**EDA - Exploratory Data Analysis datos energéticos España.**

**Hourly energy demand generation and weather**

**Electrical demand, generation by type, prices and weather in Spain**



Pilar Denia García

The Bridge \_ 01/Dic/2020

Contenido

[0.- INTRODUCCIÓN 4](#_Toc57729864)

[1.- OBJETIVO 5](#_Toc57729865)

[2.- DATOS 6](#_Toc57729866)

[3.- EXPLORATORIO 8](#_Toc57729867)

[GENERAL 8](#_Toc57729868)

[ENERGY: 8](#_Toc57729869)

[1.b) DESCARGA DE FICHERO 8](#_Toc57729870)

[2.b) INDICE ENERGY 9](#_Toc57729871)

[3.b) NAN VALUES 9](#_Toc57729872)

[4.b) SIMPLIFICACIÓN ATRIBUTOS 10](#_Toc57729873)

[5.b) HISTOGRAMAS**.** 11](#_Toc57729874)

[6.b) DENSIDAD DE PROBABILIAD 13](#_Toc57729875)

[7.b) DIAGRAMA DE CAJAS 14](#_Toc57729876)

[8.b) TRATAMIENTO DE OUTLAYERS 15](#_Toc57729877)

[9.b) CORRELACIÓN 16](#_Toc57729878)

[10.b) VISUALIZACIÓN DEL COMPORTAMIETNO DE LAS FUENTES DE ENERGÍA 18](#_Toc57729879)

[11.b) DEMANDA Y PRECIO 20](#_Toc57729880)

[WEATHER 24](#_Toc57729881)

[1.c) DESCARGA DE FICHERO 24](#_Toc57729882)

[2.c) AGRUPACIÓN VALORES RECOGIDA DE AGUA 25](#_Toc57729883)

[3.c) CONVERSOR DE TEMPERATURA 25](#_Toc57729884)

[4.c) HISTOGRAMAS WEATHER 25](#_Toc57729885)

[5.c) DIAGRAMAS DE CAJAS 26](#_Toc57729886)

[6.c) TRATAMIENTO DE OUTLAYERS 29](#_Toc57729887)

[6.c) DENSIDAD DE PROBABLIDAD, SIN OUTLAYERS. 30](#_Toc57729888)

[6.d) VISUALIZACIÓN DE ATRIBUROS FINAL (PAIRPOLT) 32](#_Toc57729889)

[DATAFRAME FINAL – MERGE ENERGY Y WEATHER 33](#_Toc57729890)

[1.d) INDICES 33](#_Toc57729891)

[2.d) UNIÓN DE LOS CONJUNTOS DE DATOS ‘ENERGY’ Y ‘WEATHER’ 37](#_Toc57729892)

[IMÁGENES: 39](#_Toc57729893)

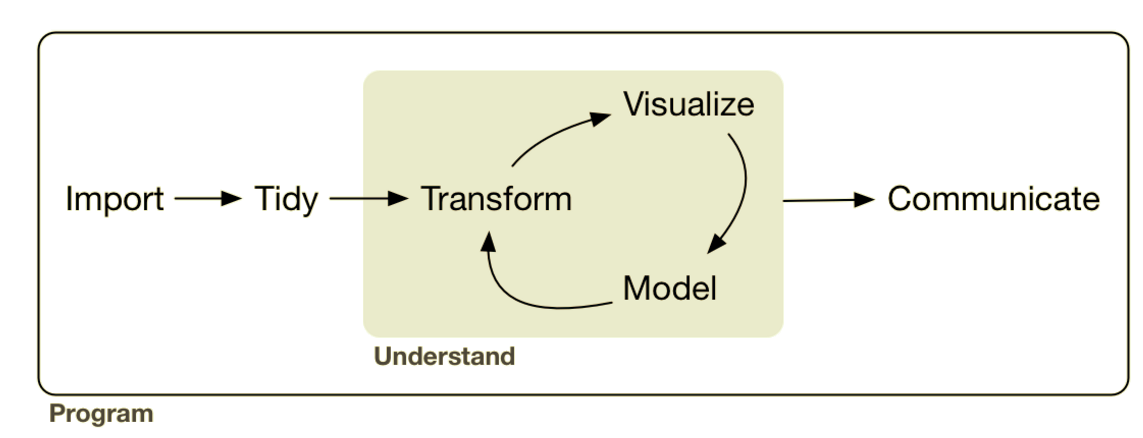
[ANEXO 40](#_Toc57729894)

# 0.- INTRODUCCIÓN

Este es un ejercicio absolutamente educativo, para afianzar los conocimientos y contenidos que se están adquiriendo en el bootcamp impartido en TheBridge, edición Junio2020 modalidad Part\_Time.

La finalidad del EDA es descubrir los patrones en los datos. Esto es fundamental para dirigirlos hacia un modelado predictivo con un objetivo final de anticipación a la hora de toma de decisiones por parte del área de negocio.

Los pasos a seguir son:



Img\_01 . – Fases del Exploratory Data Analysis.

Proceso del ejercicio exploratorio:

* Selección de las fuentes
* Importación de el/los conjunto/s de dato/s
* Estudio y ordenación
* Transformación de los datos
* Visualización de variables

# 1.- OBJETIVO

La energía es uno de los recursos en la vida de los seres humanos junto con otras necesidades esenciales que se ha vuelto primordial.

El hombre moderno se vale de la energía eléctrica para facilitar la realización de multitud de actividades complejas, y que ha supuesto una evolución y desarrollo exponencial desde principios del S.XIX.

Como ya se sabe el principio de la energía es que ni se crea ni se destruye sólo se transforma, la cantidad de energía se mantiene constante. Ahí radica la dificultad de mantener el equilibrio entre la demanda y la generación, puesto que el exceso se perdería y, en caso contrario podría provocar una caída de la red, con un impacto económico y social más importante.

Una de las finalidades del estudio objeto de este ejercicio es poder predecir la demanda para mantener ese equilibro energético.

El segundo y no menos importante para el negocio es predecir el precio a aplicar teniendo en cuenta cual será la fuente de origen que genere dicha energía.

