energy_analysis_draft

November 4, 2019

1 PRÁCTICA TECNOLOGÍAS DE GESTIÓN DE LA INFORMACIÓN

Práctica hecha por: - Mendorito - Jesi

Esto es un texto de **prueba** para *ver* cómo va este ***texto***

Ir al comienzo de la libreta

[11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[12]:
```
electricity_prices = pd.read_csv("energy_data/electricity_prices_household.
 ↪csv", delimiter=";")
electricity_prices.shape

#tomamos solo las filas que contengan lo que nos interesa: datos de españa,␣
 ↪precios sin tasas añadidas y el precio representado en euros
es_electricity = electricity_prices.loc[(electricity_prices['geo\\time'] ==␣
 ↪'ES') & (electricity_prices['tax'] == 'X_TAX') &␣
 ↪(electricity_prices['currency'] == 'EUR')]
```

[13]:
```
es_electricity.head()
```

[13]:

	product	consom	unit	tax	currency	geo\time	2019S1	2018S2	2018S1	\
139	6000	4161901	KWH	X_TAX	EUR	ES	0.5166	0.4485	0.4680	
520	6000	4161902	KWH	X_TAX	EUR	ES	0.2355	0.2393	0.2303	
901	6000	4161903	KWH	X_TAX	EUR	ES	0.1889	0.1947	0.1873	
1282	6000	4161904	KWH	X_TAX	EUR	ES	0.1608	0.1697	0.1513	
1663	6000	4161905	KWH	X_TAX	EUR	ES	0.1409	0.1414	0.1236	

	2017S2	...	2011S2	2011S1	2010S2	2010S1	2009S2	2009S1	2008S2	\
139	0.4479	...	0.3044	0.2890	0.2926	0.3174	0.2611	0.2540	0.2622	
520	0.2113	...	0.1870	0.1768	0.1681	0.1622	0.1571	0.1459	0.1475	
901	0.1712	...	0.1684	0.1597	0.1492	0.1417	0.1381	0.1294	0.1277	
1282	0.1482	...	0.1563	0.1426	0.1363	0.1271	0.1266	0.1203	0.1219	
1663	0.1311	...	0.1444	0.1251	0.1244	0.1174	0.1153	0.1163	0.1148	

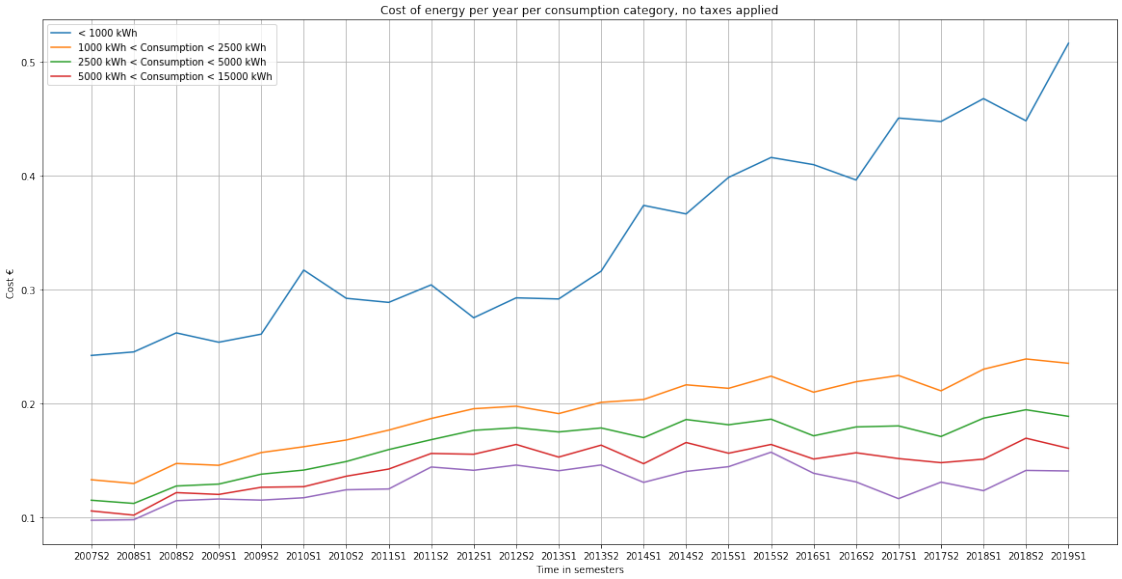
	2008S1	2007S2	2007S1

```
139   0.2455  0.2424     :
520   0.1299  0.1332     :
901   0.1124  0.1152     :
1282  0.1021  0.1058     :
1663  0.0981  0.0976     :

[5 rows x 31 columns]
```

[14]:
```
# eliminamos la columna correspondiente al primer semestre de 2007,
# ya que no hay datos de esa época
es_electricity = es_electricity.drop(columns="2007S1")

# le damos la vuelta porque los años están dispuestos del revés
# generamos la x que pondremos abajo en nuestra gráfica
x = np.flip(es_electricity.columns[6:].to_numpy())

# convertimos el DataFrame de pandas a un numpy array para manipularlo más␣
 ↪fácilmente
nparray = es_electricity.to_numpy()

# hacemos lo mismo que con la x, pero con la y
y1 = np.flip(nparray[0, 6:])
y2 = np.flip(nparray[1, 6:])
y3 = np.flip(nparray[2, 6:])
y4 = np.flip(nparray[3, 6:])
y5 = np.flip(nparray[4, 6:])

# el siguiente paso es castearlos a un tipo flotante,
# así que nos aseguramos de que los valores no numéricos
# no existan

y1[y1 == ': '] = -1
y2[y2 == ': '] = -1
y3[y3 == ': '] = -1
y4[y4 == ': '] = -1
y5[y5 == ': '] = -1

# convertimos el array a float, ya que hasta ahora era
# un array de strings
y1 = y1.astype(float)
y2 = y2.astype(float)
y3 = y3.astype(float)
y4 = y4.astype(float)
y5 = y5.astype(float)
```

[15]:
```
plt.figure(figsize=(20,10))
plt.plot(x, y1)
plt.plot(x, y2)
```

```
plt.plot(x, y3)
plt.plot(x, y4)
plt.plot(x, y5)

legend = ["< 1000 kWh", "1000 kWh < Consumption < 2500 kWh","2500 kWh <␣
 ↪Consumption < 5000 kWh","5000 kWh < Consumption < 15000 kWh",]
plt.legend(legend)

plt.ylabel('Cost ')
plt.xlabel('Time in semesters')

plt.title("Cost of energy per year per consumption category, no taxes applied")
plt.grid(True)
plt.show()
```



[16]:
```
elec_melt = electricity_prices.melt(
    id_vars=['product', 'consom', 'unit', 'tax', 'currency', 'geo\\time'],
    var_name='semester',
    value_name='cost'
)

elec_melt.head()
```

[16]:

	product	consom	unit	tax	currency	geo\time	semester	cost
0	6000	4161901	KWH	I_TAX	EUR	AL	2019S1	:
1	6000	4161901	KWH	I_TAX	EUR	AT	2019S1	0.3796
2	6000	4161901	KWH	I_TAX	EUR	BA	2019S1	0.2090
3	6000	4161901	KWH	I_TAX	EUR	BE	2019S1	0.4742
4	6000	4161901	KWH	I_TAX	EUR	BG	2019S1	0.1014

[17]:
```
es_melt = elec_melt[elec_melt['geo\\time'] == 'ES']
es_melt.head()
```

[17]:
```
     product   consom unit    tax currency geo\time semester    cost
12      6000  4161901  KWH  I_TAX      EUR       ES   2019S1  0.6570
56      6000  4161901  KWH  I_TAX      NAT       ES   2019S1  0.6570
100     6000  4161901  KWH  I_TAX      PPS       ES   2019S1  0.7183
139     6000  4161901  KWH  X_TAX      EUR       ES   2019S1  0.5166
183     6000  4161901  KWH  X_TAX      NAT       ES   2019S1  0.5166
```

[18]:
```
renewable = pd.read_csv("energy_data/share_renewable.csv");
renewable.head()
```

