



Universidad de Cantabria

Facultad de Ciencias

La velocidad del viento y el precio del KWh en el mercado eléctrico español

Data Life Cycle

Master en Data Science

Autores:

Roldán Gutiérrez García

Marcos Cobo Carrillo

Jose Miguel Fraile Muñoz

Pablo Calatayud Pelayo

Marco Antonio Melgarejo Aragón

Szymon Piotr Bramowicz

Enero 2022

Índice

1. Descripción general del proyecto	2
2. Descripción del problema	2
3. Realización del Data Management Plan	3
3.1. Información Administrativa	3
3.1.1. Informe de Reuniones.	3
3.2. Descripción de los datos, recolección y utilización de datos existentes.	3
3.3. Documentación y calidad de los datos.	4
3.4. Almacenamiento y copias de seguridad durante el proyecto.	4
3.5. Requerimientos éticos, legales y códigos de conducta.	4
3.6. Distribución de los datos y preservación.	4
3.6.1. Software Empleado	5
3.6.2. Hardware Empleado	5
3.7. Responsabilidad en la gestión de los datos y sus fuentes.	5
4. Diagrama de Gantt	5
5. Descripción de las fuentes de datos	6
6. Curar/limpiar datos	8
7. Creación de metadatos Dublin Core	17
8. Plan de preservación	18
9. Análisis de los datos y conclusiones	19

1. Descripción general del proyecto

En el presente trabajo vamos a realizar un estudio sobre el precio histórico de la energía eléctrica en España y cómo las condiciones eólicas alteran el precio de la energía eléctrica en el mercado.

En España, el coste final de la factura doméstica de la luz depende de tres grandes componentes: los costes regulados, el coste de la generación eléctrica y los impuestos. Estos últimos son fijos e iguales para todas las productoras eléctricas, al igual que los costes regulados del sistema, que abarcan el transporte, la distribución, el pago de los intereses del déficit de tarifa, el incentivo que se paga a las grandes industrias por reducir su consumo eléctrico en caso de que fuese necesario, los pagos por capacidad a tecnologías convencionales, los incentivos a las renovables y la cogeneración, y los sobre costes por la generación eléctrica en las islas, donde resulta más caro producir que en la Península. El único factor del precio de la energía que difiere para cada una de las generadoras es el coste de generación eléctrica.

El coste de la energía se fija en un mercado competitivo donde las diferentes fuentes de energía ofertan la electricidad para satisfacer la demanda prevista con un día de antelación, fijándose el precio del kWh en el punto de equilibrio entre ambas. Como el coste del viento es cero, cabe esperar que los productores de energía eólica, en días de mucha actividad eólica, pueden ofertar la electricidad a un precio más bajo en comparación a otros productores que emplean combustibles más caros, como el gas, y desplazar así el punto de equilibrio hacia una zona de precios mas bajos. Por eso, en este trabajo vamos a intentar probar, con multitud de datos, que el precio del kWh en el mercado eléctrico baja los días de más viento.

Para la realización de nuestro estudio recogeremos datos sobre el precio de la electricidad en España del propio mercado eléctrico español y de la velocidad media del viento en España a través de anemómetros previamente colocados en zonas estratégicas. En secciones posteriores profundizaremos en cómo extraeremos y procesaremos estos datos.

Una vez obtenidos los históricos de precios y velocidades eólicas en España, seremos capaces de estudiar la influencia de una variable en la otra y obtener distintas conclusiones gracias a estos datos.

2. Descripción del problema

El interés de este proyecto se extiende mucho más allá de meramente conocer si existe o no una relación entre el coste del MWh en España y la velocidad media del viento en la península. Aunque los sistemas de fijación de precios alrededor del mundo no sean idénticos al español (sí en Europa), estos sistemas, denominados sistemas marginalistas, se basan en las mismas ideas.

Si probamos que existe una correlación entre el precio del MWh y la velocidad del viento, como la mayoría de los mercados eléctricos mundiales, fundamentalmente, operan de forma parecida, podemos asumir que este resultado es extrapolable a muchos otros mercados, y esto tiene una repercusiones enormes al largo plazo.

Que el precio del mercado eléctrico caiga cuando sopla el viento nos puede sugerir que, en determinadas circunstancias, la energía eólica se impone ante otras fuentes tradicionales, como el carbón o el gas. Estas últimas fuentes, como ya sabemos, son finitas, y cada día nos acercamos más a un desabastecimiento de ellas, lo cual está acarreando un encarecimiento de las energías tradicionales y un reemplazamiento de éstas por energías verdes, entre ellas, la eólica.

A su vez, que las energías verdes reemplacen a fuentes de energía tradicionales causará una reducción en los precios del gas, petróleo, etc, dado que, al abandonar estos recursos para la producción de energía, habrá una mayor oferta en el resto de mercados.

Como ya hemos comentado, el interés de probar la existencia de una relación entre el viento y el precio es enorme. Por ejemplo, justificaría en base a los datos, la construcción de parques eólicos en zonas como los Valles Pasiegos, que tanta problemática origina.

En este trabajo abarcaremos la península ibérica para la toma de datos, tanto del precio del MWh en el mercado eléctrico (español) como la velocidad del viento media en la península. Para obtener el valor medio del viento en la península instalaremos anemómetros en 5 de los parques eólicos con mayor producción anual (situados en León, Almería, A Coruña, Zaragoza, Albacete). Estos datos se recogerán durante al menos 6 años, con el objetivo de tener un histórico suficientemente extenso como

para extraer conclusiones firmes, pero una vez alcanzado este volumen de datos seguiremos recogiendo datos de forma indefinida.

Una vez explicado el propósito de este proyecto, comencemos a desarrollarlo.

3. Realización del Data Management Plan

3.1. Información Administrativa

Versión 2.3.2. Enero 2014.

La velocidad del viento y el precio del MWh en el mercado eléctrico español.

Trabajo realizado por los alumnos del grupo de economía para la asignatura de Data Life Cycle del Máster en Ciencia de Datos de la Universidad de Cantabria - Universidad Internacional Menéndez Pelayo.

El grupo de economía esta formado por los siguientes estudiantes

- *Marcos Cobo Carrillo.*
- *José Miguel Fraile Muñóz.*
- *Roldán Gutiérrez García.*
- *Szymon Piotr Bramowicz.*
- *Marco Antonio Melgarejo Aragón.*
- *Pablo Calatayud Pelayo*

3.1.1. Informe de Reuniones.

En esta subsección se va a detallar las reuniones realizadas y los puntos relevantes que han sido tratados.

1. Propuesta de temas y elección.
8 de Diciembre de 2021 de 15:30 a 17:30 UTC+1.
2. Kick off. Delimitación del proyecto.
13 de Enero del 2022 de 20:30 a 23:00 UCT+1.
3. Primeros pasos y asignación de tareas.
18 de Enero de 2022 de 21:00 a 23:00 UTC+1.
4. Formalización de las secciones.
22 de Enero de 2022 de 10:00 a 12:00 UTC+1.
5. Revisión y valoración de cada sección, realización del análisis y comienzo de la presentación.
23 de Enero de 2022 de 10:00 a 12:00 UTC+1.
6. Revisión de la presentación.
24 de Enero de 2022 de 22:00 a 23:00 UTC+1.

3.2. Descripción de los datos, recolección y utilización de datos existentes.

La base de datos esta constituida por 2 tablas. Una primera tabla con los datos de la velocidad del viento en las cinco provincias españolas con los parques eólicos mas determinantes (medido en cada parque eólico en concreto). Una segunda tabla con los precios de la electricidad en el mercado mayorista en €/MWh. Las tablas están relacionadas por medio del sistema de tiempo universal UTC *Coordinate Universal Time*. Cada red de anemómetros cuenta con cinco dispositivos conectados a través de un módem.

La base de datos también puede ser descrita de forma temporal. Por una lado disponemos de los datos de las tablas anteriormente mencionadas desde el 1 de Enero de 2015 hasta el día que comienza el proyecto y por otro lado iremos actualizando las tablas con los datos correspondientes acorde al desarrollo del programa.

El modelo de anemómetro utilizado es el *Anemómetro PCE-ADL 11* con software integrado. Los datos serán recogidos en un moden y transferidos a través del protocolo de transferencia de archivos *FTP* al servidor donde serán tratados y añadidos a la base de datos que se encuentra almacenada en el NAS.

Los datos correspondientes al precio de la electricidad serán recogidos usando el lenguaje de programación *python* por medio de la *API* mencionada en la sección 3 *Descripción de las fuentes de los datos*.