España es un país con una fuerte dependencia de los recursos fósiles, y sin ser productores de esta materia prima, el impacto en precio energético (kWh) es alto. Por otra parte, si esa generación se obtiene de las fuentes renovables el coste y, por tanto, el precio/hora disminuye.

La generación renovable tanto la eólica como la solar, están directamente vinculadas a los efectos atmosféricos; sol, nubes, lluvia, viento, temperatura. De ahí que se haga uso también de información climatológica en el ejercicio exploratorio.

# 2.- DATOS

En el ejercicio se analiza la información obtenida en la plataforma [kaggel](https://www.kaggle.com/nicholasjhana/energy-consumption-generation-prices-and-weather?select=weather_features.csv) sobre un conjunto de datos que abarcan 4 años (Enero2015 a Diciembre2018) en España en intervalos de 1h:

A.- Documento en formato '.csv' con un peso de 28,8MB. A través de dos fuentes de origen con datos abiertos; [Entsole Transparency Platform](https://transparency.entsoe.eu/dashboard/show)  facilita los registros sobre la generación energética, previsión y demanda ,y el precio se recoge desde [Red Electric España](https://www.esios.ree.es/es?locale=en).

Fichero 'energy\_dataset.csv' está distribuido en:

|  |  |
| --- | --- |
| **Field** | **Values** |
| time | Datetime index localized to CET |
| generationbiomass | biomass generation in MW |
| generation fossil brown coal/lignite | coal/lignite generation in MW |
| generation fossil coal-derived gas | coal gas generation in MW |
| generation fossil gas | gas generation in MW |
| generation fossil hard coal | coal generation in MW |
| generation fossil oil | oil generation in MW |
| generation fossil oil shale | shale oil generation in MW |
| generation fossil peat | peat generation in MW |
| generation geothermal | geothermal generation in MW |
| generation hydro pumped storage aggregated | hydro1 generation in MW |
| generation hydro pumped storage consumption | hydro2 generation in MW |
| generation hydro run-of-river and poundage | hydro3 generation in MW |
| generation hydro water reservoir | hydro4 generation in MW |
| generation marine | sea generation in MW |
| generation nuclear | nuclear generation in MW |
| generation other | other generation in MW |
| generation other renewable | other renewable generation in MW |
| generation solar | solar generation in MW |
| generation waste | waste generation in MW |
| generation wind offshore | wind offshore generation in MW |
| generation wind onshore | wind onshore generation in MW |
| forecast solar day ahead | forecasted solar generation |
| forecast wind offshore eday ahead | forecasted offshore wind generation |
| forecast wind onshore day ahead | forecasted onshore wind generation |
| total load forecast | forecasted electrical demand |
| total load actual | actual electrical demand |
| price day ahead | forecasted price EUR/MWh |
| price actual | price in EUR/MWh |

B.- Archivo también en formato '.csv' (18,9 MB), recoge [información climatológica](mailto:https://openweathermap.org/api) de las 5 ciudades más importantes del país. Los parámetros del clima definidos en el dataset se pueden consultar en [documentación](mailto:https://openweathermap.org/history-bulk%23data) informativa de la API.

Fichero 'weather\_features.csv' está distribuido en las columnas:

|  |  |
| --- | --- |
| **Field** | **Values** |
| time | Datetime index localized to CET |
| city\_name | name of city |
| temp | in k |
| temp\_min | minimum in k |
| temp\_max | maximum in k |
| pressure | pressure in hPa |
| humidity | humidity in % |
| wind\_speed | wind speed in m/s |
| wind\_deg | wind direction |
| rain\_1h | rain in last hour in mm |
| rain\_3h | rain last 3 hours in mm |
| snow\_3h | show last 3 hours in mm |
| clouds\_all | cloud cover in % |
| weather\_id | Code used to describe weather |
| weather\_main | Short description of current weather |
| weather\_description | Long description of current weather |
| weather\_icon | Weather icon code for website |

# 3.- EXPLORATORIO

## GENERAL

Se empieza importando los módulos necesarios. Se harán uso de :

* Numpy
* Pandas
* Datetime
* Calendar
* OS

Visualización:

* Matplotlib.pyplot
* Seaborn
* Plotly.offline.iplot
* Plotly.graph\_objs

Módulo propio con funciones de tratamiento personalizado

* Functions\_EDA

## ENERGY:

### 1.b) DESCARGA DE FICHERO

Se importa fichero ‘.csv’ de datos energéticos.

Se comprueba que todos los registros facilitados en la plataforma sean descargado correctamente.

La dimensión inicial del dataset es de 35064 registros y 29 columnas.

La filas recoge los atributos:

energy.info()

RangeIndex: 35064 entries, 0 to 35063

Data columns (total 29 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 time 35064 non-null object

1 generation biomass 35045 non-null float64

2 generation fossil brown coal/lignite 35046 non-null float64

3 generation fossil coal-derived gas 35046 non-null float64

4 generation fossil gas 35046 non-null float64

5 generation fossil hard coal 35046 non-null float64

6 generation fossil oil 35045 non-null float64

7 generation fossil oil shale 35046 non-null float64

8 generation fossil peat 35046 non-null float64

9 generation geothermal 35046 non-null float64

10 generation hydro pumped storage aggregated 0 non-null float64

11 generation hydro pumped storage consumption 35045 non-null float64

12 generation hydro run-of-river and poundage 35045 non-null float64

13 generation hydro water reservoir 35046 non-null float64

14 generation marine 35045 non-null float64

15 generation nuclear 35047 non-null float64

16 generation other 35046 non-null float64

17 generation other renewable 35046 non-null float64

18 generation solar 35046 non-null float64

19 generation waste 35045 non-null float64

20 generation wind offshore 35046 non-null float64

21 generation wind onshore 35046 non-null float64

22 forecast solar day ahead 35064 non-null float64

23 forecast wind offshore eday ahead 0 non-null float64

24 forecast wind onshore day ahead 35064 non-null float64

25 total load forecast 35064 non-null float64

26 total load actual 35028 non-null float64

27 price day ahead 35064 non-null float64

28 price actual 35064 non-null float64

dtypes: float64(28), object(1)

memory usage: 7.8+ MB

Puesto que el objetivo es determinar el precio, se establece como ‘target’ la columna ‘price actual’.