[18]:
```
   TIME                            GEO                   NRG_BAL        UNIT  \
0  2008  European Union - 28 countries  Renewable energy sources  Percentage
1  2008                        Belgium  Renewable energy sources  Percentage
2  2008                       Bulgaria  Renewable energy sources  Percentage
3  2008                        Czechia  Renewable energy sources  Percentage
4  2008                        Denmark  Renewable energy sources  Percentage

    Value  Flag and Footnotes
0  11.325                 NaN
1   3.591                 NaN
2  10.492                 NaN
3   8.626                 NaN
4  18.564                 NaN
```

[19]:
```
es_renewable = renewable[renewable['GEO'] == 'Spain']
es_renewable.head()
```

[19]:
```
     TIME    GEO                   NRG_BAL        UNIT   Value  \
9    2008  Spain  Renewable energy sources  Percentage  10.737
46   2009  Spain  Renewable energy sources  Percentage  12.963
83   2010  Spain  Renewable energy sources  Percentage  13.810
120  2011  Spain  Renewable energy sources  Percentage  13.224
157  2012  Spain  Renewable energy sources  Percentage  14.286

     Flag and Footnotes
9                   NaN
46                  NaN
83                  NaN
120                 NaN
157                 NaN
```

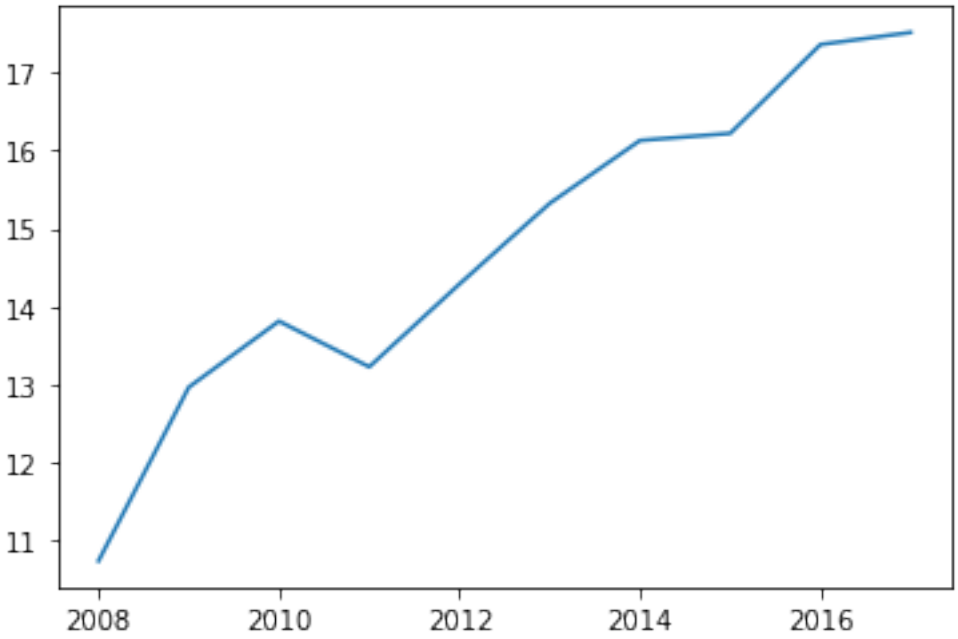
[20]:
```
renewable['GEO'].drop_duplicates()
```

[20]:
```
0                        European Union - 28 countries
1                                              Belgium
2                                             Bulgaria
3                                              Czechia
```

```
4                                                 Denmark
5      Germany (until 1990 former territory of the FRG)
6                                                 Estonia
7                                                 Ireland
8                                                  Greece
9                                                   Spain
10                                                 France
11                                                Croatia
12                                                  Italy
13                                                 Cyprus
14                                                 Latvia
15                                               Lithuania
16                                              Luxembourg
17                                                 Hungary
18                                                   Malta
19                                             Netherlands
20                                                 Austria
21                                                  Poland
22                                                Portugal
23                                                 Romania
24                                                Slovenia
25                                                Slovakia
26                                                 Finland
27                                                  Sweden
28                                          United Kingdom
29                                                 Iceland
30                                                  Norway
31                                              Montenegro
32                                          North Macedonia
33                                                 Albania
34                                                  Serbia
35                                                  Turkey
36    Kosovo (under United Nations Security Council ...
Name: GEO, dtype: object
```

[21]:
```
x = es_renewable.TIME
y = es_renewable.Value
```

[22]:
```
plt.plot(x, y)
plt.show()
```



[23]:
```
supply_consumption = pd.read_csv("energy_data/supply_consumption_renewables.
↪csv")
supply_consumption.head()
```

[23]:

	TIME	GEO	NRG_BAL	SIEC	UNIT	Value	Flag and Footnotes
0	2008	European Union - 28 countries	Inland consumption - calculated	Geothermal	Terajoule	235 262.865	NaN
1	2008	European Union - 28 countries	Inland consumption - calculated	Solar thermal	Terajoule	46 004.286	NaN
2	2008	European Union - 28 countries	Inland consumption - calculated	Primary solid biofuels	Terajoule	3 268 114.691	NaN
3	2008	European Union - 28 countries	Inland consumption - calculated	Biogases	Terajoule	280 015.181	NaN
4	2008	European Union - 28 countries	Inland consumption - calculated	Renewable municipal waste	Terajoule	303 233.213	NaN

[24]:
```
es_supply = supply_consumption[supply_consumption['GEO'] == 'Spain']
es_supply.head()
```

[24]:

	TIME	GEO	NRG_BAL	SIEC
50	2008	Spain	Inland consumption - calculated	Geothermal
51	2008	Spain	Inland consumption - calculated	Solar thermal
52	2008	Spain	Inland consumption - calculated	Primary solid biofuels
53	2008	Spain	Inland consumption - calculated	Biogases
54	2008	Spain	Inland consumption - calculated	Renewable municipal waste

```
         UNIT        Value  Flag and Footnotes
50  Terajoule      459.000                 NaN
51  Terajoule    5 378.234                 NaN
52  Terajoule  176 143.000                 NaN
53  Terajoule    8 660.000                 NaN
54  Terajoule   13 735.000                 NaN
```

[25]:
```
es_supply = es_supply.reset_index()
categories = es_supply.SIEC
```

[26]:
```
categories.head()
```

[26]:
```
0                  Geothermal
1               Solar thermal
2       Primary solid biofuels
3                    Biogases
4    Renewable municipal waste
Name: SIEC, dtype: object
```

[27]:
```
categories = categories.drop_duplicates()
```

[28]:
```
categories
```

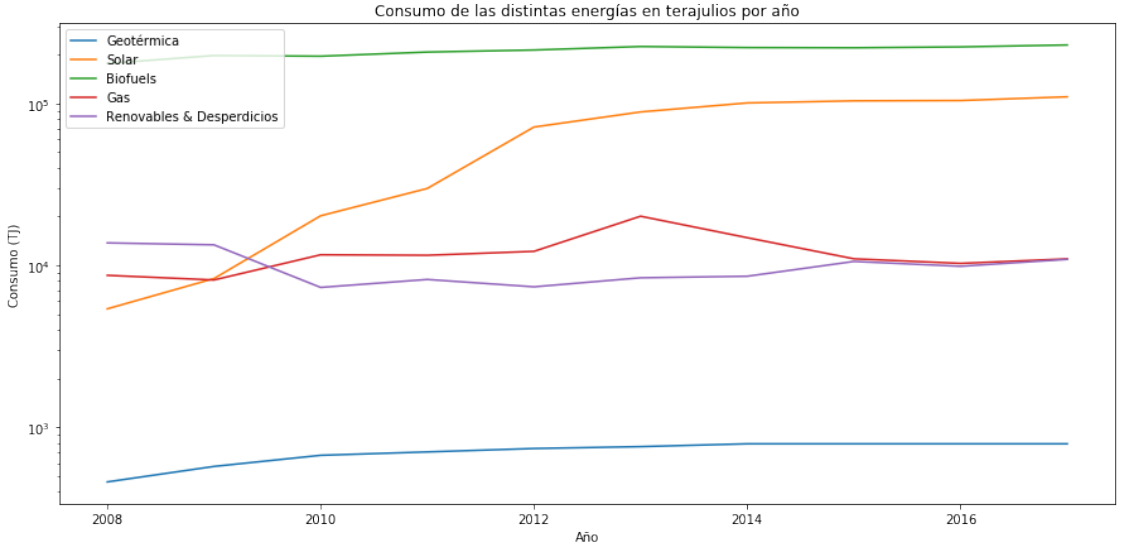
[28]:
```
0                  Geothermal
1               Solar thermal
2       Primary solid biofuels
3                    Biogases
4    Renewable municipal waste
Name: SIEC, dtype: object
```