En el sección 8 *Plan de Preservación* se explican los detalles correspondiente a las necesidades en términos de memoria que requiere la base de datos.

Tanto en el sección 3 *Descripción de las fuentes de datos* como en el sección 6 *curar/limpiar datos* se profundizará en estos dos asuntos.

3.3. Documentación y calidad de los datos.

Los metadatos con la información contenida en las tablas están establecidos por medio del estándar *Dublin Core*. En la sección 7 *Creación de metadatos Dublin Core* se dispone de toda la información concerniente a este tema.

Los controles de calidad establecidos se basan en reglas para cada anemómetro y cada red. Si un anemómetro da un valor desvirtuado para un determinado momento, entonces tomará una segunda medida. En caso de que esta segunda medida siga siendo corrupta se procederá a realizar una comparación con los demás anemómetros de su misma red para analizar la causa.

3.4. Almacenamiento y copias de seguridad durante el proyecto.

El almacenamiento de la información se realizará en dos NAS a través de un servidor. Se instalará el software libre Bacula para el tratamiento de las copias de seguridad. Se realizarán copias de seguridad de diferentes tipos y a diferentes temporalidades. Se realizarán copias de seguridad incrementales diarias y copias de seguridad completas de forma semanal. También se subirá recurrentemente a *GitHub* la última versión de la copia de seguridad.

En caso de ser necesario recuperar una copia de seguridad, el procedimiento es el siguiente. Ir a Bacula y buscar la copia de seguridad correspondiente a la fecha requerida. Esta copia de seguridad lleva un un identificador único asociado a dicha copia llamado *jobid*. Con ese *jobid* se recupera el archivo y este es almacenado en la carpeta correspondiente.

3.5. Requerimientos éticos, legales y códigos de conducta.

La electricidad en el mercado mayorista es una información de conocimiento público. Esta información es compartida por medio de instituciones reguladas que deben cumplir *la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*.

Respecto a los datos obtenidos durante el desarrollo de este proyecto serán de tipo privado pero podrán ser compartidos con otras instituciones si hay una motivación justificada. Se ha tomado esta decisión como manera de proteger la inversión en tiempo y dinero así como el esfuerzo y talento de los integrantes del grupo de economía.

3.6. Distribución de los datos y preservación.

En caso de existir una razón justificada para la distribución de los datos, esta se realizará por medio del protocolo de transferencia de archivos *FTP*. Para ello se compartirá con el cliente un *usuario/password* y una dirección *host*.

El informe del proyecto será compartido en UCrea. El repositorio institucional de la Universidad de Cantabria.

3.6.1. Software Empleado

A continuación se enumera el software requerido para la realización del proyecto.

1. Red Hat 7.1 para el servidor.
2. Bacula systems para el sistema de gestión de copias de seguridad.
3. DiskStation Manager para el sistema operativo de los NAS.
4. Python 3 para programar las rutinas requeridas.
5. SQL para el manejo de la base de datos.

3.6.2. Hardware Empleado

Por último, se enumera el hardware requerido.

1. NAS: Synology Diskstation DS220j NAS System 2-Bay. 166€/Unidad IVA exc.
2. Server: Agile S - i3 Ikoula 21st anniversary. 35€/Mes IVA exc.
3. Discos duros: WD Red- Disco duro para NAS, 1 TB. 72€/Unidad IVA exc.
4. Anemómetros: Anemómetro PCE-ADL 11. 149€/Unidad IVA exc.

Si necesario un aumento de los recursos computacionales se puede adquirir un servidor cloud con un hardware mas exigente.

3.7. Responsabilidad en la gestión de los datos y sus fuentes.

Marcos Cobo Carrillo. Responsable de la calidad de los datos y su análisis.

José Miguel Fraile Muñoz. Responsable del diagrama de Gantt y su ejecución.

Roldán Gutiérrez García. Responsable de la curación, limpieza de datos y obtención de las fuentes.

Szymon Piotr Bramowicz. Responsable del archivo de los datos.

Marco Antonio Melgarejo Aragón. Responsable de la producción de metadatos.

Pablo Calatayud Pelayo. Responsable de la gestión del plan de datos y su preservación.

4. Diagrama de Gantt

En cuanto a la planificación del proyecto, fijaremos unos objetivos para las diferentes tareas. Cada uno de estos objetivos tendrá una duración temporal dependiendo de su complejidad y necesidad. Así mismo, cada una de estas tareas se dividirá en subapartados para obtener un subobjetivo más conciso. En este proyecto se dispone de 6 grandes tareas con diferentes duraciones a lo largo del tiempo. La primera será el *Management Plan*, en el cual se definirán la estructura organizativa del proyecto, se tomarán las decisiones de Almacenamiento, Requerimientos legales o la información administrativa. Además, también cuenta con dos subapartados, el primero de ellos *Metadata Definition* donde se expondrá la Documentación y calidad de los datos y el segundo que es el *Preservation Plan* donde se define el almacenamiento y copias de seguridad durante el proyecto. Estas tareas agrupadas dentro del *Data Management Plan* tienen una duración de 3 semanas y se realizarán simultáneamente.

El segundo apartado sería la colección de datos, en la cual tendremos una primera fase de implementación de nuestro Hardware para la obtención de datos tarea que durará 4 semanas. Además, tenemos dos tareas que se realizarán simultáneamente, el servicio de almacenamiento y la integración de todo nuestro software. Por último la recogida de datos se realizará de forma continuada desde el inicio del proyecto hasta su finalización.

Con respecto a la curación de datos será uno de las tareas más laboriosas y entre la exploración de los datos y su curado tienen una duración objetivo de 9 semanas.

La tarea de Análisis tiene una duración de 6 semanas para el análisis y elaboración de informe concluyente.

La publicación del proyecto se realizará en el repositorio Ucrea de la Universidad de Cantabria.

Por último, la preservación de los datos garantiza su autenticidad, fiabilidad, utilidad e integridad y por otro lado, confirma su curación, validación y correcta preservación de los metadatos. Además de facilitar la compatibilidad entre estructuras de datos y ficheros.

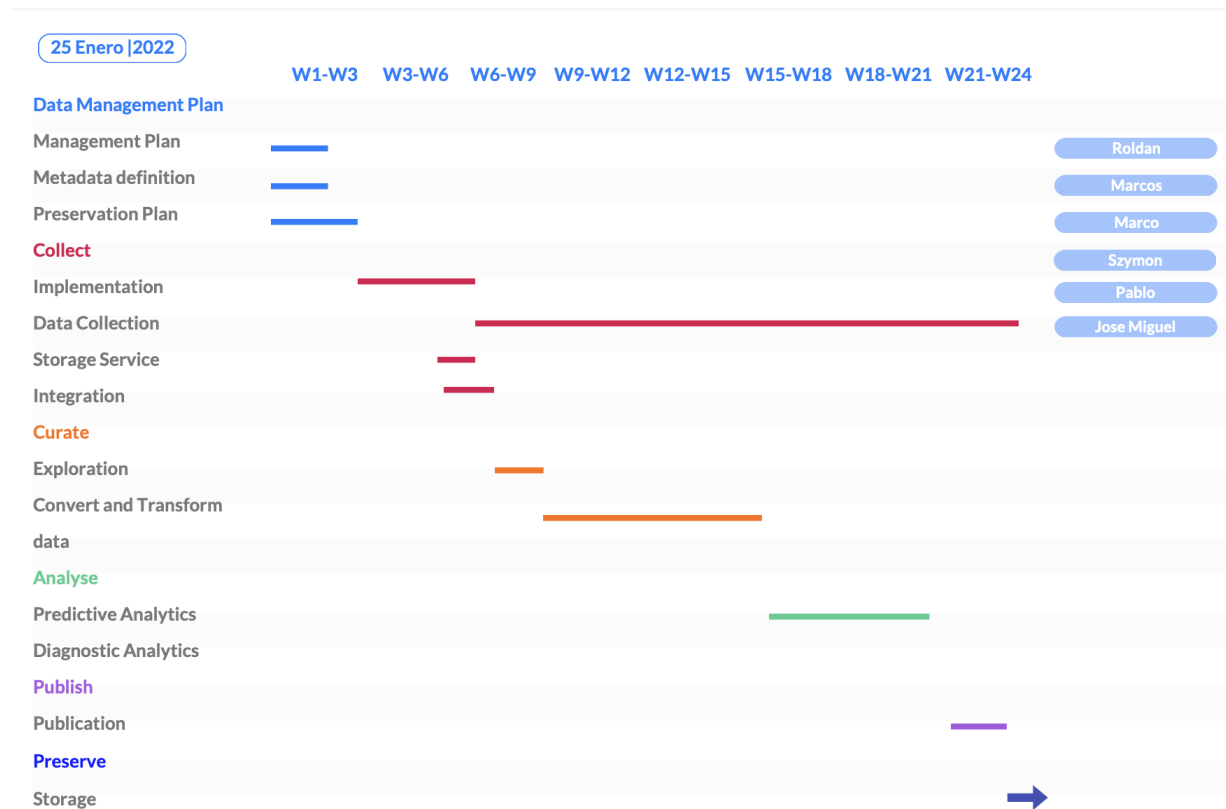


Figura 1: Diagrama de Gantt

5. Descripción de las fuentes de datos

Tenemos las siguientes fuentes de datos:

1. API de REEE - ESIOs
2. Datos históricos (dispositivos anteriores)
3. Datos propios (de nuestros dispositivos)

Por un lado, la API de REE-ESIOs ofrece datos actualizados a diario de diversos indicadores relacionados con el sector eléctrico, como son datos de generación por tecnología, consumo por banda, previsiones de consumo de REE, pagos por capacidad, etc. En nuestro caso nos interesan los precios del mercado diario (mayorista) en €/MWh, obtenidos por OMIE y actualizados en la página pública de REE diariamente alrededor de las 14.00. Se trata de una base de datos en la que los datos han sido validados varias veces, primero por OMIE antes de su publicación y luego por REE en su almacenado y posterior publicación, por lo que cabe esperar que no contengan errores. Aún así, es necesario realizar ETL sobre los mismos para obtener un dato que se pueda utilizar.

Para poder utilizar la API es necesario solicitar previamente un token personal por correo. Toda la documentación referente a la API se puede consultar en la siguiente dirección: <https://api.esios.ree.es/>. El json en cuestión es como sigue:

Item	Descripción	Formato
fecha	fecha de la medida	dd/mm/aaaa
veloc.media	velocidad media del viento (km/h)	3*n-3*n
racha.max	velocidad máxima del viento (km/h)	3*n-3*n
hora.racha	hora en la que se produce la racha máxima de viento	HH:SS

```

1 {
2   "indicator":{
3     "name":"Precio mercado SPOT Diario",
4     "short_name":"Mercado SPOT",
5     "id":600,
6     "composited":false,
7     "step_type":"linear",
8     "disaggregated":true,
9     "magnitud":[
10      {
11        "name":"Precio",
12        "id":23
13      }
14    ],
15    "tiempo":[
16      {
17        "name":"Hora",
18        "id":4
19      }
20    ],
21    "geos":[
22      {
23        "geo_id":3,
24        "geo_name":"Española"
25      }
26    ],
27    "values_updated_at":"2022-01-24T13:52:27.000+01:00",
28    "values":[
29      {
30        "value":179.97,
31        "datetime":"2022-01-24T01:00:00.000+01:00",
32        "datetime_utc":"2022-01-24T00:00:00Z",
33        "tz_time":"2022-01-24T00:00:00.000Z",
34        "geo_id":3,
35        "geo_name":"Española"
36      }
37    ]
38  }
39 }

```

Por otro lado, disponemos de los datos arrojados por los anemómetros y guardados en cada módem, para luego estar disponibles en el servidor FTP común. El formato de estos datos es un fichero *.csv, cuyo nombre es *nombreprovincia_wind_data_fecha.csv*, siendo el *nombreprovincia* el distintivo para los datos de cada zona. Dentro, aparece una línea csv con los siguientes datos: *fecha,veloc.media,racha.max,hora.racha*.

acoruna_wind_data.csv	
1	FECHA;Veloc. Media (Km/h);Racha Max (Km/h);Hora Racha
2	01/01/2015;12.96;33.12;8:10

Figura 2: CSV file

Al tratarse de un CSV, no tiene reglas de tipado, por lo que está sujeto a errores que habrá que controlar y subsanar en el proceso de curado.

6. Curar/limpiar datos

Por un lado obtenemos datos de REEE - ESIOS a través de su API: <https://api.esios.ree.es/>.

Nos interesa filtrar por el indicador 600 (*Precio mercado SPOT Diario*), que devuelve los precios del día siguiente (d+1) tanto para España como para otros países europeos en €/MWh:

```

1 {
2   "geos": [
3     {
4       "geo_id": 1,
5       "geo_name": "Portugal"
6     },
7     {
8       "geo_id": 2,
9       "geo_name": "Francia"
10    },
11    {
12      "geo_id": 3,
13      "geo_name": "Espana"
14    },
15    {
16      "geo_id": 8824,
17      "geo_name": "Reino Unido"
18    },
19    {
20      "geo_id": 8825,
21      "geo_name": "Italia"
22    },
23    {
24      "geo_id": 8826,
25      "geo_name": "Alemania"
26    },
27    {
28      "geo_id": 8827,
29      "geo_name": "Belgica"
30    },
31    {
32      "geo_id": 8828,
33      "geo_name": "Paises Bajos"
34    }
35  ]
36 }
```

Por tanto, filtramos por el *geo_id* 3 (España). Los datos devueltos para cada hora tienen la siguiente estructura dentro del fichero json:

```

1 {
2     "value":180.7,
3     "datetime":"2022-01-21T00:00:00.000+01:00",
4     "datetime_utc":"2022-01-20T23:00:00Z",
5     "tz_time":"2022-01-20T23:00:00.000Z",
6     "geo_id":3,
7     "geo_name":"Espana"
8 }

```

El primer problema al que nos enfrentamos a la hora de procesar cada hora es en qué formato queremos tener la fecha y la hora, ya que se presenta básicamente en formato *UTC +1/+2* y en *UTC*, ambos formatos separados por el valor literal *T*. En primer lugar, debemos limpiar ambos literales, *T* y *Z* (indica la hora 0 UTC), para poder dar un uso futuro a las fechas en diversas aplicaciones. Vamos a utilizar, por un lado, la hora UTC como formato para la fecha-hora de cara a facilitar la lectura por parte de posibles herramientas que puedan necesitar leer los datos que procesamos, ya que se trata de un formato universal. Por otro lado, conviene a su vez transformar fecha-hora a un formato más fácil de leer y de entender por humanos, ya sea simplemente al buscar un dato o para facilitar la elaboración de informes. Para ello vamos a transformar el valor dentro de la etiqueta *datetime* (es UTC +1 o +2) en tres columnas: *fecha*, *hora* y *bandera* (0 para invierno, 1 para verano). Para ello tenemos el siguiente código que corre automáticamente todos los días a las 14.10:

```

import requests
import json
import pandas as pd
import datetime as dt
import sqlalchemy

def indicator_gather(base_url, token, indicator, start_date, end_date, geo_id):
    """
    Based on given parameters, fetches data using REE ESIOS API and returns
    dates and values of given indicator.

    :param base_url: curl from where data is gathered
    :type base_url: str
    :param token: esios ree api token
    :type token: str
    :param indicator: esios indicator for wanted data
    :type indicator: str
    :param start_date: starting date of wanted data
    :type start_date: datetime.date
    :param end_date: ending date of wanted data
    :return: pandas.DataFrame containing requested data for each datetime
    """

    headers = {"Accept": "application/json; application/vnd.esios-api-v1+json",
               "Content-Type": "application/json",
               "Host": "api.esios.ree.es",
               "Authorization": f"Token token={token}",
               "Cookie": ""}

    # converting datetime.date to str
    start_date = start_date.strftime("%Y-%m-%d")
    end_date = end_date.strftime("%Y-%m-%d")

    # setting up parameters