Precio 🡺 dato numérico 🡺 regresión.

### 2.b) INDICE ENERGY

Se asigna como índice la columna temporal conservando ‘utc’.

### 3.b) NAN VALUES

Inicialmente, se revisa la naturaleza de los datos de las columnas, la cantidad de Nan/null o datos a '0'.

#check how many total rows and columns we have and how many of those are missing or zero values.

print(('NUMBER NULL/NAN\_VALUES OF COLUMNS DATAFRAME:\n'))

energy.isnull().sum()

NUMBER NULL/NAN\_VALUES OF COLUMNS DATAFRAME:

[34]:

generation biomass 19

generation fossil brown coal/lignite 18

generation fossil coal-derived gas 18

generation fossil gas 18

generation fossil hard coal 18

generation fossil oil 19

generation fossil oil shale 18

generation fossil peat 18

generation geothermal 18

generation hydro pumped storage aggregated 35064

generation hydro pumped storage consumption 19

generation hydro run-of-river and poundage 19

generation hydro water reservoir 18

generation marine 19

generation nuclear 17

generation other 18

generation other renewable 18

generation solar 18

generation waste 19

generation wind offshore 18

generation wind onshore 18

forecast solar day ahead 0

forecast wind offshore eday ahead 35064

forecast wind onshore day ahead 0

total load forecast 0

total load actual 36

price day ahead 0

price actual 0

dtype: int64

### 4.b) SIMPLIFICACIÓN ATRIBUTOS

Se simplifica el DataFrame a las fuentes principales de generación. Se agrupa la generación eléctrica por; fósil, hidráulica, nuclear, solar, eólica y en otras (renovables y no renovables).

Una vez se han agrupado las columnas indicadas, se vuelve a verificar las que tienen datos tipo 'NaN', y cuantos registros en cada serie.

energy.isnull().any()

generation fossil False

generation hydro False

generation nuclear True

generation other False

generation solar True

generation wind False

forecast solar day ahead False

forecast wind eday ahead False

total load forecast False

total load actual True

price day ahead False

price actual False

dtype: bool

NUMBER NULL/NAN\_VALUES OF COLUMNS DATAFRAME:

[47]:

generation fossil 0

generation hydro 0

generation nuclear 17

generation other 0

generation solar 18

generation wind 0

forecast solar day ahead 0

forecast wind eday ahead 0

total load forecast 0

total load actual 36

price day ahead 0

price actual 0

dtype: int64

### 5.b) HISTOGRAMAS**.**

Con la visualización gráfica de los histogramas de las variables del dataset 'energy', se inicia el estudio de las funciones de densidad de probabilidad del conjunto de datos, así como intuir la naturaleza de las muestras en cada uno de los atributos.

Gráfico

Descripción generada automáticamente

Img\_02 . – Energy histograms\_origin

### 6.b) DENSIDAD DE PROBABILIAD

El siguiente paso natural es estudiar desde los histogramas hacia las funciones de densidad de probabilidad, y cómo se distribuyen los datos.

Gráfico

Descripción generada automáticamente

Img\_03. – Energy KDE\_origin

### 7.b) DIAGRAMA DE CAJAS

Para identificar mejor los valores anómalos o 'outlayers', una de las mejores herramienta de visualización que se dispone es el diagrama de cajas, y así, tener en cuenta como se filtrarán estos datos en cada atributo.

Gráfico

Descripción generada automáticamente

Img\_04. – Energy\_genertion\_boxplot\_origin

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Img\_05. – Energy load boxplot origin

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Img\_06. – Energy price boxplot origin

### 8.b) TRATAMIENTO DE OUTLAYERS

Se crea una función específica para filtrar los valores anómalos de cada serie.

Esta función está dentro de un módulo propio 'functions\_EDA.py'\* que se ha importado junto con el resto de los paquetes, para optimizar su uso en los dataset que se trabajen en otro proyectos de exploración.

# columns are traversed and outlayer\_drop function is executed

for i in energy.columns:

print(i.upper())

#function drop\_outlayer

energy = f\_eda.drop\_outlayers\_df(energy, i)

print('Delete', i,'outlayers, length dataframe is ',energy.shape[0])

print('\n')

\*Anexo: Module, functions\_EDA.py

GENERATION FOSSIL

Delete generation fossil outlayers, length dataframe is 34891

GENERATION HYDRO

Delete generation hydro outlayers, length dataframe is 34582

GENERATION NUCLEAR

Delete generation nuclear outlayers, length dataframe is 34462

GENERATION OTHER

Delete generation other outlayers, length dataframe is 33723

GENERATION SOLAR

Delete generation solar outlayers, length dataframe is 33723

GENERATION WIND

Delete generation wind outlayers, length dataframe is 33330

FORECAST SOLAR DAY AHEAD

Delete forecast solar day ahead outlayers, length dataframe is 33330

FORECAST WIND EDAY AHEAD

Delete forecast wind eday ahead outlayers, length dataframe is 33156

TOTAL LOAD FORECAST

Delete total load forecast outlayers, length dataframe is 33156

TOTAL LOAD ACTUAL

Delete total load actual outlayers, length dataframe is 33156

PRICE DAY AHEAD

Delete price day ahead outlayers, length dataframe is 32562

PRICE ACTUAL

Delete price actual outlayers, length dataframe is 32229

### 9.b) CORRELACIÓN

Se visualizará las correlaciones lineales de dos en dos.

Patrón de fondo

Descripción generada automáticamenteImg\_07. – Energy pairplot origin

Aparte de las relaciones directas y lógicas que existen entre generación y estimación solar y eólica, así como en demanda y precio. Se puede observar que la generación fósil es la que tiene visualmente una correlación más evidente con la variable demanda y precio.

En este punto se van a medir las relaciones lineales que existen entre los diferentes atributos.

Se verifica numéricamente que la generación fósil es la que tiene una correlación lineal más fuerte con la variable demanda y precio. Y una relación inversa con la generación eólica.