[29]:
```
x = es_supply.TIME
x = x.drop_duplicates()
```

- Formateamos las celdas para que sean de tipo flotante:

[57]:
```
yGeo = es_supply[es_supply['SIEC'] == categories.get(0)].Value
yGeo = yGeo.astype(float)

ySol = es_supply[es_supply['SIEC'] == categories.get(1)].Value
ySol = (ySol.str.replace(" ", "")).astype(float)

yBio = es_supply[es_supply['SIEC'] == categories.get(2)].Value
yBio = (yBio.str.replace(" ", "")).astype(float)

yGas = es_supply[es_supply['SIEC'] == categories.get(3)].Value
yGas = (yGas.str.replace(" ", "")).astype(float)

yWaste = es_supply[es_supply['SIEC'] == categories.get(4)].Value
yWaste = (yWaste.str.replace(" ", "")).astype(float)
```

[31]:
```
plt.figure(figsize=(15, 7))
plt.plot(x, yGeo, x, ySol, x, yBio, x, yGas, x, yWaste)
plt.xlabel('Año')
plt.ylabel('Consumo (TJ)')
plt.title('Consumo de las distintas energías en terajulios por año')
plt.yscale('log')
plt.legend(['Geotérmica', 'Solar', 'Biofuels', 'Gas', 'Renovables &␣
↪Desperdicios'])
plt.show()
```



[32]:
```
energy_prod = pd.read_csv("energy_data/energy_productivity.csv", delimiter=';')
energy_supp = pd.read_csv("energy_data/energy_supply.csv", delimiter=';')
energy_cons = pd.read_csv("energy_data/consumo_todas_energias.csv", delimiter=';
↪')
```

[33]:
```
energy_prod.head()
```

[33]:

	Unnamed: 0	freq	unit	geo\TIME_PERIOD	2000	2001	2002	2003	\
0	AL	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	A	EUR_KGOE	AL	2.831	3.003	2.842	3.063	
2	NaN	A	PPS_KGOE	AL	5.658	6.059	5.864	6.314	
3	AT	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	A	EUR_KGOE	AT	8.691	8.341	8.430	8.086	

	2004	2005	...	2008	2009	2010	2011	2012	2013	2014	\
0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	2.945	3.091	...	3.922	3.995	4.180	4.109	4.625	3.961	4.048	
2	6.205	6.670	...	8.956	9.040	9.973	9.964	11.405	9.424	10.093	
3	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	8.152	8.100	...	8.881	9.007	8.609	9.121	9.239	9.028	9.384	

```
     2015    2016    2017
0     NaN     NaN     NaN
1   4.429   4.445   4.385
2  11.372  10.867  10.907
3     NaN     NaN     NaN
4   9.318   9.432   9.486

[5 rows x 22 columns]
```

[34]:
```
energy_prod_es = energy_prod[energy_prod['geo\\TIME_PERIOD'] == 'ES']
energy_prod_es
```

[34]:
```
   Unnamed: 0 freq      unit geo\TIME_PERIOD   2000   2001   2002   2003  \
37         NaN    A  EUR_KGOE              ES  6.649  6.711  6.717  6.703
38         NaN    A  PPS_KGOE              ES  5.864  6.040  6.271  6.273

     2004   2005  ...   2008   2009   2010   2011   2012   2013   2014   2015  \
37  6.620  6.706  ...  7.424  7.757  7.785  7.732  7.537  7.956  8.186  8.201
38  6.367  6.721  ...  8.050  8.198  8.195  8.171  8.204  8.700  9.133  9.344

     2016   2017
37  8.372  8.271
38  9.391  9.318

[2 rows x 22 columns]
```

[35]:
```
energy_prod_es = energy_prod_es.melt(
    id_vars=['Unnamed: 0', 'freq', 'unit', 'geo\\TIME_PERIOD'],
    var_name='YEAR',
    value_name='production'
)

energy_prod_es.head()
```

[35]:
```
  Unnamed: 0 freq      unit geo\TIME_PERIOD  YEAR production
0        NaN    A  EUR_KGOE              ES  2000      6.649
1        NaN    A  PPS_KGOE              ES  2000      5.864
2        NaN    A  EUR_KGOE              ES  2001      6.711
3        NaN    A  PPS_KGOE              ES  2001      6.040
4        NaN    A  EUR_KGOE              ES  2002      6.717
```

[36]:
```
energy_prod_es.production = (energy_prod_es.production).str.replace(".", "")
energy_prod_es.production[energy_prod_es.production == 'NaN'] = 1
energy_prod_es.production = pd.to_numeric(energy_prod_es.production)
```

[37]:
```
share_renewable = pd.read_csv("energy_data/share_renewable.csv")
```

[38]:
```
share_renewable.head()
```

[38]:
```
   TIME                            GEO                  NRG_BAL        UNIT  \
0  2008  European Union - 28 countries  Renewable energy sources  Percentage
1  2008                         Belgium  Renewable energy sources  Percentage
2  2008                        Bulgaria  Renewable energy sources  Percentage
3  2008                         Czechia  Renewable energy sources  Percentage
4  2008                         Denmark  Renewable energy sources  Percentage

    Value  Flag and Footnotes
0  11.325                 NaN
1   3.591                 NaN
2  10.492                 NaN
3   8.626                 NaN
4  18.564                 NaN
```

[39]:
```
share_renewable.Value = pd.to_numeric(share_renewable.Value)
```

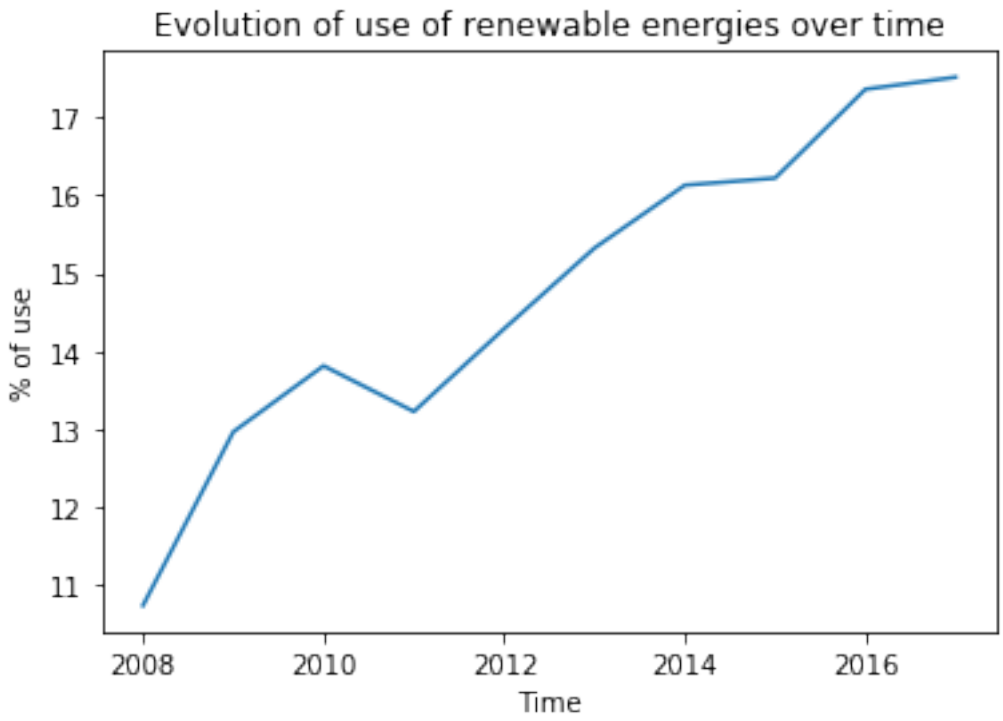
[40]:
```
share_renewable_es = share_renewable[share_renewable['GEO'] == 'Spain']
```