```

```

params = f"?start_date={start_date}T00%3A00%3A00Z&end_date={end_date}" \
        f"T00%3A00%3A00Z&time_trunc=hour&geo_ids[]={geo_id}"

# fetching
r = requests.get(base_url + indicator + params, headers=headers)

print(r.status_code)

# gathering data
price_dict = {} # empty dictionary to store data
if r.status_code == 200:
    data = json.loads(r.text)
    index = 0 # index to store the data
    for el in data["indicator"]["values"]:
        if el["geo_id"] == 3:
            price_dict[index] = [el["datetime"], el["datetime_utc"], el["value"]]
            index += 1

# creating pandas DF from dictionary
df = pd.DataFrame.from_dict(price_dict,
                            orient="index",
                            columns=["datetime", "datetime_utc", "value"])

# replacing datetime column
df["datetime"] = df["datetime"].apply(lambda x: pd.to_datetime(x,
                        yearfirst=True).tz_convert('Europe/Amsterdam'))

return df

def insert_into_db(df, table):
    """
    Creates db table if it doesn't exist and updates values from given df.

    :param df: dataframe with curated data to insert into specified table
    :type df: pandas.DataFrame
    """
    engine = sqlalchemy.create_engine('mysql+pymysql://root:DLC2022@127.0.0.1:3306\
                                      /master_data_science')

    df_to_insert = df.copy()

    if table == "da_price":
        # creating table if doesn't exist:
        try:
            query = "CREATE TABLE IF NOT EXISTS da_price(" \
                    "datetime_utc DATETIME, " \
                    "sistema CHAR(2), " \
                    "fecha DATE NOT NULL, " \
                    "hora INT NOT NULL, " \
                    "bandera INTEGER NOT NULL, " \
                    "precio FLOAT NOT NULL, " \
                    "fecha_actualizacion DATETIME NOT NULL, " \
                    "PRIMARY KEY (datetime_utc, sistema))"
            engine.execute(query)
        except exc.SQLAlchemyError as e:
            error = str(e.__dict__["orig"])

```

```

        print(error)

if table == "wind_data":
    # creating table if doesn't exist:
    try:
        query = "CREATE TABLE IF NOT EXISTS wind_data(" \
            "fecha DATE, " \
            "zona VARCHAR(20), " \
            "vel_km_h FLOAT NOT NULL, " \
            "vel_m_s FLOAT NOT NULL, " \
            "vel_mph FLOAT NOT NULL, " \
            "vel_nudos FLOAT NOT NULL, " \
            "racha_max_km_h FLOAT NOT NULL, " \
            "racha_max_m_s FLOAT NOT NULL, " \
            "racha_max_mph FLOAT NOT NULL, " \
            "racha_max_nudos FLOAT NOT NULL, " \
            "hora_racha TIME, " \
            "fecha_actualizacion DATETIME NOT NULL, " \
            "PRIMARY KEY (fecha, zona))"

        engine.execute(query)
    except exc.SQLAlchemyError as e:
        error = str(e.__dict__['orig'])
        print(error)

df_to_insert["fecha_actualizacion"] = dt.datetime.now()
for i in range(len(df_to_insert)):
    try:
        df_to_insert.iloc[i:i + 1].to_sql(name=table, if_exists='append',
            con=engine, index=False)
    except:
        print("Duplicated key, skipping.")
        pass

if __name__ == "__main__":

    token = "28783fb5c499f634c81d9e1644cb7f46a05def938f799a054a62292ded53ee12"
    base_url = "https://api.esios.ree.es/indicators/"
    indicator = "600"
    start_date = dt.date.today() + dt.timedelta(days=-1)
    end_date = dt.date.today() + dt.timedelta(days=1)
    geo_id = 3

    df = indicator_gather(base_url, token, indicator, start_date, end_date, geo_id)

    # adding date and time columns
    df["fecha"] = pd.to_datetime(df["datetime"], utc=False)
    df["fecha"] = df["fecha"].dt.date
    df["hora"] = pd.to_datetime(df["datetime"], utc=False)
    df["hora"] = df["hora"].dt.time.apply(lambda x: x.hour + 1).astype(int)

    # adding system column
    df["sistema"] = "ES"

    # adding flag (summer = 1, winter = 0)
    df.loc[df["datetime"].dt.strftime('%d/%b/%Y:%H:%M:%S %Z').str[-3:-2] == "1",

```

```

        "bandera"] = 0
df.loc[df["datetime"].dt.strftime('%d/%b/%Y:%H:%M:%S %Z').str[-3:-2] == "2",
        "bandera"] = 1

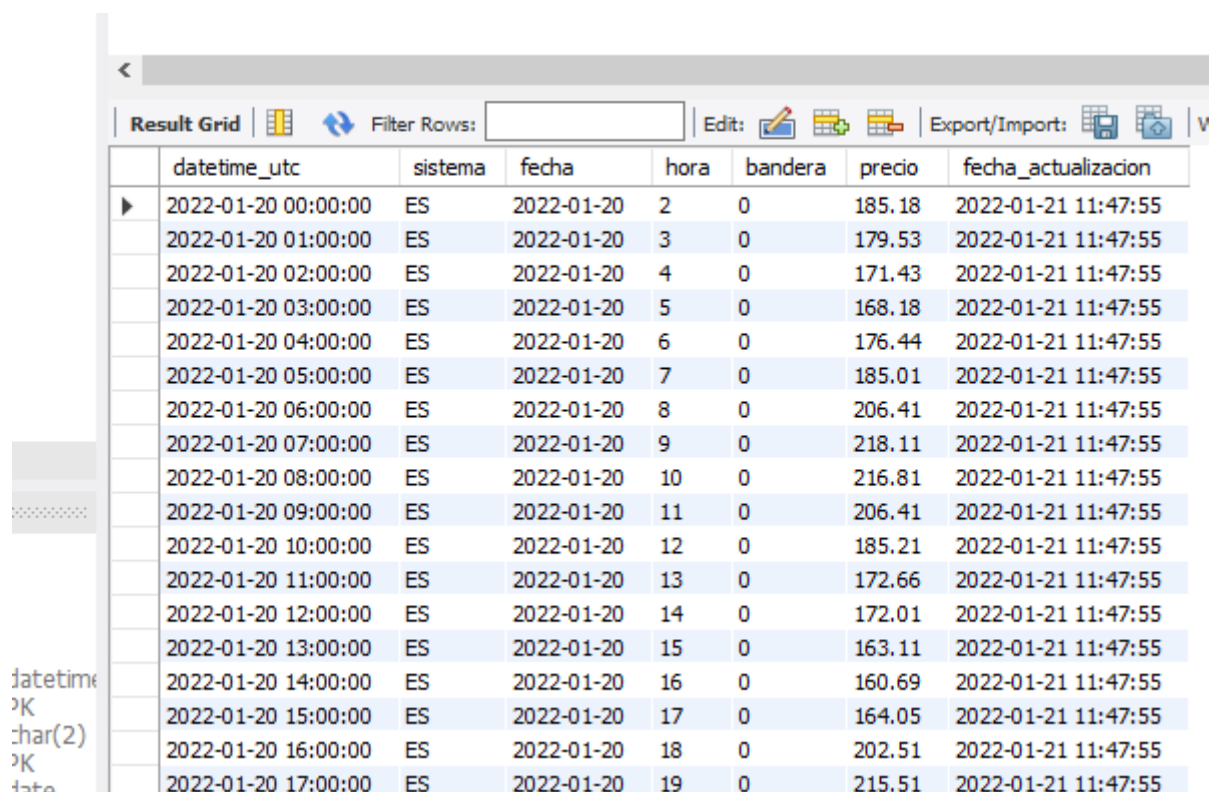
# rearranging columns
df = df[["datetime_utc", "sistema", "fecha", "hora", "bandera", "value"]]
df.rename(columns={"value": "precio"}, inplace=True)

# removing literals from UTC datetime
df["datetime_utc"] = pd.to_datetime(df["datetime_utc"],
                                    format="%Y-%m-%dT%H:%M:%S%fZ")

# calling function to update db
insert_into_db(df, "da_price")

```

El resultado una vez curado el dato y almacenado en BBDD es el siguiente:



	datetime_utc	sistema	fecha	hora	bandera	precio	fecha_actualizacion
▶	2022-01-20 00:00:00	ES	2022-01-20	2	0	185.18	2022-01-21 11:47:55
	2022-01-20 01:00:00	ES	2022-01-20	3	0	179.53	2022-01-21 11:47:55
	2022-01-20 02:00:00	ES	2022-01-20	4	0	171.43	2022-01-21 11:47:55
	2022-01-20 03:00:00	ES	2022-01-20	5	0	168.18	2022-01-21 11:47:55
	2022-01-20 04:00:00	ES	2022-01-20	6	0	176.44	2022-01-21 11:47:55
	2022-01-20 05:00:00	ES	2022-01-20	7	0	185.01	2022-01-21 11:47:55
	2022-01-20 06:00:00	ES	2022-01-20	8	0	206.41	2022-01-21 11:47:55
	2022-01-20 07:00:00	ES	2022-01-20	9	0	218.11	2022-01-21 11:47:55
	2022-01-20 08:00:00	ES	2022-01-20	10	0	216.81	2022-01-21 11:47:55
	2022-01-20 09:00:00	ES	2022-01-20	11	0	206.41	2022-01-21 11:47:55
	2022-01-20 10:00:00	ES	2022-01-20	12	0	185.21	2022-01-21 11:47:55
	2022-01-20 11:00:00	ES	2022-01-20	13	0	172.66	2022-01-21 11:47:55
	2022-01-20 12:00:00	ES	2022-01-20	14	0	172.01	2022-01-21 11:47:55
	2022-01-20 13:00:00	ES	2022-01-20	15	0	163.11	2022-01-21 11:47:55
	2022-01-20 14:00:00	ES	2022-01-20	16	0	160.69	2022-01-21 11:47:55
	2022-01-20 15:00:00	ES	2022-01-20	17	0	164.05	2022-01-21 11:47:55
	2022-01-20 16:00:00	ES	2022-01-20	18	0	202.51	2022-01-21 11:47:55
	2022-01-20 17:00:00	ES	2022-01-20	19	0	215.51	2022-01-21 11:47:55