Teniendo en cuenta que el dato que se desea averiguar a futuro es el precio por kW/hora, se ordenan las correlaciones lineales que tiene con el resto de atributos.

energy.corr()['price actual'].abs().sort\_values(ascending=False)

price actual 1.000000

price day ahead 0.720350

generation fossil 0.483992

total load actual 0.418124

total load forecast 0.417587

generation other 0.217212

generation wind 0.170474

forecast wind eday ahead 0.169225

forecast solar day ahead 0.112777

generation solar 0.109201

generation nuclear 0.072682

generation hydro 0.032006

Name: price actual, dtype: float64

Se ha podido comprobar con el gráfico 'pairplot' y, ahora con el valor numérico que existen varias columnas con correlaciones lineales prácticamente iguales.

Se eliminarán del dataset para evitar la multicolinealidad una de ambas de las columnas implicadas; total load forecast, forecast wind eday ahead y forecast solar day ahead, y en este caso se mantendrán los valores reales.

Se elimina del dataset, la previsión (price day ahead) de precio por hora, ya que se dispone del precio final cerrado de consumo.

### 10.b) VISUALIZACIÓN DEL COMPORTAMIETNO DE LAS FUENTES DE ENERGÍA

En este punto se inicia la observación del comportamiento de los diferentes tipos de generación, y se utilizará gráfico de línea.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_08. – Generation behavior of energy sources(2015-2018)

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_09. – Generation behavior of energy sources during 2018

Con ambas gráficas se puede ver que la generación hidroeléctrica, la eólica y la solar son claramente estacionales.

La energía obtenida a través del sol tiene un ligero aumento de producción en los meses de verano y principios de otoño, y un descenso moderado en invierno.

La energía eólica tiene su pico de producción en primavera, importante cuota tanto en invierno como otoño, y un valle muy pronunciado en verano.

La época de mayor generación hídrica es en el periodo de primavera, se interpreta que es por ser la época de deshielo y lluvias. Podemos observar, por ejemplo, en la gráfica entre 2015 y 2018, que en el año 2017 no se obtuvo una importante producción en dicho periodo, por tanto, pudo darse un invierno de sequía, sin muchas precipitaciones.

El comportamiento de la energía fósil, parece que está fuertemente condicionado tanto a la eólica como a la hidroeléctrica, ya que su producción es inversa a la generación de estas. Esto denota que es una fuente programable y de rápida respuesta para cubrir la demanda.

Las 'otras' fuentes de energía y la nuclear, tiene un comportamiento muy estable dentro de la generación total.

Ahora se busca visualizar los pesos de los diferentes tipos de fuentes con un gráfico de barras acumulado, colocando en la base las más constantes y que tienen menos relación directa con el clima. Se utilizará las medias mensuales, y se pintará también línea de la demanda total media. Se elegirá el último año del que se dispone para visualizar la dimensión que representa cada origen de generación.

Gráfico, Gráfico de barras

Descripción generada automáticamente

Img\_10. – Generation of energy during 2018

### 11.b) DEMANDA Y PRECIO

En este punto se analizará dos de los atributos que determinan el 'negocio' energético; la demanda y el precio, revisándose en detalle.

**Demanda** (Load energy). Visualización de la demanda teniendo en cuenta diferentes lineas temporares.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_11. – Load energy (2015 – 2018)

El comportamiento de la generación y la demanda se producen de forma ajustada, puesto que la energía es un recurso que no se puede almacenar. Cuando hay demanda la generación tiene que cubrirla.

Se utilizará los datos del último año para observar la demanda a lo largo de 2018, y se visualizan las fechas de los festivos nacionales para determinar el efecto de estas en el consumo.

Gráfico

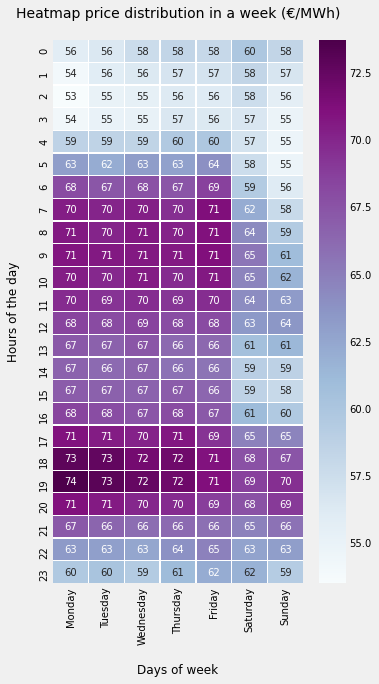
Descripción generada automáticamente

Img\_12. – Load energy during 2018

Con esta gráfica se puede apreciar que existe un efecto directo en la demanda energética durante las fiestas y los periodos vacacionales.

Ahora con una gráfica de calor, se visualizará la demanda durante el intervalo de una semana, para poder observar las horas de más pico y valle energético.

Gráfico

Descripción generada automáticamente  Img\_13. – Heatmap electric LOAD in a week Img\_14. – Heatmap electric PRICE in a week

**Precio.** Para el precio que es el atributo 'target' para establecer el modelo de este exploratorio, también se va hacer uso de un mapa de calor.

Las oscilaciones de los precios en los intervalos horarios durante una semana son prácticamente iguales a las variaciones de demanda energética en dicho momento.

En ambos gráficos se muestra con claridad que el pico de mayor uso energético es de lunes a jueves de 7:00a 11:00h y, por la tarde, entre las 17:00 y 20:00h.

Durante el horario laboral existe una alta demanda ,y el momento de menor precio es por las noche entre la 1:00 y las 3:00h.

Visualizamos la fluctuación del precio a lo largo de todo un día, teniendo en cuenta los 7 días de la semana.

Visualizamos la fluctuación del precio a lo largo de todo un día, teniendo en cuenta los 7 días de la semana.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_15. – Price fluctuation in one day (€/kWh)

## WEATHER

### 1.c) DESCARGA DE FICHERO

Se importa fichero ‘.csv’ de datos climatológicos, y en este paso se realizará la manipulación correspondiente al DataFrame 'weather'.

