Profit-margin-analysis-Naturgy

September 8, 2023

1 Business Planning, Control and Administration: Onerousness Model

After debugging in Excel, load the data to be worked with: - generation costs
- Industrial sales - Retail sales

Please note that all prices are in €/MWh and volumes in MWh.

[1]:
```
# Import the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import copy
```

[2]:
```
# Load the data to be worked with
coste_generacion = pd.read_excel('costes_generacion_totales.xlsx')
venta_industrial = pd.read_excel('precios_venta_industrial.xlsx')
venta_retail = pd.read_excel('precios_venta_retail.xlsx')
```

[3]:
```
coste_generacion
```

[3]:

	Mes	volumen_nuclear	coste_nuclear	volumen_hidro	coste_hidro	\
0	2023-07-01	300000	45.1	129000	83.0	
1	2023-08-01	337000	72.5	141000	61.2	
2	2023-09-01	498000	45.7	78000	151.4	
3	2023-10-01	419000	47.4	67000	138.3	
4	2023-11-01	404000	67.3	196000	47.9	
5	2023-12-01	515000	66.6	300000	61.3	
6	2024-01-01	292000	62.9	245000	25.7	
7	2024-02-01	370000	62.6	332000	29.7	
8	2024-03-01	366000	62.3	387000	25.9	
9	2024-04-01	245000	72.6	220000	65.6	
10	2024-05-01	232000	74.0	139000	61.2	
11	2024-06-01	204000	67.2	173000	43.3	

	volumen_renovables	coste_renovables	volumen_combi	coste_combi
0	358000	60.2	49000	42.6
1	459000	62.1	76000	56.6
2	411000	61.5	64000	36.7

3	455000	57.6	0	38.5
4	420000	44.4	0	60.0
5	419000	61.8	0	36.7
6	478000	37.6	11000	51.3
7	544000	62.4	11000	55.1
8	523000	42.5	12000	40.9
9	290000	50.4	1000	50.8
10	462000	41.2	0	49.7
11	296000	63.1	131000	64.4

[4]: venta_industrial

[4]:

	MES	precio	volumen
0	2023-07-01	11.061613	0.069747
1	2023-07-01	11.199500	443.399654
2	2023-07-01	11.323838	2.785784
3	2023-07-01	11.587623	1069.767076
4	2023-07-01	11.655044	4.075215
...	...	...	...
123411	2024-06-01	283.711331	4.312847
123412	2024-06-01	292.049235	77.539575
123413	2024-06-01	293.886915	1.602851
123414	2024-06-01	307.134436	0.010972
123415	2024-06-01	318.264745	0.095098

[123416 rows x 3 columns]

[5]: venta_retail

[5]:

	precio	2023-07-01 00:00:00	2023-08-01 00:00:00	2023-09-01 00:00:00	\
0	25.0	77.538720	76.120475	42.007294	
1	50.5	167.253842	118.543375	71.131872	
2	51.5	226.131598	176.700734	147.066446	
3	52.5	783.106320	238.364612	189.240796	
4	53.5	12595.761401	8832.957955	11163.484594	
..	...	...	...	...	
307	356.5	9618.207858	7403.236665	9562.660649	
308	357.5	4758.846111	3103.843323	5532.284895	
309	358.5	107.160411	57.053715	64.970121	
310	359.5	169.698338	102.147974	156.156927	
311	360.5	664.753597	916.413300	955.884195	

	2023-10-01 00:00:00	2023-11-01 00:00:00	2023-12-01 00:00:00	\
0	32.347748	25.569961	24.230931	
1	68.417808	64.300058	29.001756	
2	164.649968	134.925246	82.862259	
3	122.837894	167.598788	103.381071	

```
4              9293.900816          5862.186336          2577.592717
..                     ...                  ...                  ...
307            8596.174423          7145.148547          6987.214717
308            3467.849489          4252.200379          2679.683262
309              51.511949            35.748336            26.123738
310             154.358412           165.968854           133.045553
311            1006.698580          1057.403292          1063.285672

     2024-01-01 00:00:00  2024-02-01 00:00:00  2024-03-01 00:00:00  \
0              12.255006             3.328834             0.622916
1              23.974538            12.561189             0.901443
2              16.236562            16.002747            11.394577
3              52.990719            87.666911            62.564674
4             259.973705           137.438720           107.884385
..                   ...                  ...                  ...
307          5807.781362          4180.744976            88.462220
308          1435.532528           929.592937             9.216215
309            16.319738             3.778164             0.142613
310           164.774103           124.165051           140.882121
311           977.281571            96.239114            20.394426

     2024-04-01 00:00:00  2024-05-01 00:00:00  2024-06-01 00:00:00
0               0.000000             0.000000             0.000000
1               0.000000             0.000000             0.000000
2               5.389003             1.267214             0.918943
3              38.772554            30.295119            42.808891
4              92.419215            85.224480            78.261610
..                   ...                  ...                  ...
307            61.110165            99.691561            80.210330
308             0.000000             0.000000             0.000000
309             0.000000             0.000000             0.000000
310             0.000000             0.000000             0.000000
311            14.773249            11.868585            20.604884

[312 rows x 13 columns]
```

Brief processing of the retail DataFrame for a better visualisation of the structure of the data.

[6]:
```
# Get the names of the date columns (excluding the first column)
nombres_columnas_fechas = venta_retail.columns[1:]

# Get the month and year of each column name and create new names in␣
↪format "Apr 23, May 23, ..."
nuevos_nombres = [pd.to_datetime(nombre).strftime('%b %y') for nombre in
nombres_columnas_fechas]
```

```
# Assign the new column names to the DataFrame date columns
venta_retail.columns = ['precio'] + nuevos_nombres
```