Figura 3: MySQL DB. Table "da_price"

En cuanto a los datos obtenidos de nuestros dispositivos en los distintos puntos, nos enfrentamos a varios problemas. Algunas horas son incongruentes, como se puede ver en la siguiente imagen:

```

13  12/01/2015;19.00;72.07;10.00
14  13/01/2015;21.96;56.88;13:20
15  14/01/2015;16.92;63;20:20
16  15/01/2015;19.08;66.96;6:40
17  16/01/2015;0;81;75:50:00
18  17/01/2015;0;0;
19  18/01/2015;20.88;86.04;82:00:00

```

Figura 4: Error horas (incongruencia)

En realidad, lo que ocurre es que el dato ha quedado mal guardado, siendo el caso de ejemplo la hora 7:55:00. En la línea 19 se puede observar que ha vuelto a ocurrir, siendo en ese caso las 08:20:00.

Se trata de la hora en que se ha producido la racha máxima de viento. Este pasa a ser el primer error a solucionar, desplazando hacia la derecha los números siempre que la hora aparezca en formato *HH:MM:SS*, ya que en las horas que aparece el dato correcto el formato es *HH:MM*.

Por otro lado, existen campos en los que la hora de racha máxima aparece como "Varias". Convertiremos este valor en null para poder procesar los datos en una tabla en base de datos, ya que si se establece el tipo "time" para la columna no va a aceptar valores de formato "varchar/char":

124	03/05/2015;11.88;65.16;2:50
125	04/05/2015;20.88;100.08;Varias
126	05/05/2015;0;0;

Figura 5: Error horas (char)

Por último, necesitamos distinguir en qué zona se ha tomado la medida, incluida en el nombre de los ficheros, por lo que tendremos que extraer esta zona en incluirla en el dataset previo al procesado en BBDD. También nos interesa poner la velocidad del viento en diferentes unidades, ya que originalmente sólo viene en km/h y dependiendo del uso que se vaya a dar en el futuro es útil tenerlo en distintos formatos. El código final queda como sigue:

```
import pandas as pd
import datetime as dt
import sqlalchemy
import os
import glob

def insert_into_db(df, table):
    """
    Creates db table if it doesn't exist and updates values from given df.

    :param df: dataframe with curated data to insert into specified table
    :type df: pandas.DataFrame
    """
    engine = sqlalchemy.create_engine('mysql+pymysql://root:DLC2022@127.0.0.1:3306\
                                        /master_data_science')

    df_to_insert = df.copy()

    if table == "da_price":
        # creating table if doesn't exist:
        try:
            query = "CREATE TABLE IF NOT EXISTS da_price(" \
                    "datetime_utc DATETIME, " \
                    "sistema CHAR(2), " \
                    "fecha DATE NOT NULL, " \
                    "hora INT NOT NULL, " \
                    "bandera INTEGER NOT NULL, " \
                    "precio FLOAT NOT NULL, " \
                    "fecha_actualizacion DATETIME NOT NULL, " \
                    "PRIMARY KEY (datetime_utc, sistema))"
            engine.execute(query)
        except exc.SQLAlchemyError as e:
            error = str(e.__dict__['orig'])
            print(error)

    if table == "wind_data":
        # creating table if doesn't exist:
        try:
```

```

query = "CREATE TABLE IF NOT EXISTS wind_data(" \
        "fecha DATE, " \
        "zona VARCHAR(20), " \
        "vel_km_h FLOAT NOT NULL, " \
        "vel_m_s FLOAT NOT NULL, " \
        "vel_mph FLOAT NOT NULL, " \
        "vel_nudos FLOAT NOT NULL, " \
        "racha_max_km_h FLOAT NOT NULL, " \
        "racha_max_m_s FLOAT NOT NULL, " \
        "racha_max_mph FLOAT NOT NULL, " \
        "racha_max_nudos FLOAT NOT NULL, " \
        "hora_racha TIME, " \
        "fecha_actualizacion DATETIME NOT NULL, " \
        "PRIMARY KEY (fecha, zona))"
engine.execute(query)
except exc.SQLAlchemyError as e:
    error = str(e.__dict__['orig'])
    print(error)

df_to_insert["fecha_actualizacion"] = dt.datetime.now()
for i in range(len(df_to_insert)):
    try:
        df_to_insert.iloc[i:i + 1].to_sql(name=table, if_exists='append',
            con=engine, index=False)
    except:
        print("Duplicated key, skipping.")
        pass

def format_hour(val):
    """
    Transforms hour in HH:MM:SS format to HH:MM, when it has wrong values like 82:10:00
    that should be 08:21

    :param val: value to be transformed
    :type val: str
    :return: transformed value (str)
    """
    if val != "nan" and val is not None:
        if len(val.split(":")) == 3:
            val_list = val.split(":")
            val = f"{val_list[0][0]}:{val_list[0][-1]}{val_list[1][0]}"

    return val

if __name__ == "__main__":

    dir_path = r"...\\wind_data"
    csv_files = glob.glob(os.path.join(dir_path, "*.csv"))

    dfl = [] # empty list to store all generated pandas DFs
    for file in csv_files:
        df = pd.read_csv(file, sep=";", parse_dates=["FECHA"])
        zona = (file.split("\\")[-1]).split("_")[0]
        df["Hora Racha"] = df["Hora Racha"].astype(str)

```

```

df["zona"] = zona
dfl.append(df)

df = pd.concat(dfl) # merging all dfs in the list into one

# replacing "varios" for null
df.replace({"Varias": None}, inplace=True)

# renaming columns
df.rename(columns={"FECHA": "fecha",
                  "Veloc. Media (Km/h)": "vel_km_h",
                  "Racha Max (Km/h)": "racha_max_km_h",
                  "Hora Racha": "hora_racha"},
          inplace=True)

# fixing hour error
df["hora_racha"] = df["hora_racha"].apply(lambda x: format_hour(x))

# adding columns with values in m/s, mph and knots
df["vel_m_s"] = df["vel_km_h"].apply(lambda x: x / 3.6)
df["vel_mph"] = df["vel_km_h"].apply(lambda x: x / 0.44704)
df["vel_nudos"] = df["vel_km_h"].apply(lambda x: x / 0.514444)
df["racha_max_m_s"] = df["racha_max_km_h"].apply(lambda x: x / 3.6)
df["racha_max_mph"] = df["racha_max_km_h"].apply(lambda x: x / 0.44704)
df["racha_max_nudos"] = df["racha_max_km_h"].apply(lambda x: x / 0.514444)

# rearranging columns before updating db
cols = df.columns.tolist()
cols = cols[:1] + cols[4:5] + cols[1:2] + cols[5:8] + cols[2:3] + cols[8:] + cols[3:4]
df = df[cols]

# calling function to update db
insert_into_db(df, "wind_data")