Se modifica el tipo de dato de la serie temporal a numpy.datetime64 manteniendo el ‘utc’, aunque se desea utilizar el dato de fecha/hora como índice puesto que será el atributo por el que se vincule ambos datasets, de momento se mantiene el generado automáticamente por números correlativos enteros para realizar diferentes operaciones de limpieza.

La dimensión inicial del dataset es de 178396 registros y 17 columnas.

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 178396 entries, 0 to 178395

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 time 178396 non-null datetime64[ns]

1 city\_name 178396 non-null object

2 temp 178396 non-null float64

3 pressure 178396 non-null int64

4 humidity 178396 non-null int64

5 wind\_speed 178396 non-null int64

6 water\_collected 178396 non-null float64

7 clouds\_all 178396 non-null int64

8 weather\_id 178396 non-null int64

9 weather\_main 178396 non-null object

dtypes: datetime64[ns](1), float64(2), int64(5), object(2)

memory usage: 13.6+ MB

Se observa que no hay registros con valores null.

Se elimina los atributos de 'temp\_min' y 'temp\_max', puesto que disponemos de un valor general de temperatura durante ese intervalo de hora.

La columna categórica de 'weather\_description' debido a que tenemos una clasificación principal del 'tiempo' se elimina del conjunto de datos, y también 'weather\_icon' que los valores que tiene la API como imagen de representación del tiempo.

Se descarta la columna de orientación del viento puesto que es un atributo que sería importante dependiendo de la ubicación del parque eólico y los flujos de corrientes de la zona, y como inciden en la generación con respecto a otros parques, pero no se considerará en este caso a nivel nacional.

### 2.c) AGRUPACIÓN VALORES RECOGIDA DE AGUA

Se revisa el montante de los datos de lluvia recogida en '1h'y en las 3h (mm/h), para cuantificar el volumen y su peso en el conjunto de datos. Teniendo en cuenta que estos datos están recogiendo la información de agua en sólo 5 puntos de todo el territorio y su dimensión no es significativa, se decide acumularlas con los registros de lluvia que existen de 1h.

Se acumulan en una sola serie, se con el nombre de 'water\_collected'.

### 3.c) CONVERSOR DE TEMPERATURA

Los datos de la serie de temperatura están registrados en grados 'kelvin', se realiza conversión a grados 'celsius' que son la métrica con la que estamos más familiarizados.

### 4.c) HISTOGRAMAS WEATHER

En este punto se explorará los datos de cada columna, y eliminar datos fuera de rango.

Gráfico, Histograma

Descripción generada automáticamente

Img\_16 . – Weather histograms origin.

### 5.c) DIAGRAMAS DE CAJAS

Una vez visto los histogramas de las columnas, existe varios atributos que destacan por su oscilación de valores:

1.- Pressure

2.- Wind Speed

2.- Water collected

El gráfico de caja nos dará más información a la hora de identificar 'outlayers' de las columnas de 'weather', así se podrán limpiar para obtener el dataset final.

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Img\_17 . – Weather. Temperature boxplot origin.

Imagen que contiene Gráfico de líneas

Descripción generada automáticamente

Img\_18 . – Weather. Pressure boxplot origin.

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Img\_19 . – Weather. Humidity boxplot origin.

Gráfico

Descripción generada automáticamente

Img\_20 . – Weather. Wind seed boxplot origin.

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Img\_21 . – Weather. Water collected seed boxplot origin.

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Img\_22 . – Weather. Cloudiness percentage boxplot origin.

Según el gráfico de cajas hay un claro atributo que tiene valores que pueden llegar a distorsionar en exceso el resultado de la computación.

La serie 'pressure' posee outlayers con valores elevadísimos. Al visualizar los datos estadísticos de cada atributo, en 'pressure' se observa que tiene una desviación de casi 6000hPa. cuando su media, su mediana y los cuartiles 25% y 75% están entorno a 1.000 hPa.

### 6.c) TRATAMIENTO DE OUTLAYERS

Al igual que en el conjunto de datos de ‘energy’ se hace unos de la función ‘drop\_outlayers\_df()’ del módulo 'functions\_EDA.py'\*, sobre las columnas numéricas, no sobre las columnas categórica que se han eliminado previamente.

TEMP

Delete temp outlayers, length dataframe is 178203

PRESSURE

Delete pressure outlayers, length dataframe is 166406

HUMIDITY

Delete humidity outlayers, length dataframe is 166391

WIND\_SPEED

Delete wind\_speed outlayers, length dataframe is 164239

WATER\_COLLECTED

Delete water\_collected outlayers, length dataframe is 145433

CLOUDS\_ALL

Delete clouds\_all outlayers, length dataframe is 124995

#Delete rows have outlayers, use functions\_EDA module

for i in weather.columns[2:8]:

print(i.upper())

weather = f\_eda.drop\_outlayers\_df(weather, i)

#weather = drop\_outlayers\_df(weather,i)

print('Delete', i,'outlayers, length dataframe is ',weather.shape[0])

print('\n')

### 6.c) DENSIDAD DE PROBABLIDAD, SIN OUTLAYERS.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_ 23. – Weather. Temperature KDE.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_24 . – Weather. Pressure KDE.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_25 . – Weather. Humidity KDE.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_20 . – Weather. Wind seed KDE.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Img\_21 . – Weather. Water collected seed KDE.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Img\_22 . – Weather. Cloudiness percentage KDE.

La serie 'water\_collected' del dataframe después de eliminar los outlayers tiene todos los registros a '0', no aportará valor al modelo. Se elimina del conjunto de atributos del dataset.

### 6.d) VISUALIZACIÓN DE ATRIBUROS FINAL (PAIRPOLT)

Imagen que contiene Gráfico de dispersión

Descripción generada automáticamente

Img\_23 . – Weather. Pairplot attributes.

## DATAFRAME FINAL – MERGE ENERGY Y WEATHER

### 1.d) INDICES

En este paso se comprobará los índices y se tratarán para que en ambos datasets coincidan, con la finalidad de unirlos a través de esta columna.

Como se había comentado al inicio del estudio del dataset 'weather', la fecha y hora será la columna por la que se realizará la unión con el dataset 'energy'.

En este momento se fija como Índex la serie temporal.

Se comprueba que en el conjunto de datos de índices de este DataFrame no son únicos, y por tanto, se explora la naturaleza de estos registros.