[41]:
```
share_renewable_es
```

[41]:
```
     TIME    GEO                  NRG_BAL        UNIT   Value  \
9    2008  Spain  Renewable energy sources  Percentage  10.737
46   2009  Spain  Renewable energy sources  Percentage  12.963
83   2010  Spain  Renewable energy sources  Percentage  13.810
120  2011  Spain  Renewable energy sources  Percentage  13.224
157  2012  Spain  Renewable energy sources  Percentage  14.286
194  2013  Spain  Renewable energy sources  Percentage  15.320
231  2014  Spain  Renewable energy sources  Percentage  16.126
268  2015  Spain  Renewable energy sources  Percentage  16.217
305  2016  Spain  Renewable energy sources  Percentage  17.356
342  2017  Spain  Renewable energy sources  Percentage  17.511

     Flag and Footnotes
9                   NaN
46                  NaN
83                  NaN
120                 NaN
157                 NaN
194                 NaN
231                 NaN
268                 NaN
305                 NaN
342                 NaN
```

[42]:
```
plt.plot(share_renewable_es.TIME, share_renewable_es.Value)
plt.xlabel('Time')
plt.ylabel('% of use')
plt.title('Evolution of use of renewable energies over time')
```

[42]:
```
Text(0.5, 1.0, 'Evolution of use of renewable energies over time')
```



2 /////////////// COMIENZO DE LA LIBRETA ///////////////

Importamos las librerías necesarias...

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
electricity_prices_household = pd.read_csv("energy_data/
 ↪electricity_prices_household.csv", delimiter=';')
gas_prices_household = pd.read_csv("energy_data/gas_prices_household.csv",␣
 ↪delimiter=';')
consumo_energias_sector = pd.read_csv("energy_data/consumo_energias_sector.
 ↪csv", delimiter=';')
energy_supply = pd.read_csv("energy_data/energy_supply.csv", delimiter=';')
energy_productivity = pd.read_csv("energy_data/energy_productivity.csv",␣
 ↪delimiter=';')
```

Definición de funciones útiles

[3]:
```
def consulta_por_valor_columna(dataframe, valor, nombre_columna):
    return dataframe[dataframe[nombre_columna] == valor]
```

[82]:
```
def crear_grafica(x_data, y_data, titulo, x_label, y_label):
    plt.figure(figsize=(20, 10))
    plt.plot(x_data, y_data)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(titulo)
```

[5]:
```
def castear_a_float(input_list):
    '''Recibe una lista de valores y devuelve otra con los valores␣
 ↪interpretados como flotantes.
    Esta función NO edita la columna, devuelve una nueva. La columna del␣
 ↪dataframe original deberá ser sobreescrita pertinentemente.
    '''
    return input_list.apply(float)
```

[6]:
```
def string_replace_columna(columna, valor_a_buscar, valor_sustituto):
    '''Busca espacios en la columna y los elimina
    Esta función NO edita la columna, devuelve una nueva. La columna del␣
 ↪dataframe original deberá ser sobreescrita pertinentemente.
    '''
    return columna.str.replace(valor_a_buscar, valor_sustituto)
```

[7]:
```
def eliminar_valores_invalidos(dataframe, columna, valor_invalido, valor_nuevo):
    '''Sustituye las instancias de valor_invalido por las de valor nuevo en la␣
 ↪columna. Por ejemplo, para sustituir \':\' por -1
    Es recomendable pasar el valor nuevo como tipo igual al que tiene el data␣
 ↪frame, para que luego los casteos no se hagan complicados
    '''
    dataframe.loc[dataframe[columna] == valor_invalido, columna] = valor_nuevo
```

[8]:
```
electricity_prices_household = electricity_prices_household.melt(
    id_vars=['product', 'consom', 'unit', 'tax', 'currency', 'geo\\time'],
    var_name='semester',
    value_name='cost'
)
```

Ejemplo de uso de las funciones:

[9]:
```
eliminar_valores_invalidos(electricity_prices_household, 'cost', ': ', '-1')
eliminar_valores_invalidos(electricity_prices_household, 'cost', ': u', '-1')
electricity_prices_household.cost =␣
 ↪string_replace_columna(electricity_prices_household.cost, 'u', "")
electricity_prices_household.cost =␣
 ↪string_replace_columna(electricity_prices_household.cost, 'p', "")
```

[10]:
```
electricity_prices_household.cost =␣
 ↪castear_a_float(electricity_prices_household.cost)
```

[11]:
```
electricity_prices_household.cost.head()
```

[11]:
```
0   -1.0000
1    0.3796
2    0.2090
3    0.4742
4    0.1014
Name: cost, dtype: float64
```

Podemos ver que ahora el tipo de esa columna es flotante, no *object* o *str*

[12]:
```
supply_consumption_renewables = pd.read_csv("energy_data/
 ↪supply_consumption_renewables.csv")
```

[13]:
```
supply_consumption_renewables.head()
```

[13]:
```
   TIME                            GEO                          NRG_BAL  \
0  2008  European Union - 28 countries  Inland consumption - calculated
1  2008  European Union - 28 countries  Inland consumption - calculated
2  2008  European Union - 28 countries  Inland consumption - calculated
3  2008  European Union - 28 countries  Inland consumption - calculated
4  2008  European Union - 28 countries  Inland consumption - calculated

                        SIEC       UNIT          Value  Flag and Footnotes
0                Geothermal  Terajoule    235 262.865                 NaN
1             Solar thermal  Terajoule     46 004.286                 NaN
2    Primary solid biofuels  Terajoule  3 268 114.691                 NaN
3                  Biogases  Terajoule    280 015.181                 NaN
4  Renewable municipal waste  Terajoule    303 233.213                 NaN
```

[14]:
```
supply_consumption_renewables.Value =␣
 ↪string_replace_columna(supply_consumption_renewables.Value, " ", "")
```

[15]:
```
eliminar_valores_invalidos(supply_consumption_renewables, 'Value', ':', '-1')

eliminar_valores_invalidos(supply_consumption_renewables, 'Value', ': u', '-1')

supply_consumption_renewables.Value =␣
 ↪string_replace_columna(supply_consumption_renewables.Value, 'p', "")

supply_consumption_renewables.Value =␣
 ↪castear_a_float(supply_consumption_renewables.Value)
```

[16]:
```
supply_consumption_renewables.Value.head()
```

[16]:
```
0     235262.865
1      46004.286
2    3268114.691
3     280015.181
4     303233.213
Name: Value, dtype: float64
```

2.1 Comenzamos con el proceso de selección de los datos que nos interesan.

Vamos a seleccionar en las tablas los datos correspondientes a España. Y, si hiciera falta, seleccionar la categoría más relevante, como la moneda la unidad de energía u otros.

Importamos las librerías necesarias y cargamos las tablas...

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
electricity_prices_household = pd.read_csv("energy_data/
 ↪electricity_prices_household.csv", delimiter=';')
gas_prices_household = pd.read_csv("energy_data/gas_prices_household.csv",␣
 ↪delimiter=';')
```

Definición de funciones útiles

[3]:
```
def consulta_por_valor_columna(dataframe, valor, nombre_columna):
    return dataframe[dataframe[nombre_columna] == valor]
```

[4]:
```
def crear_grafica(x_data, y_data, titulo, x_label, y_label):
    plt.figure(figsize=(20, 10))
    plt.plot(x_data, y_data)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(titulo)
```

[5]:
```
def castear_a_float(input_list):
    '''Recibe una lista de valores y devuelve otra con los valores␣
 ↪interpretados como flotantes.
    Esta función NO edita la columna, devuelve una nueva. La columna del␣
 ↪dataframe original deberá ser sobreescrita pertinentemente.
    '''
    return input_list.apply(float)
```

[6]:
```
def string_replace_columna(columna, valor_a_buscar, valor_sustituto):
    '''Busca espacios en la columna y los elimina
    Esta función NO edita la columna, devuelve una nueva. La columna del␣
 ↪dataframe original deberá ser sobreescrita pertinentemente.
    '''
    return columna.str.replace(valor_a_buscar, valor_sustituto)
```

[7]:
```
def eliminar_valores_invalidos(dataframe, columna, valor_invalido, valor_nuevo):
    '''Sustituye las instancias de valor_invalido por las de valor nuevo en la␣
 ↪columna. Por ejemplo, para sustituir \':\' por -1
    Es recomendable pasar el valor nuevo como tipo igual al que tiene el data␣
 ↪frame, para que luego los casteos no se hagan complicados
    '''
    dataframe.loc[dataframe[columna] == valor_invalido, columna] = valor_nuevo
```

Seleccionamos las columnas referentes a nuestro país...