```


El resultado final, una vez finalizado tanto el proceso de curado/limpieza de los datos y el procesado en BBDD queda como sigue:

fecha	zona	vel_km_h	vel_m_s	vel_mph	vel_nudos	racha_max_km_h	racha_max_m_s	racha_max_mph	racha_max_nudos	hora_racha	fecha_actualizacion
2021-12-03	acoruna	5.04	1.4	11.2742	9.79698	24.12	6.7	53.9549	46.8856	13:50:00	2022-01-21 17:41:30
2021-12-03	albacete	11.16	3.1	24.9642	21.6933	41.04	11.4	91.8039	79.7754	14:00:00	2022-01-21 17:44:31
2021-12-03	almeria	3.96	1.1	8.85827	7.69763	24.84	6.9	55.5655	48.2851	18:00:00	2022-01-21 17:44:31
2021-12-03	leon	14.04	3.9	31.4066	27.2916	51.12	14.2	114.352	99.3694	10:20:00	2022-01-21 17:44:31
2021-12-03	zaragoza	12.96	3.6	28.9907	25.1922	37.08	10.3	82.9456	72.0778	12:40:00	2022-01-21 17:44:31
2021-12-04	acoruna	6.12	1.7	13.6901	11.8963	33.12	9.2	74.0873	64.3802	00:50:00	2022-01-21 17:41:30
2021-12-04	albacete	11.88	3.3	26.5748	23.0929	28.08	7.8	62.8132	54.5832	16:40:00	2022-01-21 17:44:31
2021-12-04	almeria	9	2.5	20.1324	17.4946	38.88	10.8	86.9721	75.5767	12:20:00	2022-01-21 17:44:31
2021-12-04	leon	10.08	2.8	22.5483	19.594	36	10	80.5297	69.9785	14:30:00	2022-01-21 17:44:31
2021-12-04	zaragoza	9	2.5	20.1324	17.4946	24.84	6.9	55.5655	48.2851	17:00:00	2022-01-21 17:44:31
2021-12-05	acoruna	6.12	1.7	13.6901	11.8963	51.84	14.4	115.963	100.769	03:40:00	2022-01-21 17:41:30
2021-12-05	albacete	15.12	4.2	33.8225	29.391	46.08	12.8	103.078	89.5724	15:50:00	2022-01-21 17:44:31
2021-12-05	almeria	6.84	1.9	15.3006	13.2959	28.08	7.8	62.8132	54.5832	17:20:00	2022-01-21 17:44:31
2021-12-05	leon	23.04	6.4	51.539	44.7862	66.96	18.6	149.785	130.16	05:50:00	2022-01-21 17:44:31
2021-12-05	zaragoza	20.88	5.8	46.7072	40.5875	68.04	18.9	152.201	132.259	13:30:00	2022-01-21 17:44:31
2021-12-06	acoruna	12.96	3.6	28.9907	25.1922	42.12	11.7	94.2198	81.8748	04:00:00	2022-01-21 17:41:30
2021-12-06	albacete	10.08	2.8	22.5483	19.594	41.04	11.4	91.8039	79.7754	15:30:00	2022-01-21 17:44:31
2021-12-06	almeria	12.96	3.6	28.9907	25.1922	48.96	13.6	109.52	95.1707	20:10:00	2022-01-21 17:44:31

Figura 6: MySQL DB. Table "wind_data"

Estos datos a su vez se comparten a través de zenodo para cumplir los principios FAIR, teniendo el siguiente DOI:

DOI 10.5281/zenodo.5900902

Figura 7: DOI

January 24, 2022
Dataset Open Access
Edit
New version
Communities
power data
Remove
43 views
16 downloads
See more details...
Indexed in
OpenAIRE
Publication date:
January 24, 2022
DOI:
DOI 10.5281/zenodo.5900902
Keyword(s):
Power Price Wind
Related identifiers:
Previous versions
10.5281/zenodo.5898982 (Dataset)
Communities:
power data
License (for files):
Creative Commons Attribution 4.0 International

Preview

datetime_utc	sistema	fecha	hora	bandera	precio	fecha_actualizacion
2022-01-25 00:00:00	ES	2022-01-25	2	0	212.9	2022-01-24 17:24:29
2022-01-24 23:00:00	ES	2022-01-25	1	0	220.27	2022-01-24 17:24:29
2022-01-24 22:00:00	ES	2022-01-24	24	0	215.02	2022-01-24 17:24:29
2022-01-24 21:00:00	ES	2022-01-24	23	0	235.02	2022-01-24 17:24:29
2022-01-24 20:00:00	ES	2022-01-24	22	0	250	2022-01-24 17:24:29
2022-01-24 19:00:00	ES	2022-01-24	21	0	274.98	2022-01-24 17:24:29
2022-01-24 18:00:00	ES	2022-01-24	20	0	297.33	2022-01-24 17:24:29
2022-01-24 17:00:00	ES	2022-01-24	19	0	265	2022-01-24 17:24:29
2022-01-24 16:00:00	ES	2022-01-24	18	0	240	2022-01-24 17:24:29
2022-01-24 15:00:00	ES	2022-01-24	17	0	231.77	2022-01-24 17:24:29

Files (5.8 MB)

Name	Size
da_price.csv	4.1 MB
md5b3af99828cf02b985e82f817fddfb3d	
precio_vel_2015.png	45.7 kB
md529388b88abb02c8c15c4ec22bebbf4b	
precio_vel_2016.png	50.7 kB

Figura 8: Files at Zenodo

De esta forma se encuentra publicado tanto los datasets de partida como los datos de análisis, pudiendo ser reproducidos por cualquiera.

7. Creación de metadatos Dublin Core

En este apartado, se proporcionan los metadatos de los datos con los que se van a trabajar en el presente proyecto. Tales metadatos son de tipo descriptivo ya que definen e identifican objetos digitales, además de dotar elementos que hacen que los datos sean localizables. Estarán basados en Dublin Core, un modelo estándar cuyos términos se exponen en el siguiente enlace: <https://www.dublincore.org/specifications/dublin-core/dcmi-terms/>. Así, se procede a describir una colección, o agregación de recursos, la cual es una agrupación de los dos *datasets*, para después describir a estos últimos de forma separada, tal y como especifica la iniciativa Dublin Core.

Colección

Title= "Colección de datos sobre el precio de luz y la velocidad del viento diarios en 5 provincias de España."

Type= "Collection"

Subject= "Economía"; "Velocidad del viento"; "España"; "Mercado"; "Precio de la luz"; "Energías Renovables"

Accrual Method= "Protocolo FTP"

Accrual Periodicity= "Diario"

Date= "22/01/2022"

Rights= "Privado"

Dataset de los precios de la electricidad.

Title= "Precios diarios de la electricidad del sistema eléctrico Español"

Creator= "Grupo Economía DLC"

Subject= "Economía"; "Electricidad"; "España"; "Mercado"; "Precio de la luz."

Description= "Precio, por hora, de la luz (por MWh) en España"

Publisher= "ESIOS-REE"

Contributor= "OMIE"

Date= "22/01/2022"

Type= "Dataset"

Format= "1831555 observables y 7 variables"

Format= "datetime_utc, sistema, fecha, hora, bandera, precio, fecha_actualizacion"

Format= ".db, tamaño variable (3.52MB hasta ahora)"

Identifier= "ID:0.1109/5.771073"

Source= "https://www.esios.ree.es/es" [ESIOS/REE]

Language= "sp"

Relation= "Colección de datos sobre el precio de luz y la velocidad del viento diarios en 5 provincias de España." [Colección que engloba a ambos datasets del presente trabajo.]

Coverage= "De 01-01-2015 hasta actualmente"

Coverage= "España"

Rights= "Público"

Dataset de las velocidades del viento.

Title= "Datos diarios de la velocidad del viento en distintas unidades de medida tomados en 5 provincias españolas."

Creator= "Grupo Economía DLC"

Subject= "Velocidad del viento"; "Racha de viento"; "España"; "Energías renovables"; "Precio de la luz."

Description= "Velocidades del viento, media diaria y racha máxima, en las siguientes cinco parques eólico de España, ubicados en las provincias: La Coruña, Albacete, Almería, León y Zaragoza."

Publisher= "Grupo Economía DLC"

Date= "22/01/2022"

Type= "Dataset"
Format= "1831555 observables y 12 variables."
Format= "fecha, zona, vel_km_h, vel_m_s, vel_mph, vel_nudos, racha_max_km_h, racha_max_m_s, racha_max_mph, racha_max_nudos, hora_racha, fecha_actualizacion"
Format= ".db, tamaño variable (2.52MB actual)"
Identifier= "10.5281/zenodo.500902"
Source= "Dispositivos de medición instalados en los siguientes parques eólicos: Puerto de San Isidro (León), Hellín (Albacete), Monte Iroite (A Coruña), Abia (Almería), Burjaraloz (Zaragoza)."
Language= "sp"
Relation= "Colección de datos sobre el precio de luz y la velocidad del viento diarios en 5 provincias de España." [Colección que engloba a ambos datasets del presente trabajo.]
Coverage= "De 01-01-2015 hasta actualmente"
Coverage= "España"
Rights= "Privado"

Estos metadatos permiten que los datos cumplan con los principios FAIR (*FINDABLE*, *ACCESIBLE*, *INTEROPERABLE*, *REUSABLE* y, además, *REPRODUCIBLE*), apoyando así a que estos se utilicen y publiquen de manera justa (*FAIR* en inglés).