(124995, 32229)

weather.shape[0], energy.shape[0]

False

weather.index.is\_unique

Se seleccionan 2 índices aleatorios, para su observación.

[366]:

|  | **time** | **city\_name** | **temp** | **pressure** | **humidity** | **wind\_speed** | **water\_collected** | **clouds\_all** | **weather\_id** | **weather\_main** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **time** |  |  |  |  |  |  |  |  |  |  |
| **2018-01-15 19:00:00** | 2018-01-15 19:00:00 | Valencia | 11.15 | 1025 | 53 | 1 | 0.0 | 0 | 800 | clear |
| **2018-01-15 19:00:00** | 2018-01-15 19:00:00 | Madrid | 5.91 | 1027 | 70 | 3 | 0.0 | 0 | 800 | clear |
| **2018-01-15 19:00:00** | 2018-01-15 19:00:00 | Barcelona | 8.65 | 1022 | 75 | 4 | 0.0 | 0 | 800 | clear |
| **2018-01-15 19:00:00** | 2018-01-15 19:00:00 | Seville | 10.18 | 1028 | 76 | 3 | 0.0 | 0 | 800 | clear |
| **2018-04-06 02:00:00** | 2018-04-06 02:00:00 | Valencia | 11.15 | 1017 | 81 | 2 | 0.0 | 40 | 802 | clouds |
| **2018-04-06 02:00:00** | 2018-04-06 02:00:00 | Madrid | 9.15 | 1013 | 76 | 2 | 0.0 | 0 | 800 | clear |
| **2018-04-06 02:00:00** | 2018-04-06 02:00:00 | Bilbao | 12.83 | 1009 | 55 | 4 | 0.0 | 0 | 800 | clear |
| **2018-04-06 02:00:00** | 2018-04-06 02:00:00 | Barcelona | 9.15 | 1017 | 93 | 1 | 0.0 | 8 | 800 | clear |
| **2018-04-06 02:00:00** | 2018-04-06 02:00:00 | Seville | 12.33 | 1011 | 67 | 3 | 0.0 | 0 | 800 | clear |

weather.loc[np.random.choice(weather.index, size=2),:]

Los registros del dataset 'weather' que se ha descargado para el EDA, recogen sólo información climatológica de 1 a 5 puntos distintos (las ciudades de Valencia, Madrid, Bilbao, Barcelona, Sevilla) del territorio. Por tanto, en el mismo intervalo de día/hora se pueden llegar a tener 5 filas repetidas con ese índice pero con diferentes valores en el resto de atributos.

En este momento se tiene que estudiar el mejor criterio a la hora de eliminar los registros duplicados del dataset 'weather' para que coincida con los indices de 'energy'.

Una de las opciones sería seleccionar una de las ciudades, por ejemplo 'Madrid' puesto que se encuentra en el punto central y a una altura intermedia del país, y eliminar el resto de filas que tienen duplicado el índice.

Pero se considera más interesante a la hora de descartar las duplicidades del índice, agrupándolos por su índice y extrayendo sus valores medios según los fenómenos atmosféricos.

#unify values attributes with the same index

for index, value in weather.time.items():

#print(value) #indexDateTime == values

#print(weather.loc[value,:]) #DataFrame rows with the same index

#print(type(weather.loc[value,:]))

s\_means = weather.loc[value,weather.columns[2]:weather.columns[7]].mean() #Serie with means of attributes

if isinstance(weather.loc[value,:], pd.DataFrame):

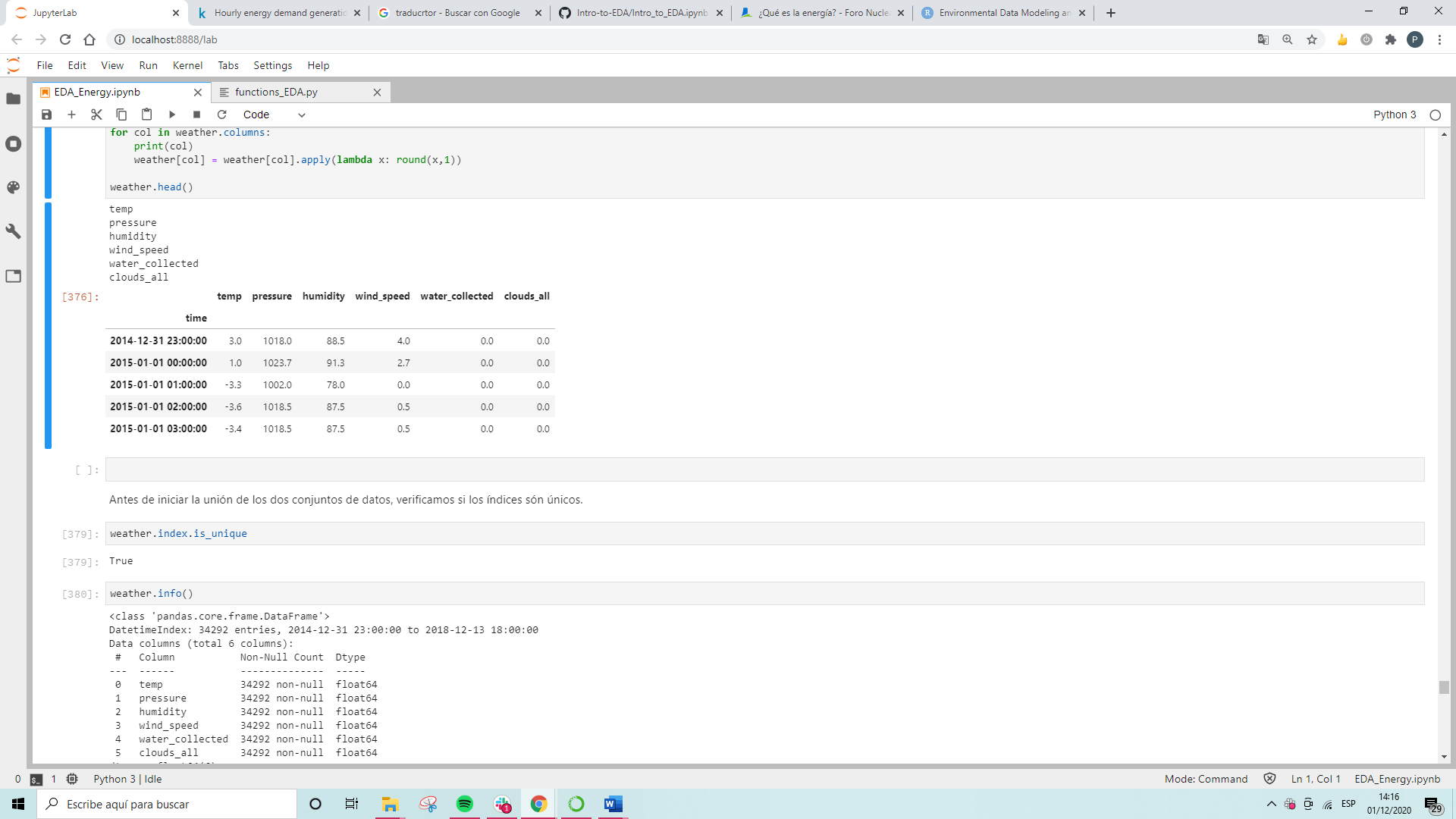
for column in weather.columns[2:-2]:

#print(s\_means[column])

weather.loc[value,column] = s\_means[column]

#print('\n')

Una vez los datos del mismo índice están unificados, se eliminarán los duplicados manteniendo el primer registro, y se elimina la columna con la se ha manipulado el dataset, que coincide con el IndexDateTime.



Antes de iniciar la unión de los dos conjuntos de datos, verificamos si los índices son únicos.

<class 'pandas.core.frame.DataFrame'>

DatetimeIndex: 34292 entries, 2014-12-31 23:00:00 to 2018-12-13 18:00:00

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 temp 34292 non-null float64

1 pressure 34292 non-null float64

2 humidity 34292 non-null float64

3 wind\_speed 34292 non-null float64

4 water\_collected 34292 non-null float64

5 clouds\_all 34292 non-null float64

dtypes: float64(6)

weather.info()

Verificamos cuantos Nan\_values y que porcentaje de valores a '0' cero hay en cada columna.

temp False

pressure False

humidity False

wind\_speed False

water\_collected False

clouds\_all False

dtype: bool

PERCENTAGE ZERO\_VALUES OF COLUMNS DATAFRAME:

0.- temp : 0.017 %

1.- pressure : 0.0 %

2.- humidity : 0.0 %

3.- wind\_speed : 0.321 %

4.- water\_collected : 100.0 %

5.- clouds\_all : 22.76 %

f\_eda.percentage\_zeros\_columns(weather)

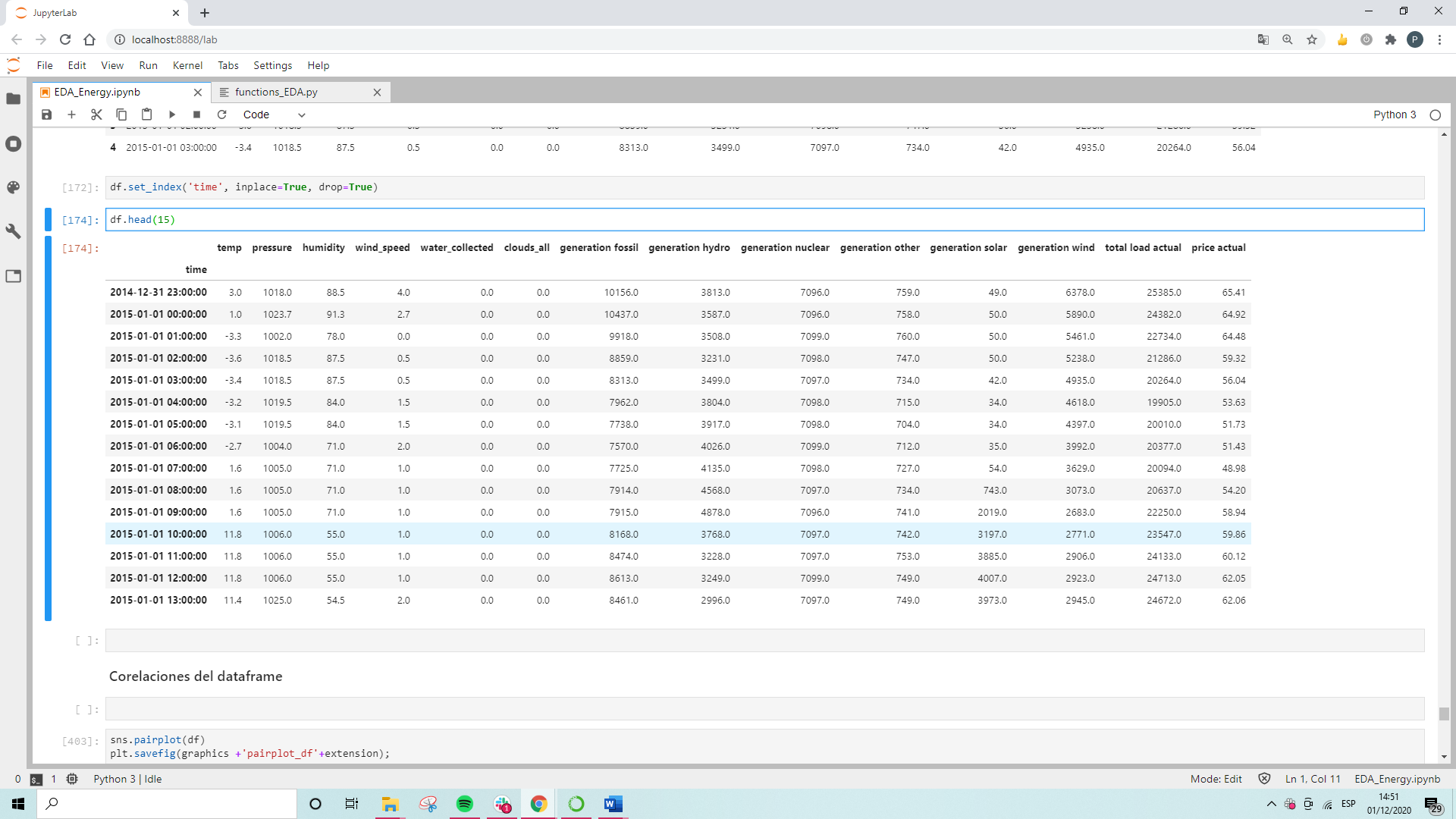
weather.isnull().any()

