
```



En tipos de dispositivos, si bien hay alguna categoría que destaca (como Gaming), el resto de dispositivos parece que abarcan una franja amplia de precios, con lo que no se ve recomendable incorporarla al modelo.

A continuación procedemos a codificar las variables categóricas como numéricas mediante el método de one-hot para su inclusión en el modelo que generaremos a posteriori. Dado que desgranamos una variable categórica en tantas variables como categorias tenga (k), siempre podremos eliminar una de estas variables resultantes para mejorar el rendimiento del modelo y evitar redundancias.

In [66]:

```
# One-Hot de CPU_Company
dummies = pandas.get_dummies(laptops_initial['CPU_Company'], drop_first = True)
dummies = dummies.rename(columns={"Intel": "CPU_Intel", "Samsung": "CPU_Samsung"})
laptops_initial = laptops_initial.join(dummies)
laptops_initial
```

Out[66]:

	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Pri
0	Apple	Ultrabook	13.3	8	macOS	1.37	
1	Apple	Ultrabook	13.3	8	macOS	1.34	
2	HP	Notebook	15.6	8	No OS	1.86	
3	Apple	Ultrabook	15.4	16	macOS	1.83	
4	Apple	Ultrabook	13.3	8	macOS	1.37	
...	...	...	...	...	...	...	
1298	Lenovo	2 in 1 Convertible	14.0	4	Windows 10	1.80	
1299	Lenovo	2 in 1 Convertible	13.3	16	Windows 10	1.30	
1300	Lenovo	Notebook	14.0	2	Windows 10	1.50	
1301	HP	Notebook	15.6	6	Windows 10	2.19	
1302	Asus	Notebook	15.6	4	Windows 10	2.20	

1303 rows × 20 columns

In [67]:

```
# One-Hot de GPU_Company
dummies = pandas.get_dummies(laptops_initial['GPU_Company'], drop_first
= True)
dummies = dummies.rename(columns={"ARM": "GPU_ARM", "Intel": "GPU_Inte
l", "Nvidia": "GPU_Nvidia"})
laptops_initial = laptops_initial.join(dummies)
laptops_initial
```

Out[67]:

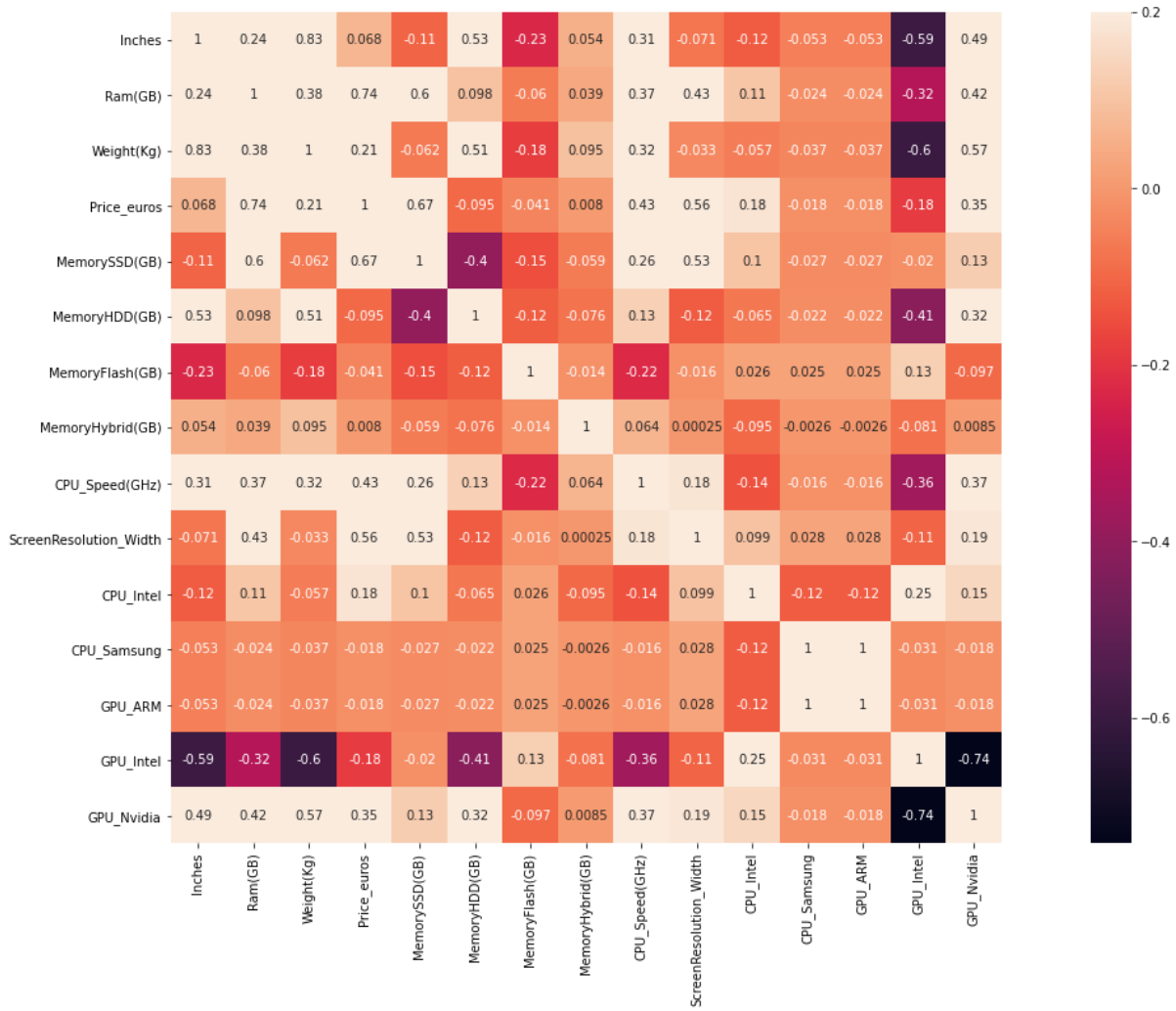
	Company	TypeName	Inches	Ram(GB)	OpSys	Weight(Kg)	Pri
0	Apple	Ultrabook	13.3	8	macOS	1.37	
1	Apple	Ultrabook	13.3	8	macOS	1.34	
2	HP	Notebook	15.6	8	No OS	1.86	
3	Apple	Ultrabook	15.4	16	macOS	1.83	
4	Apple	Ultrabook	13.3	8	macOS	1.37	
...	...	...	...	...	...	...	
1298	Lenovo	2 in 1 Convertible	14.0	4	Windows 10	1.80	
1299	Lenovo	2 in 1 Convertible	13.3	16	Windows 10	1.30	
1300	Lenovo	Notebook	14.0	2	Windows 10	1.50	
1301	HP	Notebook	15.6	6	Windows 10	2.19	
1302	Asus	Notebook	15.6	4	Windows 10	2.20	

1303 rows × 23 columns

Para tratar de ver el efecto real de estas variables, ejecutaremos de nuevo la matriz de correlación y veremos si la relación entre las variables que hemos pasado a numéricas y el precio es buena para incluirlas en el modelo.

In [68]:

```
# Matriz de correlación:
corrmat = laptops_initial.corr()
f, ax = plt.subplots(figsize=(25, 12))
sns.heatmap(corrmat, annot=True, vmax=.2, square=True);
```



A partir de los resultados, vemos que la única variables que podría ser relevante para el modelo es la de GPU_Nvidia, al tener una correlación con el precio de 0.35. También podría llegar a ser candidata CPU_Intel, al haber obtenido un 0.18 de correlación.

Creación y evaluación del modelo

In [69]:

```
# Generación del dataset para cargar en el modelo con las columnas sele
ccionadas
laptops_finished = laptops_initial[["Ram(GB)", "Weight(Kg)", "MemorySSD
(GB)", "CPU_Speed(GHz)", "ScreenResolution_Width", "GPU_Nvidia", "Price
_euros"]]
```

In [70]:

```
# Normalización de los valores del dataset
scaler = MinMaxScaler(feature_range=(0,1))
ind_cols = ["Ram(GB)", "Weight(Kg)", "MemorySSD(GB)", "CPU_Speed(GHz)",
"ScreenResolution_Width", "GPU_Nvidia", "Price_euros"]
final_data = laptops_finished.values
laptops_finished = pandas.DataFrame(scaler.fit_transform(final_data), c
olumns=ind_cols)
laptops_finished.head(5)
```

Out[70]:

	Ram(GB)	Weight(Kg)	MemorySSD(GB)	CPU_Speed(GHz)	ScreenRes
0	0.096774	0.169576	0.125	0.518519	
1	0.096774	0.162095	0.000	0.333333	
2	0.096774	0.291771	0.250	0.592593	
3	0.225806	0.284289	0.500	0.666667	
4	0.096774	0.169576	0.250	0.814815	

A continuación, procederemos a dividir el dataset en un subset de entrenamiento y otro de test para evaluar el rendimiento del modelo.

In [71]:

```
# Hacemos uso de la librería sklearn para dividir en conjunto de train
 y test
train, test = train_test_split(laptops_finished, test_size = 0.30)
```

In [72]:

```
# Preparación de las variables de train y test separando variables depe
ndientes e independientes
x_train = train.iloc[:, :-1]
y_train = train.iloc[:, -1]
x_test = test.iloc[:, :-1]
y_test = test.iloc[:, -1]

# Creación del modelo de regresión lineal
model = LinearRegression().fit(x_train, y_train)
r_sq = model.score(x_train, y_train)
print('Coefficient of determination:', round(r_sq, 2), "\n")

# Ejecución del modelo para predecir precios sobre el df de test
y_pred = model.predict(x_test)

# Inversión de la normalización para tener los precios en la escala cor
recta
# Se debe disponer de un dataframe del mismo tamaño que el normalizado
x_test = x_test.reset_index(drop=True)
interm_df = pandas.concat([x_test, pandas.Series(y_pred)], axis=1)

# Aplicación de la inversión de la normalización
y_pred = scaler.inverse_transform(interm_df)[:, -1]
y_test = scaler.inverse_transform(test)[:, -1]

# Generación de un dataframe para presentar los resultados reales y pre
dichos
pred_price = pandas.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(pred_price)
```

```
Coefficient of determination: 0.69

     Actual    Predicted
0    1186.0  1168.174767
1     998.0  1435.266345
2     579.0   714.353251
3    1109.0  1179.187858
4    1299.0  1505.429389
..      ...          ...
386   899.0  1697.164687
387   589.0   664.039042
388   989.0   880.534751
389  1249.0  1179.187858
390  1975.0  1739.722602

[391 rows x 2 columns]
```

Los resultados de esta regresión no dan un resultado excesivamente alto respecto a la precisión del modelo, ya que esta se queda en un coeficiente de determinación de 0.7, lo que indica que el 70% de la variabilidad de la variable dependiente está explicada por las variables independientes. No obstante, puede ser suficiente para identificar si un precio de mercado está por encima o por debajo de lo esperado por sus especificaciones.