8. Plan de preservación

El plan de preservación tiene como objetivo la creación de un protocolo para asegurar el almacenamiento y credibilidad de los datos. Por un lado, garantiza su autenticidad, fiabilidad, utilidad e integridad y por otro lado, confirma su curación, validación y correcta preservación de los metadatos. Por último, facilita la compatibilidad entre estructuras de datos y ficheros.

En nuestro caso particular disponemos de dos bases de datos. Una relativa al precio del kWh en el mercado eléctrico español y otra con la velocidad del viento en 5 ciudades españolas.

En la tabla de la velocidad del viento se añaden 5 datos con 12 campos cada día. En la tabla del precio se añade cada día 24 datos correspondientes a cada hora con 7 campos cada uno. El impacto que tienen estos datos en la memoria es de un aumento de 14592 bytes o 0.001824 MB al día. Por otro lado la tabla con la velocidad del viento requiere de 2.52 MB y la tabla con los precios ocupa 3.52 MB.

Para el sistema de back up se propone el software libre Bacula. Es un software de *backup* compatible con dispositivos UNIX/LINUX y que ofrece una gran versatilidad y control. También hay que añadir a los costes de memoria de las bases de datos los scripts utilizados para el análisis y modelaje, así como los papers e informes obtenidos a consecuencia del proyecto. Estos también se encontrarán en la plataforma de control de versiones GitHub, pero para garantizar una mayor independencia se irán guardando en nuestro entorno.

La programación de las backups tendrán la siguiente estructura. Se realizarán *backup* incrementales todos los días a las 23:00 UTC y backup completas toda las semanas los domingos a las 01:00 UTC.

Para el almacenamiento se propone el uso de dos NAS con diferentes geolocalizaciones para aumentar la seguridad y prevenir catástrofes físicas, así como un almacenamiento de la última copia de seguridad en *GitHub*. El modelo de NAS elegido es el *Synology Diskstation DS220j NAS System 2-Bay*. Cada NAS contará con 2 discos duros de 1TB y serán una copia el uno del otro. El acceso al servidor será a través de *SSH* con el software *Putty* desde dispositivos *UNIX/LINUX* para acceder a la terminal y a través de *x11* para establecer conexiones remotas con el servidor. Se creará un entorno *SMB* para que usuarios con dispositivos *Windows* puedan acceder a sesiones remoto en el servidor. Para la recuperación de *jobs* en *Bacula* se puede usar tanto su interfaz como la terminal de *UNIX*.

Si en el largo plazo hay una mayor exigencia de recursos de memoria por requisitos propios del proyecto, la escalabilidad de los recursos memorísticos se puede realizar incrementando la capacidad de los discos duros contenidos en los NAS.

9. Análisis de los datos y conclusiones

En esta última sección del trabajo llevaremos a cabo un pequeño estudio de los datos recolectados. Para ello, crearemos un entorno de trabajo en Python, dónde previamente extraeremos los datos de velocidad del viento y precio del MWh de nuestra base de datos y, posteriormente, obtendremos un par de gráficos en los que apoyarnos a la hora de establecer conclusiones.

Recordando el objetivo del proyecto, el cual es estudiar si existe o no una relación entre la velocidad del viento y el precio del MWh en el mercado Español, lo primero que vamos a hacer es representar una variable frente a la otra, para todo el histórico de datos que poseemos.

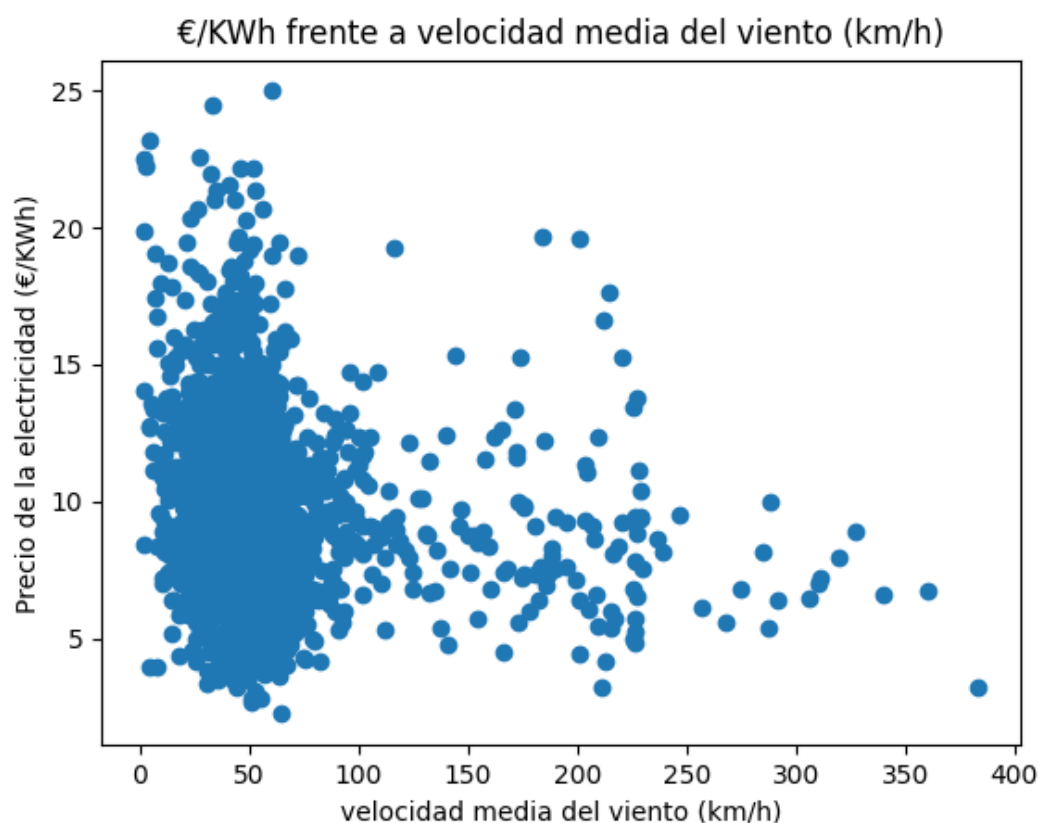


Figura 9: €/KWh frente a velocidad media del viento (km/h)

Parece erróneo pensar que existe una correlación entre la variable precio y la variable velocidad media del viento, de hecho, el coeficiente de correlación de Pearson para estas dos variables es de entorno a -0.15, lo cual indica una correlación demasiado débil como para sacar ninguna conclusión.

Pero tenemos que tener en cuenta que este histórico abarca desde 2015 hasta la actualidad, y que cada día están cobrando una mayor fuerza las energías renovables, entre ellas, la eólica. Por ello, vamos a estudiar la relación en algunos años. A continuación se muestran los gráficos relativos a los años 2018 y 2020, junto con sus coeficientes de correlación:

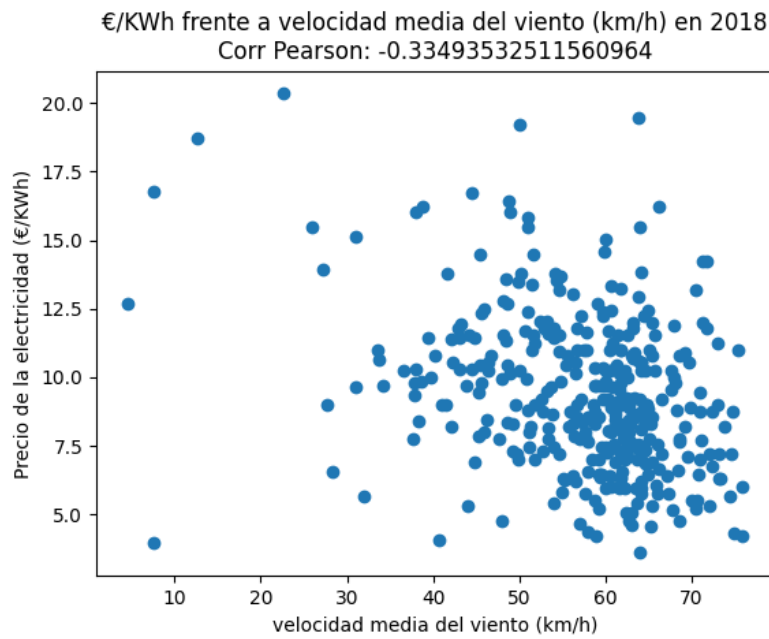


Figura 10: €/KWh frente a velocidad media del viento (km/h) 2018

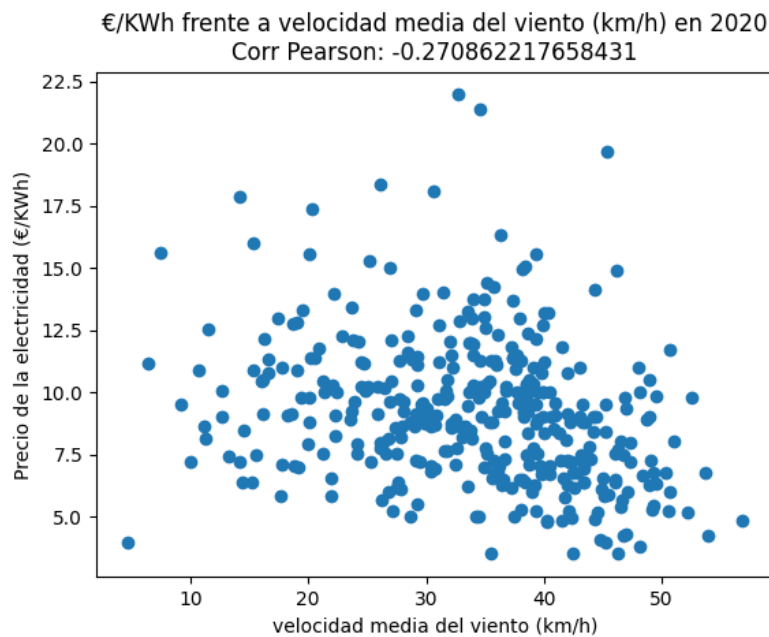


Figura 11: €/KWh frente a velocidad media del viento (km/h) 2020

En todos ellos se intuye cierta correlación negativa. Vemos como la nube de puntos sigue una tendencia oblicua que se acentúa a medida que las velocidades eólicas aumentan, es decir, cuanto más fuerte sopla el viento, con mayor fuerza se acumulan los puntos en zonas de precios bajos.

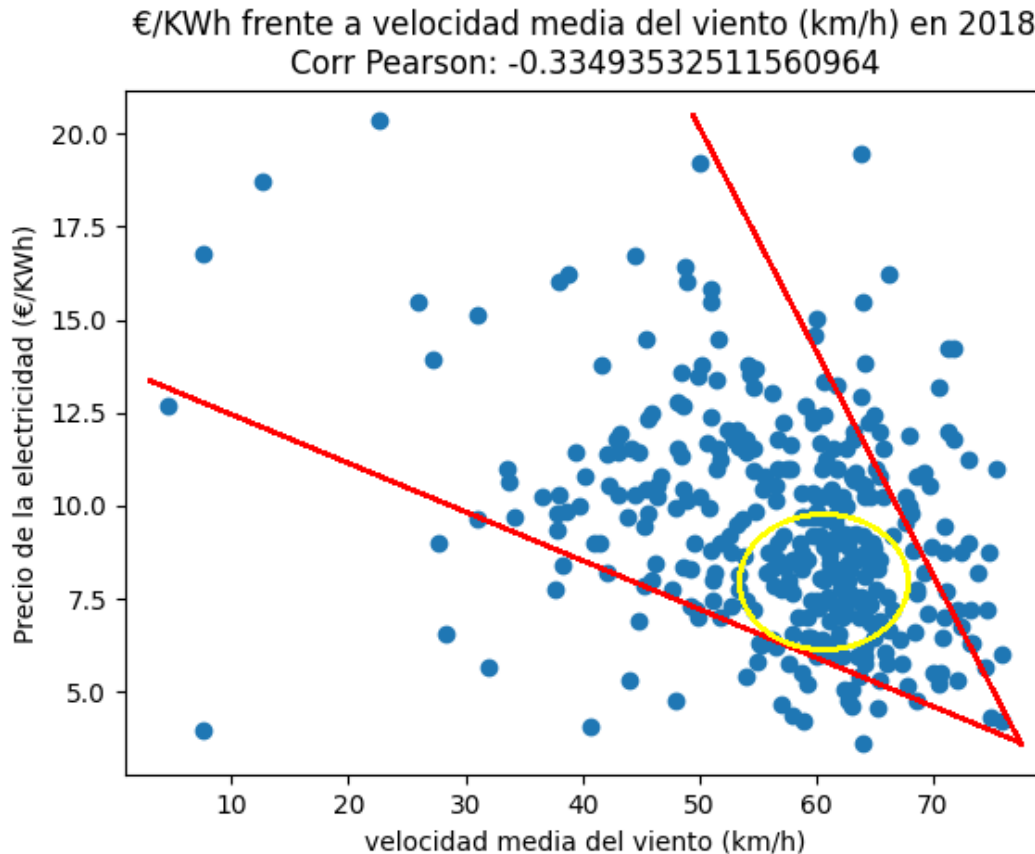


Figura 12: €/KWh frente a velocidad media del viento (km/h) 2018 explicada

En la actualidad no tenemos datos suficientemente concluyentes como para poder afirmar con total certeza que la velocidad del viento influye en el mercado eléctrico, pero sí nos dejan entrever una ligera correlación.

Cabe recordar que el precio del MWh depende de muchos otros factores, además de otras muchas fuentes de energía aparte de la eólica. Por este motivo es normal que el precio de la electricidad este amortiguado en días de mucho viento. Aún así, como ya hemos mencionado, hay momentos en los que si se puede ver cierta relación entre estas dos variables, por lo que, como comentábamos al comienzo de este trabajo, en determinadas circunstancias la energía eólica predomina en el mercado por encima del resto, y eso es una gran noticia, no solo económica, sino también social.

A continuación detallamos el código que hemos empleado en esta sección para el estudio anterior.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt

from sqlalchemy.ext.automap import automap_base
from sqlalchemy.orm import Session
from sqlalchemy import create_engine
from sqlalchemy.sql import func

Base = automap_base()

# engine, suppose it has many tables
engine = create_engine('mysql+pymysql://root:DLC2022@127.0.0.1:3306/master_data_science')
```

```

# reflect the tables
Base.prepare(engine, reflect=True)

# mapped classes are now created with names by default
# matching that of the table name.
DaPrice = Base.classes.da_price
WindData = Base.classes.wind_data
session = Session(engine)

rset = (session.query(DaPrice.fecha.label("fecha"),
                      func.min(DaPrice.precio).label("precio_min"),
                      func.avg(DaPrice.precio).label("precio_avg"),
                      func.max(DaPrice.precio).label("precio_max"),
                      func.avg(WindData.vel_km_h).label("vel_avg"),
                      func.max(WindData.racha_max_km_h).label("racha_max")).
      filter(WindData.fecha == DaPrice.fecha)).group_by(DaPrice.fecha).all()

# print(list(rset))

fecha = [i[0] for i in rset]
min_precio = [i[1] for i in rset]
avg_precio = [i[2] for i in rset]
max_precio = [i[3] for i in rset]
avg_wind = [i[4] for i in rset]
max_wind = [i[5] for i in rset]

df = pd.DataFrame(
    {"fecha": fecha,
     "precio_min": min_precio,
     "precio_avg": avg_precio,
     "precio_max": max_precio,
     "vel_avg": avg_wind,
     "racha_max": max_wind,
    })

print(np.corrcoef(df["precio_avg"], df["vel_avg"])[0,1])
plt.scatter(df["precio_avg"], df["vel_avg"])
plt.ylabel("Precio de la electricidad (€/KWh)")
plt.xlabel("velocidad media del viento (km/h)")
plt.title('€/KWh frente a velocidad media del viento (km/h)')
plt.savefig('precio_vel.png'); plt.show()

years = ['2015', '2016', '2017', '2018', '2019', '2020', '2021', '2022']
for i in range(len(years) - 1):
    precio = df["precio_avg"].loc[(df["fecha"] >= years[i]) & (df["fecha"] < years[i+1])]
    vel = df["vel_avg"].loc[(df["fecha"] >= years[i]) & (df["fecha"] < years[i+1])]
    corr_pearson = np.corrcoef(precio, vel)[0,1]

    plt.scatter(precio, vel)
    plt.ylabel("Precio de la electricidad (€/KWh)")
    plt.xlabel("velocidad media del viento (km/h)")
    plt.title('€/KWh frente a velocidad media del viento (km/h) en ' + years[i] +
              '\nCorr Pearson: {}'.format(corr_pearson))
    plt.savefig('precio_vel_' + years[i] + '.png'); plt.show()

```