### 

### 2.d) UNIÓN DE LOS CONJUNTOS DE DATOS ‘ENERGY’ Y ‘WEATHER’

A partir de ahora se juntan los dos dataframes creando un nuevo dataframe utilizando los índices que coinciden. Descartamos las filas que no coinciden en ambos conjuntos dado que tenemos muchos registros y podemos descartar directamente los generarán ‘missing values’.

df = weather.merge(energy, left\_on=weather.index, right\_on=energy.index, how='inner')



Se ha finalizado el proceso de Exploración de los Datos.

El conjunto de datos resultantes consta de 31.601 filas y de 13 columnas;

* 'temp'
* 'pressure'
* 'humidity'
* 'wind\_speed'
* 'water\_collected'
* 'clouds\_all'
* 'generation fossil'
* 'generation hydro'
* 'generation nuclear'
* 'generation other'
* 'generation solar'
* 'generation wind'
* 'total load actual'
* 'price actual'

La columna de precio será a asignada como ‘target’ para el siguiente paso, el

Estudio de los modelos de ‘Machine Learning’, donde la columna de precio será asignada como ‘target’.

# IMÁGENES:

Img\_01 . – Fases del Exploratory Data Analysis.

Img\_02 . – Energy histograms origin.

Img\_03. – Energy KDE origin

Img\_04. – Energy generation boxplot origin

Img\_05. – Energy load boxplot origin

Img\_06. – Energy price boxplot origin

Img\_07. – Energy pairplot origin

Img\_08. – Generation behavior of energy sources(2015-2018)

Img\_09. – Generation behavior of energy sources during 2018

Img\_10. – Generation of energy during 2018

Img\_11. – Load energy (2015 – 2018)

Img\_12. – Load energy during 2018

Img\_13. – Heatmap electric LOAD in a week

Img\_14. – Heatmap electric PRICE in a week

Img\_15. – Price fluctuation in one day (€/kWh)

Img\_16 . – Weather histograms origin.

Img\_17 . – Weather. Temperature boxplot origin.

Img\_18 . – Weather. Pressure boxplot origin.

Img\_19 . – Weather. Humidity boxplot origin.

Img\_20 . – Weather. Wind seed boxplot origin.

Img\_21 . – Weather. Water collected seed boxplot origin.

Img\_22 . – Weather. Cloudiness percentage boxplot origin.

Img\_ 23. – Weather. Temperature KDE.

Img\_24 . – Weather. Pressure KDE.

Img\_25 . – Weather. Humidity KDE.

Img\_20 . – Weather. Wind seed KDE.

Img\_21 . – Weather. Water collected seed KDE.

Img\_22 . – Weather. Cloudiness percentage KDE.

Img\_23 . – Weather. Pairplot attributes.

# ANEXO

DOC\_1. Module functions\_EDA.py

import numpy as np

import pandas as pd

#Function explores number of Nan-Null values ​​in each Dataframe column

def number\_nan\_columns(df):

print(('number null/Nan\_values of columns Dataframe:\n').upper())

for i in range(len(df.columns)):

print(i,df.columns[i],':',df.iloc[:,i][df.iloc[:,i].isnull()].shape[0])

#Function explores percentage of Nan-Null values ​​in each Dataframe column

def percentage\_nan\_columns(df):

lenght\_col = df.shape[0]

print(('Percentage nan\_values of columns Dataframe:\n').upper())

for i in range(len(df.columns)):

num\_nan\_col = df.iloc[:,i][df.iloc[:,i].isnull()].shape[0]

print(str(i)+'.-',df.columns[i],':',round((num\_nan\_col/lenght\_col)\*100,4),'%\n')

#Function explores percentage of Zero values in each Dataframe column

def percentage\_zeros\_columns(df):

ar\_num\_zero = np.count\_nonzero(df, axis=0)

lenght\_col = df.shape[0]

print(('Percentage zero\_values of columns Dataframe:\n').upper())

for i in range(len(df.columns)):

print(str(i)+'.-',df.columns[i],':',

round(((lenght\_col-ar\_num\_zero[i])/lenght\_col)\*100,3),'%\n')

#function drop outlayers\_maximum and outlayers\_minimum of DataFrame

#return df

#function drop outlayers\_maximum and outlayers\_minimum of DataFrame

#return df

def drop\_outlayers\_df(df,column):

#get 1est quartile and 3er quartile

q1 = df[column].quantile(q=0.25)

q3 = df[column].quantile(q=0.75)

#inter quartile range

iqr = q3 - q1

#get maximun

maximum = q3 + (1.5\*iqr)

#get minimum

minimum = q1 - (1.5\*iqr)

#get indexes row outlayers, max and min

idx\_maximum = df[df[column] > maximum].index

idx\_minimum = df[df[column] < minimum].index

#drop rox outlayers

df.drop(idx\_maximum, axis = 0, inplace = True)

df.drop(idx\_minimum, axis = 0, inplace = True)

return df

#function drop outlayers\_maximum and outlayers\_minimum of DataFrame

#return df

def drop\_outlayers\_df(df,column):

#get 1est quartile and 3er quartile

q1 = df[column].quantile(q=0.25)

q3 = df[column].quantile(q=0.75)

#inter quartile range

iqr = q3 - q1

#get maximun

maximum = q3 + (1.5\*iqr)

#get minimum

minimum = q1 - (1.5\*iqr)

#get indexes row outlayers, max and min

idx\_maximum = df[df[column] > maximum].index

idx\_minimum = df[df[column] < minimum].index

#drop rox outlayers

df.drop(idx\_maximum, axis = 0, inplace = True)

df.drop(idx\_minimum, axis = 0, inplace = True)

return df