[7]:
```
venta_retail.head()
```

[7]:
```
   precio        Jul 23       Aug 23       Sep 23       Oct 23       Nov 23  \
0    25.0     77.538720    76.120475    42.007294    32.347748    25.569961
1    50.5    167.253842   118.543375    71.131872    68.417808    64.300058
2    51.5    226.131598   176.700734   147.066446   164.649968   134.925246
3    52.5    783.106320   238.364612   189.240796   122.837894   167.598788
4    53.5  12595.761401  8832.957955  11163.484594  9293.900816  5862.186336

        Dec 23       Jan 24       Feb 24       Mar 24       Apr 24       May 24  \
0    24.230931    12.255006     3.328834     0.622916     0.000000     0.000000
1    29.001756    23.974538    12.561189     0.901443     0.000000     0.000000
2    82.862259    16.236562    16.002747    11.394577     5.389003     1.267214
3   103.381071    52.990719    87.666911    62.564674    38.772554    30.295119
4  2577.592717   259.973705   137.438720   107.884385    92.419215    85.224480

       Jun 24
0    0.000000
1    0.000000
2    0.918943
3   42.808891
4   78.261610
```

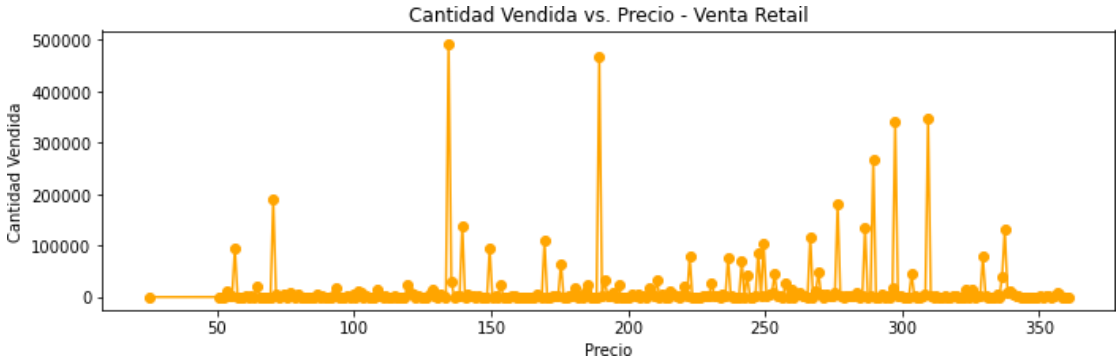
1.1 Creation of graphics

- Price distribution and quantity sold for retail sales

[8]:
```
# Create the price distribution chart
plt.figure(figsize=(10, 6))
```

[8]:
```
# Chart of quantity sold vs. price of the DataFrame sale_retail
plt.subplot(2, 1, 2)
plt.plot(venta_retail['precio'], venta_retail['Jul 23'], marker='o',␣
 ↪color='orange')
plt.title('Cantidad Vendida vs. Precio - Venta Retail')
plt.xlabel('Precio')
plt.ylabel('Cantidad  Vendida')

plt.tight_layout()
plt.show()
```



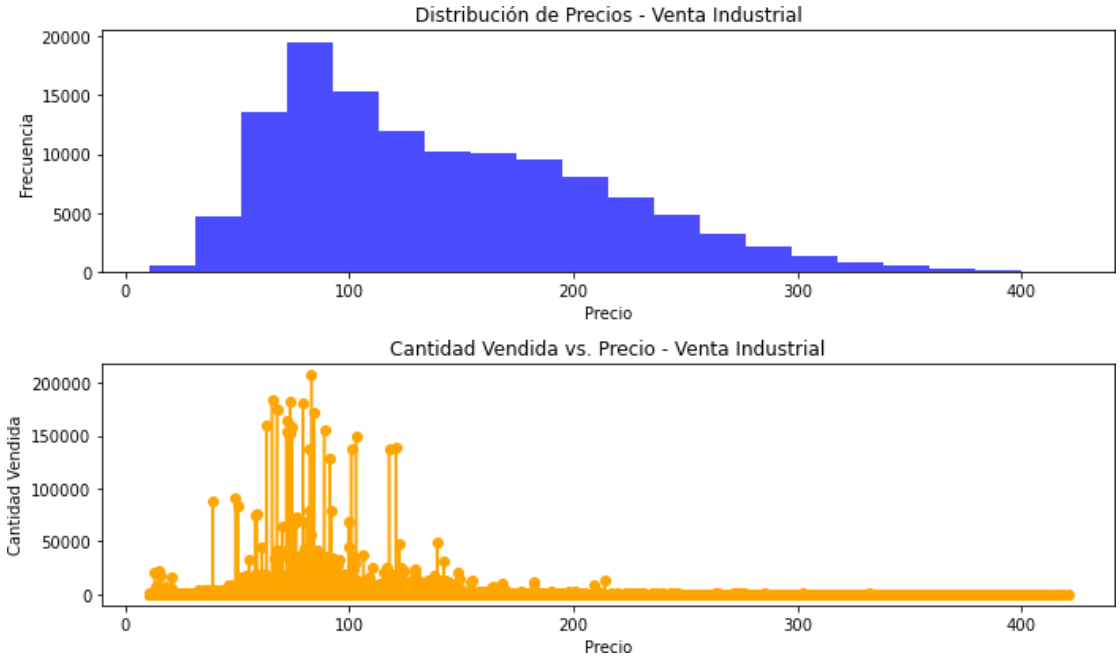
- Price distribution and quantity sold for industrial sales

[9]:
```
# Create the price distribution chart
plt.figure(figsize=(10, 6))

# Price histogram of the DataFrame venta_industrial
plt.subplot(2, 1, 1)
plt.hist(venta_industrial['precio'], bins=20, color='blue', alpha=0.7)
plt.title('Distribución de Precios - Venta Industrial')
plt.xlabel('Precio')
plt.ylabel('Frecuencia')

# Graph of quantity sold vs. price of the venta_industrial DataFrame
plt.subplot(2, 1, 2)
plt.plot(venta_industrial['precio'], venta_industrial['volumen'], marker='o',
 color='orange')
plt.title('Cantidad Vendida vs. Precio - Venta Industrial')
plt.xlabel('Precio')
plt.ylabel('Cantidad Vendida')

plt.tight_layout()
plt.show()
```

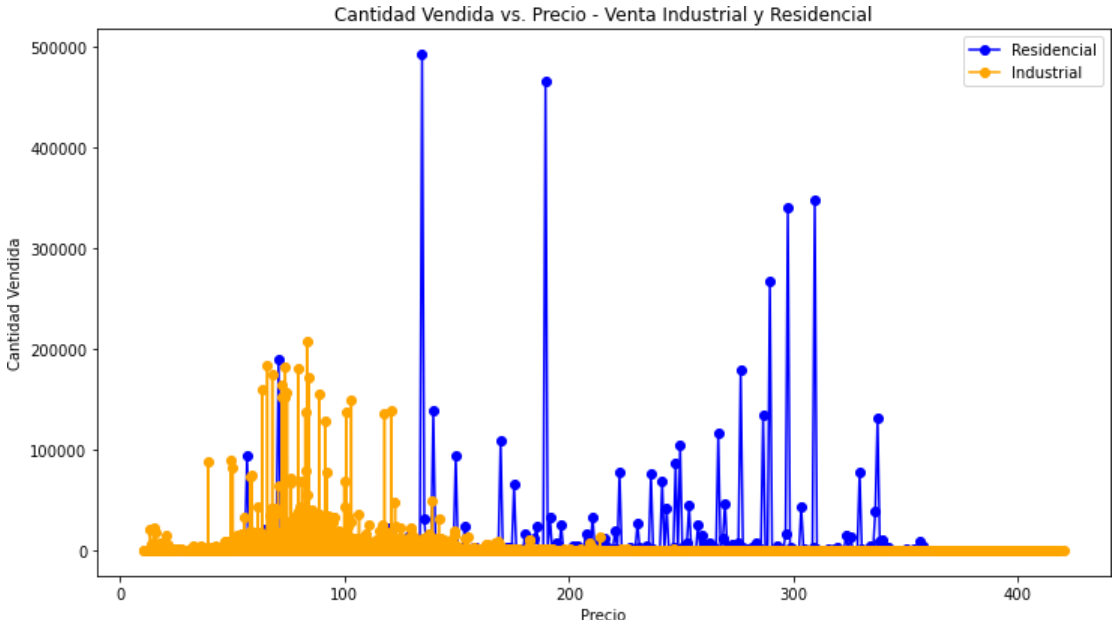


- Price distribution and quantity sold for total sales

[10]:
```
plt.figure(figsize=(20, 6))

# Graph of quantity sold vs. price of total sales
plt.subplot(1, 2, 1)
plt.plot(venta_retail['precio'], venta_retail['Jul 23'], marker='o',
 color='blue', label='Residencial')
plt.plot(venta_industrial['precio'], venta_industrial['volumen'], marker='o',
 color='orange', label='Industrial')
plt.title('Cantidad Vendida vs. Precio - Venta Industrial y Residencial')
plt.xlabel('Precio')
plt.ylabel('Cantidad Vendida')
plt.legend()

plt.tight_layout()
plt.show()
```



The distribution of the sales volume according to the sales price is very disparate between the two files, making it very difficult to choose dynamic price ranges that are well adjusted to both. For this reason, it is decided to work with equal intervals of 5€/MWh.

1.2 Sales processing

Create price ranges in the industrial sales file, grouping the data by month and sorting them from lowest to highest price

[11]:
```
venta_industrial.describe()
```

[11]:

	precio	volumen
count	123416.000000	123416.000000
mean	141.293671	193.807637
std	70.170733	2339.279498
min	10.443817	0.000091
25%	84.585370	2.589358
50%	126.364753	12.682747
75%	188.383520	49.364226
max	420.574164	207776.761045

The minimum price of the industrial sales data is 10.44€/MWh, so the intervals created will start at 10€/MWh and increase by 5€/MWh, in order to be in accordance with the process that will be followed for the retail sales data and, later on, to be able to concatenate both DataFrames.

[12]:
```
# Calculate the maximum value in prices
max_price = venta_industrial['precio'].max()
```

```
# Create intervals from the minimum to the maximum value+ 5 by 5
intervalos = range(10, int(max_price) + 5, 5)

# Add a column to the DataFrame indicating to which range each price belongs to
venta_industrial['Rango_Precio'] = pd.cut(venta_industrial['precio'],
 bins=intervalos, right=False)

# Convert the date column to datetime type
venta_industrial['MES'] = pd.to_datetime(venta_industrial['MES'])

# Sort the DataFrame by the column of months in chronological order and then by the
price range
venta_industrial = venta_industrial.sort_values(by=['MES', 'Rango_Precio'])

# Perform aggregation to calculate the average price and sum of the volume
industrial_rangos = venta_industrial.groupby(['MES', 'Rango_Precio']).agg({
    'precio': 'mean',
    'volumen': 'sum'
}).reset_index()
```

[13]:
```
industrial_rangos
```

[13]:

	MES	Rango_Precio	precio	volumen
0	2023-07-01	[10, 15)	13.208874	6045.816117
1	2023-07-01	[15, 20)	17.302245	38760.517078
2	2023-07-01	[20, 25)	22.211753	4589.759293
3	2023-07-01	[25, 30)	27.486343	3312.084203
4	2023-07-01	[30, 35)	33.092617	10438.770905
..	...	...	...	...
979	2024-06-01	[395, 400)	NaN	0.000000
980	2024-06-01	[400, 405)	NaN	0.000000
981	2024-06-01	[405, 410)	NaN	0.000000
982	2024-06-01	[410, 415)	NaN	0.000000
983	2024-06-01	[415, 420)	NaN	0.000000

[984 rows x 4 columns]

Process the retail sales data in such a way as to obtain a table with the same structure as industrial sales. In addition, create price ranges by grouping the data again by month and sorting them from lowest to highest price.

[14]:
```
venta_retail.describe()
```

[14]:

	precio	Jul 23	Aug 23	Sep 23	Oct 23	\
count	312.000000	312.000000	312.000000	312.000000	312.000000	

mean	204.921474	15429.607214	14354.421975	13131.961026	11152.342533
std	90.357202	54721.734432	52302.723599	48610.400766	38624.638729
min	25.000000	37.336197	32.083741	13.615851	12.450103
25%	127.250000	247.775611	160.598280	96.505750	86.217704
50%	205.000000	1117.138073	1028.279607	904.978381	825.716772
75%	282.750000	4753.763340	4594.159649	4002.509508	3900.831018
max	360.500000	493385.374682	461032.713578	442560.510655	324905.207178

	Nov 23	Dec 23	Jan 24	Feb 24	Mar 24 \
count	312.000000	312.000000	312.000000	312.000000	312.000000
mean	8441.780108	6998.067037	4725.598118	2533.224514	133.980953
std	28258.594290	24079.320949	15985.360285	9080.123104	391.761316
min	9.335662	4.330354	1.305440	0.691694	0.000000
25%	65.044356	51.578253	35.721196	27.725278	4.303993
50%	756.412319	638.431043	292.703680	154.842292	22.896898
75%	3482.785548	2922.732914	2068.362312	1236.804185	81.745777
max	267557.420078	217375.600397	159778.425627	86195.955485	3885.430782

	Apr 24	May 24	Jun 24
count	312.000000	312.000000	312.000000
mean	43.284310	29.664716	26.688204
std	305.248260	271.084936	265.140562
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.456774	0.210587	0.066128
75%	28.940658	14.401230	6.098722
max	5294.293989	4776.003874	4664.441494

The minimum price for retail sales is 25, so the price ranges will start at this figure and increase by 5 to match the industrial sales data and combine the two.

[15]:
```
# Convert columns of months to rows and keep only the relevant columns
venta_retail_melted = venta_retail.melt(id_vars=['precio'], var_name='MES',␣
 ↪value_name='volumen')

# Calculate the maximum value in prices
max_price = venta_retail_melted['precio'].max()

# Create intervals from the minimum to the maximum value+ 5 by 5
intervalos = range(25, int(max_price) + 5, 5)

# Add a column for the price range each row belongs to
venta_retail_melted['Rango_Precio'] = pd.cut(venta_retail_melted['precio'],␣
 ↪bins=intervalos, right=False)
```

```
# Convert the column MES to type datetime
venta_retail_melted['MES'] = pd.to_datetime(venta_retail_melted['MES'],
 format='%b %y')

# Sort the DataFrame by the column MONTH in chronological order then by Price_Range
venta_retail_melted = venta_retail_melted.sort_values(by=['MES',
 'Rango_Precio'])

# Maintain all data (price and volume) for each price range in each month
retail_rangos = venta_retail_melted.groupby(['MES', 'Rango_Precio']).
 apply(lambda x: x[['MES', 'Rango_Precio', 'precio', 'volumen']]).
 reset_index(drop=True)

retail_rangos = retail_rangos.groupby(['MES', 'Rango_Precio']).agg({
    'precio': 'mean',
    'volumen': 'sum'
    }).reset_index()
```

[16]:
```
retail_rangos
```

[16]:

	MES	Rango_Precio	precio	volumen
0	2023-07-01	[25, 30)	25.0	77.538720
1	2023-07-01	[30, 35)	NaN	0.000000
2	2023-07-01	[35, 40)	NaN	0.000000
3	2023-07-01	[40, 45)	NaN	0.000000
4	2023-07-01	[45, 50)	NaN	0.000000
..	...	...	...	...
799	2024-06-01	[335, 340)	337.5	1.140650
800	2024-06-01	[340, 345)	342.5	30.687737
801	2024-06-01	[345, 350)	347.5	55.939438
802	2024-06-01	[350, 355)	352.5	109.489140
803	2024-06-01	[355, 360)	357.5	86.305210

[804 rows x 4 columns]

Once the same data structure is achieved in both DataFrames, add a distinctive column to each so that: - I corresponds to industrial sales data - R corresponds to retail sales data

Concatenate both DataFrames, group the data by month and year and sort them from lowest to highest price range

[17]:
```
industrial_rangos['Origen'] = 'I'
retail_rangos['Origen'] = 'R'
```

[18]:
```
# Concatenate DataFrames using pd.concat and resetting the index
venta_total = pd.concat([industrial_rangos, retail_rangos], ignore_index=True)

# Sort the combined DataFrame by the columns MES, Rango_Precio, precio y Origen
venta_total = venta_total.sort_values(by=['MES', 'Rango_Precio', 'precio',
 's'Origen'])

#  Reindex the  DataFrame
venta_total = venta_total.reset_index(drop=True)
```

[19]:
```
venta_total
```

[19]:

	MES	Rango_Precio	precio	volumen	Origen
0	2023-07-01	[10, 15)	13.208874	6045.816117	I
1	2023-07-01	[15, 20)	17.302245	38760.517078	I
2	2023-07-01	[20, 25)	22.211753	4589.759293	I
3	2023-07-01	[25, 30)	25.000000	77.538720	R
4	2023-07-01	[25, 30)	27.486343	3312.084203	I
...	...	...	...	...	...
1783	2024-06-01	[395, 400)	NaN	0.000000	I
1784	2024-06-01	[400, 405)	NaN	0.000000	I
1785	2024-06-01	[405, 410)	NaN	0.000000	I
1786	2024-06-01	[410, 415)	NaN	0.000000	I
1787	2024-06-01	[415, 420)	NaN	0.000000	I

[1788 rows x 5 columns]

Add some columns to the total sales DataFrame that will become relevant later for the onerousness model:

- Sale (€) : Price (€/MWh) x Volume (MWh)
- Total Cost (€) : Average Cost (€/MWh) x Allocated Volume (MWh)
- Unit Margin (€/MWh) : Price (€/MWh) - Average Cost (€/MWh)
- Total Margin (€) : Sale (€) - Total Cost (€)

[20]:
```
venta_total["Venta"] = ""
venta_total["Coste Total"] = ""
venta_total["Margen unitario"] = ""
venta_total["Margen total"] = ""
```

[21]:
```
venta_total
```

[21] :

	MES	Rango_Precio	precio	volumen	Origen	Venta	\
0	2023-07-01	[10, 15)	13.208874	6045.816117	I		
1	2023-07-01	[15, 20)	17.302245	38760.517078	I		
2	2023-07-01	[20, 25)	22.211753	4589.759293	I		
3	2023-07-01	[25, 30)	25.000000	77.538720	R		

```
4     2023-07-01      [25, 30)  27.486343  3312.084203      ▌
...          ...           ...        ...          ...     ...     ...
1783  2024-06-01    [395, 400)        NaN     0.000000      ▌
1784  2024-06-01    [400, 405)        NaN     0.000000      ▌
1785  2024-06-01    [405, 410)        NaN     0.000000      ▌
1786  2024-06-01    [410, 415)        NaN     0.000000      ▌
1787  2024-06-01    [415, 420)        NaN     0.000000      ▌

      Coste Total Margen unitario Margen total
0
1
2
3
4
…           …                 …
            … 1783
1784
1785
1786
1787

[1788 rows x 9 columns]
```

Create a dictionary in which each key corresponds to each month.

[22] :
```
# Convert column "MES" to type datetime.datetime
venta_total["MES"] = pd.to_datetime(venta_total["MES"])

# Create a dictionary to store the monthly DataFrames
venta_mensual = {}

# Iterate through the single months and create monthly DataFrames
for date in venta_total["MES"].dt.to_period("M").unique():
    year = date.year
    month = date.month
    df_name = f"{year}_{month:02d}"        # Create the DataFrame name
    df_month = venta_total[
        (venta_total["MES"].dt.year == year) & (venta_total["MES"].dt.month ==␣
 ↪month)
    ].copy()    # Filter by month
    venta_mensual[df_name] = df_month
```

[23] :
```
type(venta_mensual)
```

[23] :
```
dict
```

[24] :
```
venta_mensual.keys()
```

[24] :
```
dict_keys(['2023_07', '2023_08', '2023_09', '2023_10', '2023_11', '2023_12',
'2024_01', '2024_02', '2024_03', '2024_04', '2024_05', '2024_06'])
```

[25] :
```
venta_mensual['2023_07']
```

[25]:

	MES	Rango_Precio	precio	volumen	Origen	Venta	\
0	2023-07-01	[10, 15)	13.208874	6045.816117		I	
1	2023-07-01	[15, 20)	17.302245	38760.517078		I	
2	2023-07-01	[20, 25)	22.211753	4589.759293		I	
3	2023-07-01	[25, 30)	25.000000	77.538720		R	
4	2023-07-01	[25, 30)	27.486343	3312.084203		I	
..	...	...	...	...	...	...	
144	2023-07-01	[395, 400)	397.560363	3.787721		I	
145	2023-07-01	[400, 405)	401.727642	23.264327		I	
146	2023-07-01	[405, 410)	409.279706	16.447313		I	
147	2023-07-01	[410, 415)	NaN	0.000000		I	
148	2023-07-01	[415, 420)	415.423771	5.210364		I	

	Coste Total	Margen unitario	Margen total
0			
1			
2			
3			
4			
..	...	...	...
144			
145			
146			
147			
148			

[149 rows x 9 columns]

1.2 Cost processing

Create a dictionary in which each key corresponds to each month.

[26] :
```
# Convert column "Month" to type datetime.datetime
coste_generacion["Mes"] = pd.to_datetime(coste_generacion["Mes"])

# Create a dictionary to store the monthly DataFrames
coste_generacion_mensual = {}

# Iterate through the single months and create monthly DataFrames
for date in coste_generacion["Mes"].dt.to_period("M").unique():
    year = date.year
    month = date.month
```

```
    df_name = f"{year}_{month:02d}"        # Create the DataFrame name
    print(f"Generando DataFrame: {df_name}")    # Add this line for verify the generated
    key
    df_month = coste_generacion[
        (coste_generacion["Mes"].dt.year == year) & (coste_generacion["Mes"].dt.
month == month)
    ].copy()    # Filter by month
    print(df_month)  # Add this line to print the monthly DataFrame
    coste_generacion_mensual[df_name] = df_month
```

```
Generando DataFrame: 2023_07
         Mes  volumen_nuclear  coste_nuclear  volumen_hidro  coste_hidro  \
0 2023-07-01            300000           45.1         129000         83.0

   volumen_renovables  coste_renovables  volumen_combi  coste_combi
0              358000              60.2          49000         42.6
Generando DataFrame: 2023_08
         Mes  volumen_nuclear  coste_nuclear  volumen_hidro  coste_hidro  \
1 2023-08-01            337000           72.5         141000         61.2

   volumen_renovables  coste_renovables  volumen_combi  coste_combi
1              459000              62.1          76000         56.6
Generando DataFrame: 2023_09
         Mes  volumen_nuclear  coste_nuclear  volumen_hidro  coste_hidro  \
2 2023-09-01            498000           45.7          78000        151.4

   volumen_renovables  coste_renovables  volumen_combi  coste_combi
2              411000              61.5          64000         36.7
Generando DataFrame: 2023_10
         Mes  volumen_nuclear  coste_nuclear  volumen_hidro  coste_hidro  \
3 2023-10-01            419000           47.4          67000        138.3

   volumen_renovables  coste_renovables  volumen_combi  coste_combi
3              455000              57.6              0         38.5
Generando DataFrame: 2023_11
         Mes  volumen_nuclear  coste_nuclear  volumen_hidro  coste_hidro  \
4 2023-11-01            404000           67.3         196000         47.9

   volumen_renovables  coste_renovables  volumen_combi  coste_combi
4              420000              44.4              0         60.0
Generando DataFrame: 2023_12
         Mes  volumen_nuclear  coste_nuclear  volumen_hidro  coste_hidro  \
5 2023-12-01            515000           66.6         300000         61.3

   volumen_renovables  coste_renovables  volumen_combi  coste_combi
5              419000              61.8              0         36.7
Generando DataFrame: 2024_01
```

```
          Mes  volumen_nuclear  coste_nuclear   volumen_hidro  coste_hidro   \
6 2024-01-01           292000           62.9          245000         25.7

   volumen_renovables  coste_renovables   volumen_combi coste_combi
6              478000              37.6           11000         51.3
Generando DataFrame: 2024_02
          Mes  volumen_nuclear  coste_nuclear   volumen_hidro  coste_hidro   \
7 2024-02-01           370000           62.6          332000         29.7

   volumen_renovables  coste_renovables   volumen_combi coste_combi
7              544000              62.4           11000         55.1
Generando DataFrame: 2024_03
          Mes  volumen_nuclear  coste_nuclear   volumen_hidro  coste_hidro   \
8 2024-03-01           366000           62.3          387000         25.9

   volumen_renovables  coste_renovables   volumen_combi coste_combi
8              523000              42.5           12000         40.9
Generando DataFrame: 2024_04
          Mes  volumen_nuclear  coste_nuclear   volumen_hidro  coste_hidro   \
9 2024-04-01           245000           72.6          220000         65.6

   volumen_renovables  coste_renovables   volumen_combi coste_combi
9              290000              50.4            1000         50.8
Generando DataFrame: 2024_05
          Mes  volumen_nuclear  coste_nuclear   volumen_hidro  coste_hidro   \
10 2024-05-01           232000           74.0          139000         61.2

    volumen_renovables  coste_renovables   volumen_combi coste_combi
10              462000              41.2               0         49.7
Generando DataFrame: 2024_06
          Mes  volumen_nuclear  coste_nuclear   volumen_hidro  coste_hidro   \
11 2024-06-01           204000           67.2          173000         43.3

    volumen_renovables  coste_renovables   volumen_combi coste_combi
11              296000              63.1          131000         64.4
```