[8]:
```
electricity_prices_household_es =␣
 ↪consulta_por_valor_columna(electricity_prices_household, 'ES', 'geo\\time')
gas_prices_household_es = consulta_por_valor_columna(gas_prices_household,␣
 ↪'ES', 'geo\\time')
```

Seleccionamos la moneda y las unidades relevantes...

[9]:
```
electricity_prices_household_es_eur =␣
 ↪consulta_por_valor_columna(electricity_prices_household_es, 'EUR',␣
 ↪'currency')
electricity_prices_household_es_eur_xtax =␣
 ↪consulta_por_valor_columna(electricity_prices_household_es_eur, 'X_TAX',␣
 ↪'tax')


gas_prices_household_es_eur =␣
 ↪consulta_por_valor_columna(gas_prices_household_es, 'EUR', 'currency')
gas_prices_household_es_eur_xtax =␣
 ↪consulta_por_valor_columna(gas_prices_household_es_eur, 'X_TAX', 'tax')
```

Hemos acabado el proceso de selección. Ahora tenemos 3 dataframes mucho más concretos y capaces de aportar bastante información.

Vamos a crear una gráfica con ellos, y a sacar algunas conclusiones de esta primera fase. Para ello, primero debemos tratar los datos para convertir las columnas que representaremos en formato numérico.

[10]:
```
electricity_prices_household_es_eur_xtax.head()
```

[10]:
```
      product   consom unit    tax currency geo\time  2019S1  2018S2  2018S1  \
139      6000  4161901  KWH  X_TAX      EUR       ES  0.5166  0.4485  0.4680
520      6000  4161902  KWH  X_TAX      EUR       ES  0.2355  0.2393  0.2303
901      6000  4161903  KWH  X_TAX      EUR       ES  0.1889  0.1947  0.1873
1282     6000  4161904  KWH  X_TAX      EUR       ES  0.1608  0.1697  0.1513
1663     6000  4161905  KWH  X_TAX      EUR       ES  0.1409  0.1414  0.1236

      2017S2  ...  2011S2  2011S1  2010S2  2010S1  2009S2  2009S1  2008S2  \
139   0.4479  ...  0.3044  0.2890  0.2926  0.3174  0.2611  0.2540  0.2622
520   0.2113  ...  0.1870  0.1768  0.1681  0.1622  0.1571  0.1459  0.1475
901   0.1712  ...  0.1684  0.1597  0.1492  0.1417  0.1381  0.1294  0.1277
1282  0.1482  ...  0.1563  0.1426  0.1363  0.1271  0.1266  0.1203  0.1219
1663  0.1311  ...  0.1444  0.1251  0.1244  0.1174  0.1153  0.1163  0.1148

      2008S1  2007S2 2007S1
139   0.2455  0.2424      :
520   0.1299  0.1332      :
```

```
901   0.1124  0.1152     :
1282  0.1021  0.1058     :
1663  0.0981  0.0976     :

[5 rows x 31 columns]
```

El formato en el que se presenta la tabla es bastante inconveniente para procesar los datos. Vamos a unir todas las columnas referentes a los años en solo dos, una con los semestres y otra con su valor.

[11]:
```
electricity_prices_household_es_eur_xtax = electricity_prices_household_es_eur_xtax.melt(
    id_vars=['product', 'consom', 'unit', 'tax', 'currency', 'geo\\time'],
    var_name='Semestres',
    value_name='Precio'
)

electricity_prices_household_es_eur_xtax.head()
```

[11]:

	product	consom	unit	tax	currency	geo\time	Semestres	Precio
0	6000	4161901	KWH	X_TAX	EUR	ES	2019S1	0.5166
1	6000	4161902	KWH	X_TAX	EUR	ES	2019S1	0.2355
2	6000	4161903	KWH	X_TAX	EUR	ES	2019S1	0.1889
3	6000	4161904	KWH	X_TAX	EUR	ES	2019S1	0.1608
4	6000	4161905	KWH	X_TAX	EUR	ES	2019S1	0.1409

[12]:
```
gas_prices_household_es_eur_xtax = gas_prices_household_es_eur_xtax.melt(
    id_vars=['product', 'consom', 'unit', 'tax', 'currency', 'geo\\time'],
    var_name='Semestres',
    value_name='Precio'
)

gas_prices_household_es_eur_xtax.head()
```

[12]:

	product	consom	unit	tax	currency	geo\time	Semestres	Precio
0	4100	4141901	GJ_GCV	X_TAX	EUR	ES	2019S1	:
1	4100	4141901	KWH	X_TAX	EUR	ES	2019S1	:
2	4100	4141902	GJ_GCV	X_TAX	EUR	ES	2019S1	:
3	4100	4141902	KWH	X_TAX	EUR	ES	2019S1	:
4	4100	4141903	GJ_GCV	X_TAX	EUR	ES	2019S1	:

Ahora, vamos a convertir las columnas pertinentes a flotantes.

Primero debemos eliminar los valores inválidos:

[13]:
```
eliminar_valores_invalidos(electricity_prices_household_es_eur_xtax, 'Precio', ": ", "-1")
eliminar_valores_invalidos(electricity_prices_household_es_eur_xtax, 'Precio', ": u", "-1")
```

```
electricity_prices_household_es_eur_xtax.Precio =␣
 ↪string_replace_columna(electricity_prices_household_es_eur_xtax.Precio, 'p',␣
 ↪"")
electricity_prices_household_es_eur_xtax.Precio =␣
 ↪string_replace_columna(electricity_prices_household_es_eur_xtax.Precio, " ",␣
 ↪"")
```

Ahora que hemos limpiado la columna precio, podemos convertirla a `float`.

[14]:
```
electricity_prices_household_es_eur_xtax.Precio =␣
 ↪castear_a_float(electricity_prices_household_es_eur_xtax.Precio)

electricity_prices_household_es_eur_xtax.Precio.head()
```

[14]:
```
0    0.5166
1    0.2355
2    0.1889
3    0.1608
4    0.1409
Name: Precio, dtype: float64
```

2.1.1 ąGenial! Procedamos con las demás.

[15]:
```
eliminar_valores_invalidos(gas_prices_household_es_eur_xtax, 'Precio', ": ",␣
 ↪"-1")
eliminar_valores_invalidos(gas_prices_household_es_eur_xtax, 'Precio', ": u",␣
 ↪"-1")

gas_prices_household_es_eur_xtax.Precio =␣
 ↪string_replace_columna(gas_prices_household_es_eur_xtax.Precio, 'p', "")
gas_prices_household_es_eur_xtax.Precio =␣
 ↪string_replace_columna(gas_prices_household_es_eur_xtax.Precio, " ", "")
```

[16]:
```
gas_prices_household_es_eur_xtax.Precio =␣
 ↪castear_a_float(gas_prices_household_es_eur_xtax.Precio)

gas_prices_household_es_eur_xtax.Precio.head()
```

[16]:
```
0   -1.0
1   -1.0
2   -1.0
3   -1.0
4   -1.0
Name: Precio, dtype: float64
```

2.1.2 Ahora que tenemos los datos bien formateados, ąvamos a representarlos!

Para representar los datos del consumo de electricidad, primero debemos agruparlos por las diferentes categorías que presentan, como el precio según la tarifa. Primero, vamos a eliminar

la columna de 2007S1, ya que, durante el proceso de análisis, nos hemos dado cuenta de que no posee ningún valor.

[17]:
```
electricity_prices_household_es_eur_xtax = electricity_prices_household_es_eur_xtax[electricity_prices_household_es_eur_xtax['Semestre
!= '2007S1']
```

De las fuentes de datos, sabemos que los diferentes valores de la columna Consom corresponden a las diferentes facturas de electricidad. Vamos a separarlas en diferentes variables.

[18]:
```
less_1k = (consulta_por_valor_columna(electricity_prices_household_es_eur_xtax,
4161901, 'consom')).Precio
from_1k_to_2k =
(consulta_por_valor_columna(electricity_prices_household_es_eur_xtax,
4161902, 'consom')).Precio
from_2k_to_5k =
(consulta_por_valor_columna(electricity_prices_household_es_eur_xtax,
4161903, 'consom')).Precio
from_5k_to_15k =
(consulta_por_valor_columna(electricity_prices_household_es_eur_xtax,
4161904, 'consom')).Precio
over_15k =
(consulta_por_valor_columna(electricity_prices_household_es_eur_xtax,
4161905, 'consom')).Precio
```

[19]:
```
semestres_elec = (electricity_prices_household_es_eur_xtax.Semestres).
drop_duplicates()
```

Como los datos están dispuestos del revés cronológicamente, vamos a darles la vuelta.

[20]:
```
semestres_elec = semestres_elec.iloc[::-1]
less_1k = less_1k.iloc[::-1]
from_1k_to_2k = from_1k_to_2k.iloc[::-1]
from_2k_to_5k = from_2k_to_5k.iloc[::-1]
from_5k_to_15k = from_5k_to_15k.iloc[::-1]
over_15k = over_15k.iloc[::-1]
```

[21]:
```
avg_elec = np.array([less_1k, from_1k_to_2k, from_2k_to_5k, from_5k_to_15k,
over_15k])
avg_elec = np.mean(avg_elec, axis=0)
avg_elec = pd.Series(avg_elec)
```

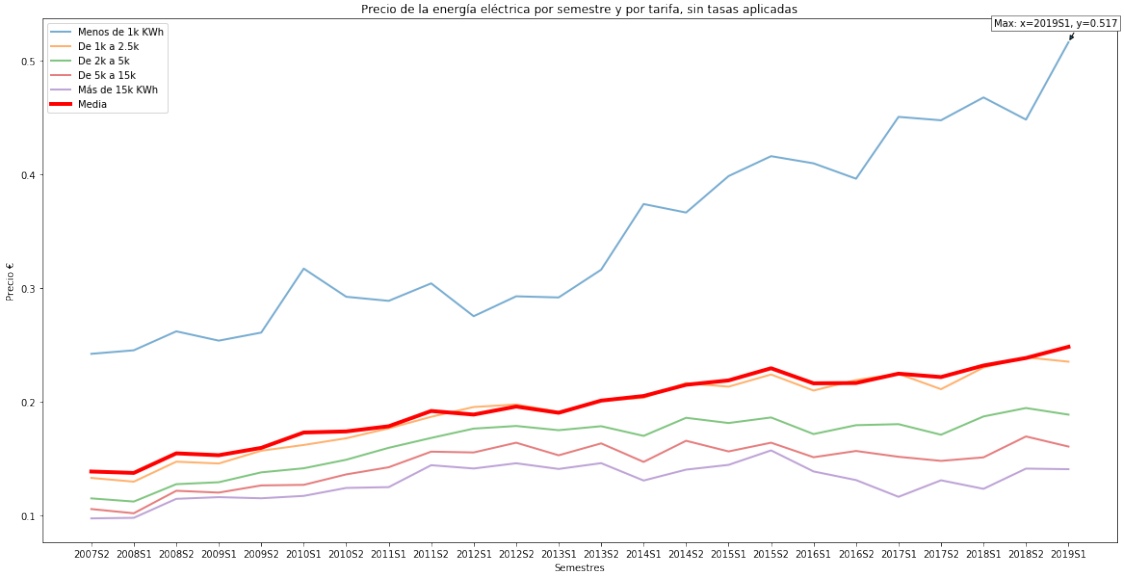
Definimos una función que nos permite indicar cuál es el máximo

[22]:
```
def annot_max(x, y, ax=None, xPos=1, yPos=1):
    xmax = x[pd.Series.idxmax(y)]
    ymax = y.max()
    text = "Max: x={0:s}, y={1:.3f}".format(xmax, float(ymax))
    if not ax:
        ax=plt.gca()
```

```
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.52)
    arrowprops =␣
 ↪dict(arrowstyle="-|>",connectionstyle="angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha="right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=(xPos, yPos), **kw)
```

[23]:
```
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111)

ax.plot(semestres_elec, less_1k, semestres_elec, from_1k_to_2k, semestres_elec,␣
 ↪from_2k_to_5k, semestres_elec, from_5k_to_15k, semestres_elec, over_15k,␣
 ↪linewidth=2, alpha=0.6)
ax.plot(semestres_elec, avg_elec, 'r-', linewidth=4)

annot_max(semestres_elec, less_1k)

plt.title('Precio de la energía eléctrica por semestre y por tarifa, sin tasas␣
 ↪aplicadas')
plt.legend(['Menos de 1k KWh', 'De 1k a 2.5k', 'De 2k a 5k', 'De 5k a 15k',␣
 ↪'Más de 15k KWh', 'Media'])
plt.xlabel('Semestres')
plt.ylabel('Precio ')
```

[23]:
```
Text(0, 0.5, 'Precio ')
```



2.1.3 Ahora, los datos del precio del gas.

[24]:
```
gas_prices_household_es_eur_xtax.head()
```

[24]:

	product	consom	unit	tax	currency	geo\time	Semestres	Precio
0	4100	4141901	GJ_GCV	X_TAX	EUR	ES	2019S1	-1.0
1	4100	4141901	KWH	X_TAX	EUR	ES	2019S1	-1.0
2	4100	4141902	GJ_GCV	X_TAX	EUR	ES	2019S1	-1.0
3	4100	4141902	KWH	X_TAX	EUR	ES	2019S1	-1.0
4	4100	4141903	GJ_GCV	X_TAX	EUR	ES	2019S1	-1.0

Nos quedamos con los GJ (Giga Julios), que es la unidad que se utiliza como referencia.

[25]:
```
gas_prices_household_es_eur_xtax_gj = consulta_por_valor_columna(gas_prices_household_es_eur_xtax, 'GJ_GCV','unit')
```

[26]:
```
gas_prices_household_es_eur_xtax_gj = gas_prices_household_es_eur_xtax_gj[(gas_prices_household_es_eur_xtax_gj['Semestres'] != '2019S1') & (gas_prices_household_es_eur_xtax_gj['Semestres'] != '2007S1')]
```

[27]:
```
semestres_gas = (gas_prices_household_es_eur_xtax_gj.Semestres).drop_duplicates()
```

[28]:
```
gas_prices_household_es_eur_xtax_gj.head()
```

[28]:

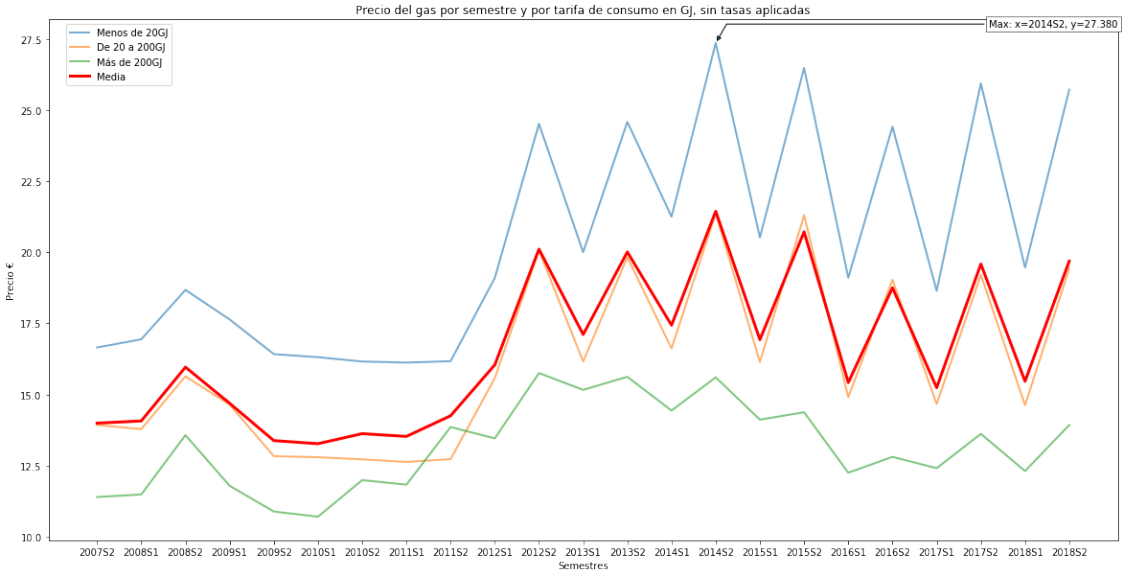
	product	consom	unit	tax	currency	geo\time	Semestres	Precio
6	4100	4141901	GJ_GCV	X_TAX	EUR	ES	2018S2	25.7334
8	4100	4141902	GJ_GCV	X_TAX	EUR	ES	2018S2	19.4281
10	4100	4141903	GJ_GCV	X_TAX	EUR	ES	2018S2	13.9205
12	4100	4141901	GJ_GCV	X_TAX	EUR	ES	2018S1	19.4699
14	4100	4141902	GJ_GCV	X_TAX	EUR	ES	2018S1	14.6187

[29]:
```
less_20GJ = (consulta_por_valor_columna(gas_prices_household_es_eur_xtax_gj, 4141901, 'consom')).Precio
from_20GJ_to_200GJ = (consulta_por_valor_columna(gas_prices_household_es_eur_xtax_gj, 4141902, 'consom')).Precio
over_200GJ = (consulta_por_valor_columna(gas_prices_household_es_eur_xtax_gj, 4141903, 'consom')).Precio
```

[30]:
```
less_20GJ = less_20GJ.iloc[::-1]
from_20GJ_to_200GJ = from_20GJ_to_200GJ.iloc[::-1]
over_200GJ = over_200GJ.iloc[::-1]
semestres_gas = semestres_gas.iloc[::-1]
```

[31]:
```
averages = np.array([less_20GJ, from_20GJ_to_200GJ, over_200GJ])
averages = (np.mean(averages, axis=0))
averages = pd.Series(averages)
```

[32]:
```
fig = plt.figure(figsize=(20, 10))
ax  = fig.add_subplot(111)

ax.plot(semestres_gas, less_20GJ, semestres_gas, from_20GJ_to_200GJ,␣
 ↪semestres_gas, over_200GJ, linewidth=2, alpha=0.6)
ax.plot(semestres_gas, averages, 'r-', linewidth=3)

annot_max(semestres_gas, less_20GJ)

plt.title('Precio del gas por semestre y por tarifa de consumo en GJ, sin tasas␣
 ↪aplicadas')
leg = plt.legend(['Menos de 20GJ', 'De 20 a 200GJ', 'Más de 200GJ', 'Media'],␣
 ↪loc='best')
plt.xlabel('Semestres')
plt.ylabel('Precio ')

plt.draw()

bb = leg.get_bbox_to_anchor().inverse_transformed(ax.transAxes)

# Change to location of the legend.
xOffset = -0.88
bb.x0 += xOffset
bb.x1 += xOffset
leg.set_bbox_to_anchor(bb, transform = ax.transAxes)


# Update the plot
plt.show()
```



2.1.4 Como vemos, el precio de ambas energías ha subido progresivamente en los últimos años.

También es destacable que las tarifas más caras por unidad son aquellas que contratan menos energía. Tiene sentido, siempre que se compra de algo, cuanto más te lleves, más barato te sale por unidad.

[33]:
```
print("Precio en 2008 / Precio en 2019: {:.2f}".format(less_20GJ.iloc[-1] /␣
↪less_20GJ.iloc[0]))
```

```
Precio en 2008 / Precio en 2019: 1.55
```

El precio del gas ha subido en un 1.5, es decir, un 50% más que en 2008

[34]:
```
print("Precio en 2008 / Precio en 2019: {:.2f}".format(less_1k.iloc[-1] /␣
↪less_1k.iloc[0]))
```

```
Precio en 2008 / Precio en 2019: 2.13
```

El de la electricidad ha aumentado en más de un 100%, es decir, prácticamente se ha duplicado desde 2007

2.2 Hemos analizado el coste de las energías. Vamos a ver ahora cuánto se consume de cada una.

[35]:
```
consum = pd.read_csv('energy_data/consumo_energias_sector.csv', delimiter=';')
```

[36]:
```
consum.head()
```

[36]:

	freq	nrg_bal	siec	unit	geo\TIME_PERIOD	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
0	A	FC_E	TOTAL	KTOE	AL	1685.582	1672.393	1740.442	1840.737	1897.918	1953.468	1790.583	1961.866	2058.559	1963.484	1883.404	2071.754
1	A	FC_IND_E	TOTAL	KTOE	AL	267.587	258.050	228.550	282.307	354.766	379.510	276.800	281.446	372.753	302.114	283.644	398.286
2	A	FC_OTH_CP_E	TOTAL	KTOE	AL	145.967	100.258	161.636	173.753	159.469	163.473	157.272	190.984	179.239	196.902	175.590	213.657
3	A	FC_OTH_HH_E	TOTAL	KTOE	AL	451.947	423.106	445.281	487.171	489.039	498.280	509.086	580.003	556.243	532.287	486.135	492.580
4	A	FC_TRA_E	TOTAL	KTOE	AL	636.304	689.281	745.409	754.562	732.789	768.010	737.022	804.074	833.978			

```
4    829.298    832.543    831.899
```

Como sabemos de la documentación de las tablas, para los usuarios finales, el valor es `FC_OTH_HH_E`. Adelantamos el proceso y tomamos también los datos referentes a España.

[37]:
```
consum_hh_es = consum[(consum['nrg_bal'] == 'FC_OTH_HH_E') &␣
 ↪(consum['geo\\TIME_PERIOD'] == 'ES')]
```

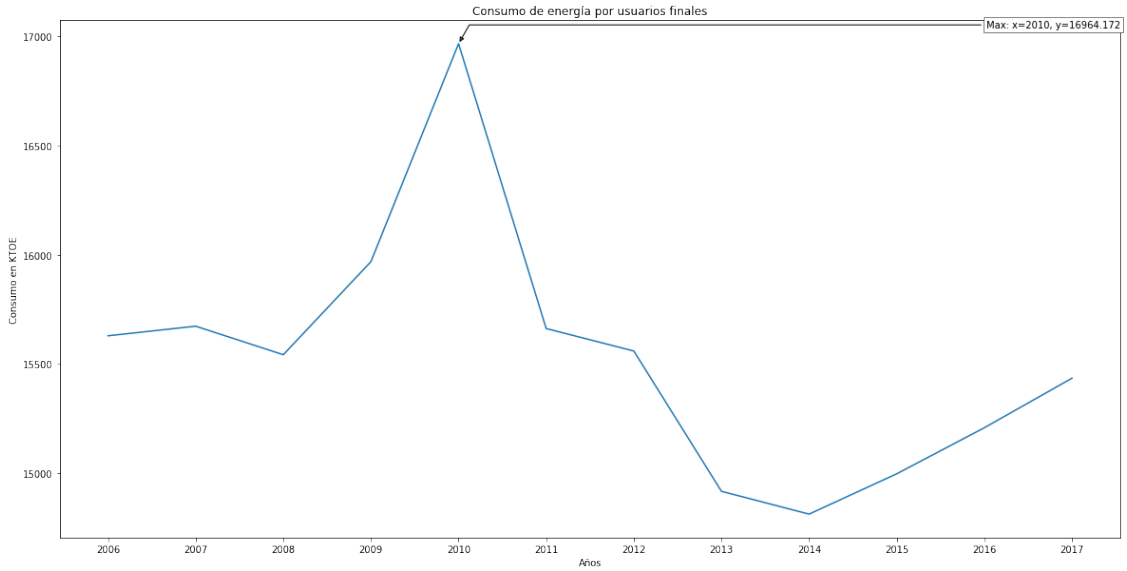
[38]:
```
consum_hh_es
```

[38]:
```
   freq      nrg_bal   siec  unit geo\TIME_PERIOD       2006       2007  \
63    A  FC_OTH_HH_E  TOTAL  KTOE              ES  15629.050  15672.903

         2008       2009       2010       2011       2012       2013  \
63  15542.505  15967.814  16964.172  15662.135  15559.386  14918.154

         2014       2015       2016       2017
63  14814.004  14998.213  15208.781  15435.162
```