[27] :
```
coste_generacion_mensual["2023_07"]
```

[27] :
```
          Mes  volumen_nuclear coste_nuclear   volumen_hidro coste_hidro   \
0 2023-07-01           300000           45.1          129000         83.0

   volumen_renovables  coste_renovables   volumen_combi coste_combi
0              358000              60.2           49000         42.6
```

Modify the structure of the data so that the indices correspond to the types of energy, the columns correspond to the price and volume of each one, and are ordered according to the lowest to highest price for each month.

[28] :
```
# Define the names of the price and volume columns for each type of product energy
columnas_precio = ["coste_combi", "coste_hidro", "coste_renovables", "coste_nuclear"]
columnas_volumen = ["volumen_combi", "volumen_hidro", "volumen_renovables", "volumen_nuclear"]

# Scroll through all monthly DataFrames and reorganise the data
for year in range(2023, 2025):
    for month in range(1, 13):
        df_key = f"{year}_{month:02d}"
        if df_key in coste_generacion_mensual:
            df_mensual = coste_generacion_mensual[df_key]

            df_reorganizado = pd.DataFrame(columns=["precio", "volumen"], index=["combi", "hidro", "renovables", "nuclear"])

            for tipo_energia, col_precio, col_volumen in zip(["combi", "hidro", "renovables", "nuclear"], columnas_precio, columnas_volumen):
                if col_precio in df_mensual.columns and col_volumen in df_mensual.columns:
                    precio_valor = df_mensual[col_precio].iloc[0]
                    volumen_valor = df_mensual[col_volumen].iloc[0]
                    df_reorganizado.loc[tipo_energia] = [precio_valor, volumen_valor]

            # Sort the rearranged DataFrame by price from smallest to largest
            df_reorganizado = df_reorganizado.sort_values(by="precio")

        # Save the reorganised DataFrame in a new column name
          in the original DataFrame

            new_column_name = f"{df_key}"
            coste_generacion_mensual[new_column_name] = df_reorganizado
```

[29] :
```
type(coste_generacion_mensual)
```

[29] : dict

[30] :
```
coste_generacion_mensual.keys()
```

[30] : dict_keys(['2023_07', '2023_08', '2023_09', '2023_10', '2023_11', '2023_12', '2024_01', '2024_02', '2024_03', '2024_04', '2024_05', '2024_06'])

[31] :
```
coste_generacion_mensual["2023_07"]
```

[31]:

	precio	volumen
combi	42.6	49000
nuclear	45.1	300000
renovables	60.2	358000
hidro	83.0	129000

[32]:
```
coste_generacion_mensual["2024_01"]
```

[32]:

	precio	volumen
hidro	25.7	245000
renovables	37.6	478000
combi	51.3	11000
nuclear	62.9	292000

1.3 Cost allocation by price range

Create a copy of the generation cost data, so as not to overwrite the original data.

[33]:
```
coste_generacion_mensual_v2 = copy.deepcopy(coste_generacion_mensual)
```

[34]:
```
coste_generacion_mensual_v2["2023_07"]
```

[34]:

	precio	volumen
combi	42.6	49000
nuclear	45.1	300000
renovables	60.2	358000
hidro	83.0	129000

[35]:
```
for mes in venta_mensual.keys():
    print(mes)

    venta_mensual[mes]["volumen_asignado"] = 0
    venta_mensual[mes]["Coste medio"] = 0

    fila_coste = 0
    for venta in venta_mensual[mes]["volumen"].index.values:
        volumen = venta_mensual[mes]["volumen"][venta]
        print('el volumen que necesito es', volumen)
        coste_medio = 0
        volumen_total = volumen
        fila_coste = fila_coste
        while volumen > 0 and fila_coste <␣
 ↪len(coste_generacion_mensual_v2[mes]):

            if volumen <= coste_generacion_mensual_v2[mes].
 ↪iloc[fila_coste]['volumen']:
                print('me sobra con esta energia')
```

```
                    coste_generacion_mensual_v2[mes].iloc[fila_coste]['volumen']     -=␣
↪volumen
                    venta_mensual[mes].loc[venta, "volumen_asignado"] += volumen
                    coste_medio += volumen / volumen_total *␣
↪coste_generacion_mensual_v2[mes].iloc[fila_coste]['precio']
                    print('me queda tanto de esta energia',␣
↪coste_generacion_mensual_v2[mes].iloc[fila_coste]['volumen'] )
                    volumen = 0
                    print('el coste medio es', coste_medio)

                else:
                    print('no es suficiente con esta energia')
                    volumen_asignado = coste_generacion_mensual_v2[mes].
↪iloc[fila_coste]['volumen']
                    coste_generacion_mensual_v2[mes].iloc[fila_coste]['volumen'] = 0
                    venta_mensual[mes].loc[venta, "volumen_asignado"] +=␣
↪volumen_asignado
                    volumen -= volumen_asignado
                    coste_medio += volumen_asignado / volumen_total *␣
↪coste_generacion_mensual_v2[mes].iloc[fila_coste]['precio']
                    fila_coste += 1
                if volumen > 0 and fila_coste ==␣
↪len(coste_generacion_mensual_v2[mes]):
                    # Allocate the weighted average cost of the remaining volume with the last
                    price
                     coste_medio += volumen / volumen_total *␣
↪coste_generacion_mensual_v2[mes].iloc[-1]['precio']
                venta_mensual[mes].loc[venta, "Coste medio"] = coste_medio
```

```
2023_07
el volumen que necesito es 6045.816117470881
me sobra con esta energia
me queda tanto de esta energia 42954.18388252912
el coste medio es 42.6
el volumen que necesito es 38760.517077739205
me sobra con esta energia
me queda tanto de esta energia 4193.666804789915
el coste medio es 42.6
el volumen que necesito es 4589.759292509972
no es suficiente con esta energia
me sobra con esta energia
me queda tanto de esta energia 299603.9075122799
el coste medio es 42.81574796327906
el volumen que necesito es 77.5387199692827
me sobra con esta energia
me queda tanto de esta energia 299526.36879231065
el coste medio es 45.1
```

el volumen que necesito es 3312.0842031043608
me sobra con esta energia
me queda tanto de esta energia 296214.2845892063
el coste medio es 45.1
el volumen que necesito es 10438.77090475199
me sobra con esta energia
me queda tanto de esta energia 285775.51368445426
el coste medio es 45.1
el volumen que necesito es 0.0
el volumen que necesito es 13043.557553908975
me sobra con esta energia
me queda tanto de esta energia 272731.9561305453
el coste medio es 45.1
el volumen que necesito es 0.0
el volumen que necesito es 17749.891230593657
me sobra con esta energia
me queda tanto de esta energia 254982.06489995163
el coste medio es 45.1
el volumen que necesito es 0.0
el volumen que necesito es 103079.26507958089
me sobra con esta energia
me queda tanto de esta energia 151902.79982037074
el coste medio es 45.1
el volumen que necesito es 0.0
el volumen que necesito es 17411.459195128155
me sobra con esta energia
me queda tanto de esta energia 134491.3406252426
el coste medio es 45.1
el volumen que necesito es 35630.590240518686
me sobra con esta energia
me queda tanto de esta energia 98860.75038472391
el coste medio es 45.1
el volumen que necesito es 99968.86474243819
no es suficiente con esta energia
me sobra con esta energia
me queda tanto de esta energia 356891.8856422857
el coste medio es 45.26737738139366
el volumen que necesito es 60585.67558561834
me sobra con esta energia
me queda tanto de esta energia 296306.21005666733
el coste medio es 60.2
el volumen que necesito es 26896.881341157357
me sobra con esta energia
me queda tanto de esta energia 269409.32871550997
el coste medio es 60.2
el volumen que necesito es 96090.61238778346
me sobra con esta energia
me queda tanto de esta energia 173318.7163277265

el coste medio es 60.2
el volumen que necesito es 1528.2327564757315
me sobra con esta energia
me queda tanto de esta energia 171790.48357125075
el coste medio es 60.2
el volumen que necesito es 164569.12439748918
me sobra con esta energia
me queda tanto de esta energia 7221.359173761564
el coste medio es 60.2
el volumen que necesito es 398870.01708636014
no es suficiente con esta energia
no es suficiente con esta energia
el volumen que necesito es 205898.7186156449
el volumen que necesito es 19097.804303089062
el volumen que necesito es 161075.20368594915
el volumen que necesito es 213740.2164304287
el volumen que necesito es 1816.1890202257453
el volumen que necesito es 142931.28679671732
el volumen que necesito es 7667.801731413311
el volumen que necesito es 19490.112347662056
el volumen que necesito es 152129.6859149425
el volumen que necesito es 7350.270096698053
el volumen que necesito es 78528.45516619844
el volumen que necesito es 115338.26496481943
el volumen que necesito es 22693.541020699835
el volumen que necesito es 50617.57803064146
el volumen que necesito es 16458.38834299618
el volumen que necesito es 3899.612955366813
el volumen que necesito es 48063.093382900166
el volumen que necesito es 74645.31577223701
el volumen que necesito es 23969.36858930175
el volumen que necesito es 13883.75729024499
el volumen que necesito es 53703.96514410097
el volumen que necesito es 86972.99008898539
el volumen que necesito es 24248.063112091542
el volumen que necesito es 500707.1405616114
el volumen que necesito es 39889.94755717077
el volumen que necesito es 175393.30709124022
el volumen que necesito es 50111.17886242878
el volumen que necesito es 49185.51408071199
el volumen que necesito es 8092.311541757188
el volumen que necesito es 31492.199549687295
el volumen que necesito es 96808.20463167781
el volumen que necesito es 26439.583169224697
el volumen que necesito es 26304.597619445583
el volumen que necesito es 32198.578633104993
el volumen que necesito es 6762.220306993517
el volumen que necesito es 22159.30885466366

```
el volumen que necesito es 0.0
el volumen que necesito es 0.0
el volumen que necesito es 0.0
el volumen que necesito es 0.0
el volumen que necesito es 0.0
el volumen que necesito es 0.0
```

[36]:
```
venta_mensual["2023_08"].head(25)
```

[36]:

	MES	Rango_Precio	precio	volumen	Origen Venta	\
149	2023-08-01	[10, 15)	13.031470	405.670156	I	
150	2023-08-01	[15, 20)	17.460059	27241.453858	I	
151	2023-08-01	[20, 25)	22.251894	19130.741890	I	
152	2023-08-01	[25, 30)	25.000000	76.120475	R	
153	2023-08-01	[25, 30)	27.503019	4393.036633	I	
154	2023-08-01	[30, 35)	32.994857	7301.474626	I	
155	2023-08-01	[30, 35)	NaN	0.000000	R	
156	2023-08-01	[35, 40)	37.634147	102686.579540	I	
157	2023-08-01	[35, 40)	NaN	0.000000	R	
158	2023-08-01	[40, 45)	42.568438	16825.160797	I	
159	2023-08-01	[40, 45)	NaN	0.000000	R	
160	2023-08-01	[45, 50)	47.461347	15592.593237	I	
161	2023-08-01	[45, 50)	NaN	0.000000	R	
162	2023-08-01	[50, 55)	52.500000	11176.196762	R	
163	2023-08-01	[50, 55)	52.551032	44145.661121	I	
164	2023-08-01	[55, 60)	57.500000	97093.195452	R	
165	2023-08-01	[55, 60)	57.592235	127056.648995	I	
166	2023-08-01	[60, 65)	62.500000	28270.367090	R	
167	2023-08-01	[60, 65)	62.549297	89449.560047	I	
168	2023-08-01	[65, 70)	67.500000	915.499865	R	
169	2023-08-01	[65, 70)	67.538348	110080.200081	I	
170	2023-08-01	[70, 75)	72.500000	315089.493529	R	
171	2023-08-01	[70, 75)	72.512139	112160.992992	I	
172	2023-08-01	[75, 80)	77.500000	18649.461604	R	
173	2023-08-01	[75, 80)	77.549787	369224.984031	I	

	Coste Total	Margen unitario	Margen total	volumen_asignado	Coste medio
149				405.670156	56.600000
150				27241.453858	56.600000
151				19130.741890	56.600000
152				76.120475	56.600000
153				4393.036633	56.600000
154				7301.474626	56.600000
155				0.000000	0.000000
156				102686.579540	60.418234
157				0.000000	0.000000
158				16825.160797	61.200000

159	0.000000	0.000000
160	15592.593237	61.200000
161	0.000000	0.000000
162	11176.196762	61.200000
163	44145.661121	61.851870
164	97093.195452	62.100000
165	127056.648995	62.100000
166	28270.367090	62.100000
167	89449.560047	62.100000
168	915.499865	62.100000
169	110080.200081	64.541290
170	311159.839374	72.500000
171	0.000000	0.000000
172	0.000000	0.000000
173	0.000000	0.000000

Check that the energy volumes of the copied generation cost data have been correctly allocated, while the originals remain unchanged.

[37]:
```
coste_generacion_mensual_v2["2023_07"]
```

[37]:

	precio	volumen
combi	42.6	0
nuclear	45.1	0
renovables	60.2	0
hidro	83.0	0

[38]:
```
coste_generacion_mensual["2023_07"]
```

[38]:

	precio	volumen
combi	42.6	49000
nuclear	45.1	300000
renovables	60.2	358000
hidro	83.0	129000

Check for a month that the sum of allocated volumes corresponds to the sum of the total volume to be allocated.

[39]:
```
suma_total = venta_mensual["2023_07"]["volumen_asignado"].sum()
print("Suma total de volumen_asignado:", suma_total)
```

```
Suma total de volumen_asignado: 835999.9999999998
```

[40]:
```
suma_total = coste_generacion_mensual["2023_07"]["volumen"].sum()
print("Suma total de volumen por asignar:", suma_total)
```

Total sum of volume to be allocated: 83600

[41] :
```
for mes, mes_data in venta_mensual.items():
    volumen_asignado = mes_data["volumen_asignado"]
    coste_medio = mes_data["Coste medio"]
    mes_data["Coste Total"] = [v * c for v, c in zip(volumen_asignado,␣
 ↪coste_medio)]
```

[42] :
```
for mes, mes_data in venta_mensual.items():
    precios = mes_data["precio"]
    costes_medios = mes_data["Coste medio"]

    # Calculate the unit margin only if the average cost is greater than 0.
    margen_unitario = [(p - c) if c > 0 else None for p, c in zip(precios,␣
 ↪costes_medios)]

    mes_data["Margen unitario"] = margen_unitario
```

[43] :
```
# Scroll through each month in the DataFrame
for month, data in venta_mensual.items():

    # Perform the multiplication and store the results in the column 'Sale'.
    data['Venta'] = [precio * volumen if volumen_asignado > 0 else 0 for␣
 ↪precio, volumen, volumen_asignado in zip(data['precio'], data['volumen'],␣
 ↪data['volumen_asignado'])]

# Print the updated DataFrame for one month (e.g. July 2023)
print(venta_mensual['2023_07'])
```

```
          MES Rango_Precio      precio      volumen Origen          Venta  \
0  2023-07-01      [10, 15)   13.208874  6045.816117      I   79858.424366
1  2023-07-01      [15, 20)   17.302245  38760.517078      I  670643.951443
2  2023-07-01      [20, 25)   22.211753  4589.759293      I   101946.599406
3  2023-07-01      [25, 30)   25.000000    77.538720      R    1938.467999
4  2023-07-01      [25, 30)   27.486343  3312.084203      I   91037.081840
..        ...           ...         ...         ...    ...           ...
144 2023-07-01    [395, 400)  397.560363     3.787721      I      0.000000
145 2023-07-01    [400, 405)  401.727642    23.264327      I      0.000000

146 2023-07-01    [405, 410)  409.279706    16.447313      I      0.000000
147 2023-07-01    [410, 415)         NaN     0.000000      I      0.000000
148 2023-07-01    [415, 420)  415.423771     5.210364      I      0.000000

      Coste Total  Margen unitario  Margen total  volumen_asignado  Coste medio
0    2.575518e+05       -29.391126                  6045.816117    42.600000
1    1.651198e+06       -25.297755                 38760.517078    42.600000
2    1.965140e+05       -20.603995                  4589.759293    42.815748
3    3.496996e+03       -20.100000                    77.538720    45.100000
```

```
4      1.493750e+05        -17.613657                    3312.084203     45.100000
..              ...               ...            ...             ...          ...
144   0.000000e+00               NaN                       0.000000      0.000000
145  0.000000e+00               NaN                       0.000000      0.000000
146  0.000000e+00               NaN                       0.000000      0.000000
147  0.000000e+00               NaN                       0.000000      0.000000
148  0.000000e+00               NaN                       0.000000      0.000000

[149 rows x 11 columns]
```

[44] :
```
for mes, mes_data in venta_mensual.items():
    ventas = mes_data["Venta"]
    costes_totales = mes_data["Coste Total"]

# Calculate the total margin by subtracting Sales minus Total Cost if Total Cost > 0
    margen_total = [v - c if v is not None and c is not None and c > 0 else␣
 ↪None for v, c in zip(ventas, costes_totales)]

    mes_data["Margen total"] = margen_total

# Show updated dictionaries
for month, month_data in venta_mensual.items():
    print(f"Mes: {month}")
    print(month_data)
    print("\n")
```

```
# Create a list of DataFrames from data
data_frames = [pd.DataFrame(datos) for datos in venta_mensual.values()]

# Concatenate the DataFrames into a single DataFrame
df_combined = pd.concat(data_frames, ignore_index=True)

# Create an Excel file
nombre_archivo = "ventas_mensuales_en_csv.csv"
df_combined.to_csv(nombre_archivo, index=False)

print(f"File {nombre_archivo} successfully created.")
```

```
File ventas_mensuales_en_csv.csv successfully created
```