[39]:
```
consum_hh_es = consum_hh_es.melt(
    id_vars=['freq', 'nrg_bal', 'siec', 'unit', 'geo\\TIME_PERIOD'],
    var_name='Años',
    value_name='Consumo'
)
```

[40]:
```
consum_hh_es.Consumo = castear_a_float(consum_hh_es.Consumo)
```

[41]:
```
title = 'Consumo de energía por usuarios finales'
ylabel = 'Consumo en KTOE'
xlabel = 'Años'
crear_grafica(consum_hh_es.Años, consum_hh_es.Consumo, title, xlabel, ylabel)
annot_max(consum_hh_es.Años, consum_hh_es.Consumo)
```



[42]:
```
consum_total = pd.read_csv("energy_data/consumo_todas_energias.csv",␣
↪delimiter=';')
```

[43]:
```
consum_total.head()
```

[43]:
```
  freq nrg_bal             siec  unit geo\TIME_PERIOD     2006     2007  \
0    A     FC_E  C0000X0350-0370  KTOE               AL   12.695   12.695
1    A     FC_E       C0350-0370  KTOE               AL        0        0
2    A     FC_E            E7000  KTOE               AL  298.968  315.563
3    A     FC_E            G3000  KTOE               AL        0    1.806
4    A     FC_E            H8000  KTOE               AL        0        0

      2008     2009     2010     2011     2012     2013     2014     2015  \
0   16.457  100.500  110.900  138.974   77.225   67.005   85.149   95.081
1        0        0        0        0        0        0        0        0
2  397.077  462.339  487.790  486.414  495.271  592.777  560.877  507.051
3        0      817      709    1.548    3.332    5.589    6.836   11.156
4        0        0        0        0        0        0        0        0

      2016     2017
0   50.729  115.975
1        0        0
2  472.829  512.164
3   10.035   12.500
4        0        0
```

Tomamos el consumo total de electricidad en España...

[44]:
```
consum_elect = consulta_por_valor_columna(consum_total, 'E7000', 'siec')
consum_elect_es = consulta_por_valor_columna(consum_elect, 'ES',␣
↪'geo\\TIME_PERIOD')
consum_elect_es.head()
```

[44]:
```
     freq nrg_bal   siec  unit geo\TIME_PERIOD       2006       2007  \
145     A    FC_E  E7000  KTOE               ES  21162.941  21563.629

           2008       2009       2010       2011       2012       2013  \
145  21934.136  20617.197  21049.183  20938.005  20657.610  19783.921

           2014       2015       2016       2017
145  19509.630  19951.677  19992.691  20169.475
```

Y el consumo de gas.

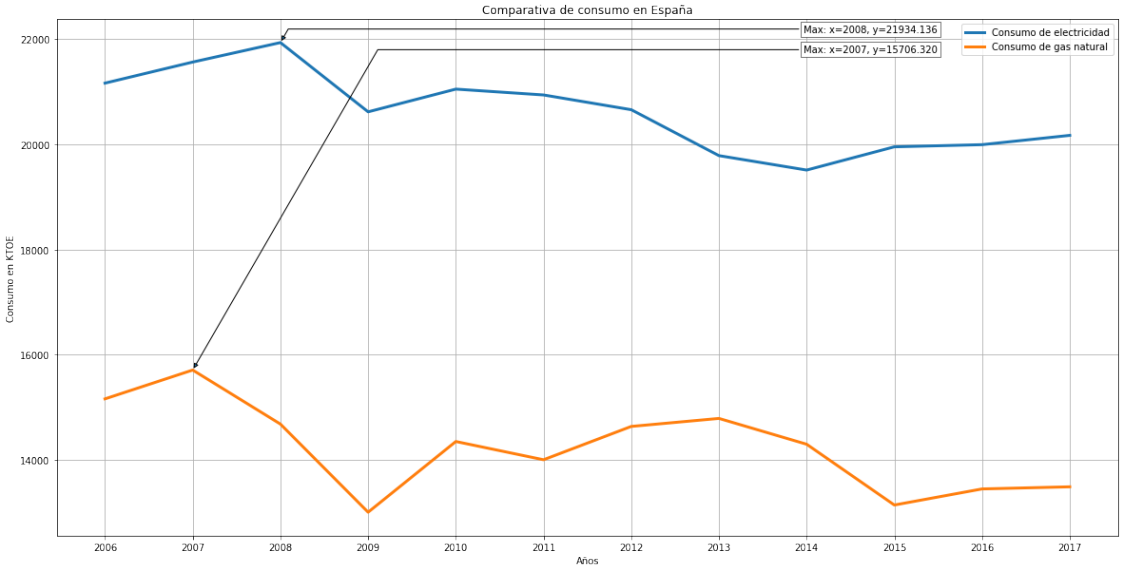
[45]:
```
consum_gas = consulta_por_valor_columna(consum_total, 'G3000', 'siec')
consum_gas_es = consulta_por_valor_columna(consum_gas, 'ES', 'geo\\TIME_PERIOD')
consum_gas_es.head()
```

[45]:
```
    freq nrg_bal   siec  unit geo\TIME_PERIOD       2006       2007  \
146    A    FC_E  G3000  KTOE              ES  15158.491  15706.320

          2008       2009       2010       2011       2012       2013  \
146  14678.848  13002.580  14347.120  14000.516  14633.964  14786.199

          2014       2015       2016       2017
146  14295.314  13139.402  13445.421  13485.834
```

[46]:
```
consum_elect_es = consum_elect_es.melt(
    id_vars=['freq', 'nrg_bal', 'siec', 'unit', 'geo\\TIME_PERIOD'],
    var_name='Años',
    value_name='Consumo'
)

consum_gas_es = consum_gas_es.melt(
    id_vars=['freq', 'nrg_bal', 'siec', 'unit', 'geo\\TIME_PERIOD'],
    var_name='Años',
    value_name='Consumo'
)
```

[47]:
```
consum_elect_es.Consumo = castear_a_float(consum_elect_es.Consumo)
consum_gas_es.Consumo = castear_a_float(consum_gas_es.Consumo)
```

[48]:
```
plt.figure(figsize=(20, 10))
plt.plot(consum_elect_es.Años, consum_elect_es.Consumo, consum_gas_es.Años,␣
 ↪consum_gas_es.Consumo, linewidth=3)
plt.title('Comparativa de consumo en España')
plt.xlabel('Años')
plt.ylabel('Consumo en KTOE')

plt.grid(True)
plt.legend(['Consumo de electricidad', 'Consumo de gas natural'])


annot_max(consum_elect_es.Años, consum_elect_es.Consumo, xPos=0.83, yPos=0.99)
annot_max(consum_gas_es.Años, consum_gas_es.Consumo, xPos=0.83, yPos=0.95)
```


Comparativa de consumo en España
Max: x=2008, y=21934.136
Max: x=2007, y=15706.320
Consumo de electricidad
Consumo de gas natural
Consumo en KTOE
Años
22000
20000
18000
16000
14000
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

[ ]: