TRABAJOS FIN DE GRADO

Universidad de Jaén

Facultad de Ciencias Experimentales

HERRAMIENTAS DE SOFTWARE PARA LA QUÍMICA

Autor: Pablo Rísquez Cañada

Grado: Química

Fecha: 07/06/2024

Resumen:

Se ha llevado a cabo una extracción de datos físico-químicos de internet con técnica de webscraping con scripts de Python y se ha hecho un gráfico con los datos.

Se ha realizado un gráfico interactivo con Python que nos permite visualizar un diagrama líquido-vapor cambiando el parámetro de la temperatura en tiempo real.

Se ha diseñado un gráfico interactivo con Python de un diagrama líquido-vapor tal que se nos permite ver los datos relativos a la regla de la palanca en tiempo real, con el ratón.

Abstract:

Data on physicochemical properties has been extracted from the internet using web scraping techniques with Python scripts, and a graph has been created with the data.

An interactive graph has been made with Python that allows us to visualize a liquid-vapor diagram, changing the temperature parameter in real-time.

An interactive graph of a liquid-vapor diagram has been designed with Python, which allows us to see data related to the lever rule in real-time, using the cursor.

Preámbulo.

El presente trabajo de fin de grado (TFG) supone el colofón al Grado en Química. En él se plantea la utilidad de las herramientas de software, particularmente de la programación de código en lenguaje Python para el recién graduado en Química o químico profesional.

La elección de este tema se ve motivada por un interés en aprender programación, así como de un trabajo personal y conocimientos previos sobre el tema. Este trabajo no podría haberse hecho de otra manera, no hubiera sido posible sin un bagaje personal al respecto. En el Grado en Química no se enseña ni se usa Python.

El trabajo consiste en scripts (piezas de código ejecutables) con utilidades concretas que sirven de demo técnica de qué puede hacer Python para el químico a un nivel personal y como introducción a las posibilidades del lenguaje a nivel profesional de la química y el ámbito general.

Estos scripts pudieran ser una introducción a la programación en Python para un estudiante de química. No obstante, no pretenden servir como curso o tutorial al respecto.

Todos los scripts han sido escritos por y para el máster, son de autoría personal.

Por su complejidad estimamos que se situán más bien en algún punto del camino entre un conocimiento sólido de las bases y el uso profesional, como el que se puede enseñar en un Máster.

Este TFG se ha visto constreñido por la limitación de su extensión por ser un TFG. No todos los scripts hechos para este TFG se incluyen en él, así como una revisión de los distintos tipos de software en la industria que también queda fuera en favor de los scripts escogidos.

El código aquí presente y sus explicaciones no pretende ni debe de ser tomado como un ejemplo de buenas prácticas de programación. Esto es un trabajo de química, no de informática. Este trabajo y su desarrollo no ha tenido ninguna lectura ni revisión por ningún otro programador más que el autor.

En la sección 1 se incluye una introducción comentando sobre el contexto en que se realiza este trabajo, el advenimiento y popularización de las IAs generativas.

En la sección 2 se realiza una extracción de datos de internet con técnica de webscraping con scripts de Python.

En la sección 3 se usa Python para crear un widget que nos permite visualizar un diagrama líquido-vapor cambiando el parámetro de la temperatura en tiempo real.

En la sección 4 se grafica un diagrama líquido-vapor tal que se nos permite ver los datos relativos a la regla de la palanca en tiempo real, con el ratón.

Agradecimientos:

A mis profesores Cristóbal Cara Corpas y Priscilla Rocío Bautista, a los cuales visité con preguntas y consultas durante el transcurso de este trabajo.

También a los profesores Miguel Moreno Carretero , Justo Cobo Domingo, Juan Francisco García Reyes , Francisco Partal Ureña y Tomás Peña Ruiz que me respondieron uno o más correos al respecto.

A Pedro Garrancho García y Consuelo Rosales Cárdenas, tutores de este TFG.

A mis amigos, a los que mandaba capturas de imágenes cuando me funcionaban los scripts.

A aquel alumno de informática que sentado en mi mesa que le pregunté si sabía por qué aquello me daba error

1.Introducción:

No tendría sentido realizar un trabajo de fin de grado sobre software hoy en 2024 sin mencionar al elefante en la habitación. La inteligencia artificial o inteligencias artificiales. Hemos decidido comenzar hablando de ello porque, queramos o no, va a moldear la economía global, nuestro futuro y con ello nuestra industria y disciplina; la química.

Todo lo que escribamos o leamos sobre software hoy va a estar afectado por el advenimiento de la IA, incluida la revisión del software para con la química que haremos más adelante aunque estos softwares sean anteriores a la IA generativa actual y los lenguajes de programación predominantes hoy.

Es un tema enorme que se nos escapa, pero sí podemos hablar desde la perspectiva del graduado en química que quiere medrar profesionalmente y tiene interés por la programación. Nuestra perspectiva (naíf o no, el tiempo lo dirá) es que el software va a generalizarse y será herramienta y lenguaje común del científico y de todo profesional, como ya hoy lo son el inglés, las matemáticas y la informática. Hemos de integrarnos en esta corriente, no quedarnos atrás.

En su momento la aparición de diccionarios, enciclopedias y calculadoras no sustituyeron a lingüistas, profesores ni matemáticos.

Tampoco lo hicieron los softwares de traducción, el buscador de Google ni las hojas de cálculo.

De esta manera tampoco nos dejará obsoletos la IA generativa ni ninguna forma de machine learning, deep learning o cualquier tipo de algoritmo ni base de datos.

Pero sí cambiará nuestra forma de trabajar, forzándonos a trabajar de otras maneras más productiva y eficientes para usar estas herramientas y que no nos sustituyan. Actualizarse o morir.

Creemos que limitarnos a hacer uso de las herramientas públicas (ChatGPT, Google Copilot,etc) no es suficiente. El universitario de hoy habrá de ser parte de los avances del futuro. Habrá de participar en el desarrollo. Tendrá que aprender a programar y programar.

Nuestro lenguaje de elección ha sido Python. No ha sido una elección arbitraria, aunque no es propósito de este trabajo cantar las virtudes de este lenguaje en particular.

Por decir algo sólido y con peso: ChatGPT está escrito en Python.

Me gustaría aclarar que no obstante no es por eso, que yo empecé a aprender Python allá por 2020 cuando aún no había ChatGPT. He vivido el antes y el después.

Me complace la idea de que este TFG pueda ser representativo del *zeitgeist* de este momento, de este tiempo de cambio.

Sea el mío otro testimonio de que la IA no va a trabajar por nosotros; sino darnos herramientas para producir más y con ello exigirnos a formarnos más y ser mejores y más productivos. No se puede ignorar.

También es un antes y después para los trabajos escolares y universitarios, y con ello los TFGs.

Durante la realización de este trabajo yo desconocía el límite máximo de páginas y; aparte de los scripts (porque iba a haber scripts sí o sí, de hecho el script de la regresión lineal se basa en algo que yo ya tenía hecho y es lo que motivó la elección de este trabajo), planteé hacer una revisión bibliográfica de usos de software en la química para hablar de ello antes de hablar de mis scripts de creación propia.

Esta parte no se incluye en el TFG final porque no cabe y no estaba entre las prioridades. 25 páginas tenía hechas. Me hubiera gustado incluir aquí mi experiencia usando el ChatGPT para redactar textos.

Quiero decir; que este trabajo no lo ha escrito el ChatGPT. Aunque quizá pudiera haberme escrito un TFG válido. Sí se ha usado como consulta para la programación.

Usando Google, Github y ChatGPT para consultas resulta muy difícil hacer una bibliografía relativa al código. No la hay. Que el código se justifique por sí mismo en tanto funcionando.

Mi referencia más sólida es el curso *Automate the Boring Stuff with Python* de *Al Sweigart*, la primera mitad. Lo recomiendo como curso para empezar con Python.

2.Extracción de datos físico-químicos con Webscraping.

Queremos hacer un plot con unos datos físico-químicos.(*proton affinity, gas basicity*)

Del inglés *Plot*

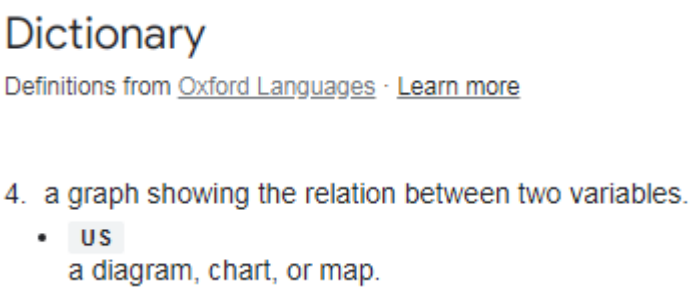


Gráfico que muestra la relación entre dos variables. Diagrama, cuadro o mapa.

Decidimos hacer un plot de la afinidad protónica. (Proton affinity).

Escogemos este parámetro por ser uno no en exceso estudiado o conocido, que puede ser de mayor interés plotear en lugar de algo más manido como la electronegatividad o afinidad electrónica.

Proton affinity:

The negative of the enthalpy change in the gas phase reaction (real or hypothetical) between a proton (more appropriately hydron) and the chemical species concerned, usually an electrically neutral species to give the conjugate acid of that species. Proton affinity is often, but unofficially, abbreviated as PA.

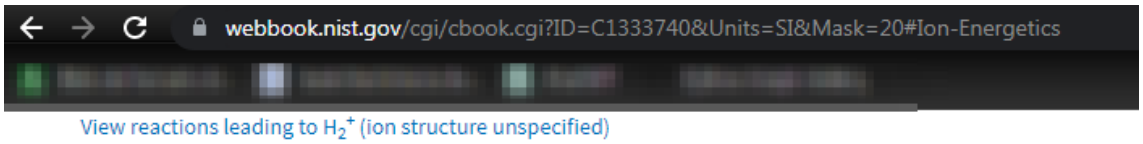
'proton affinity' in *IUPAC Compendium of Chemical Terminology*, 3rd ed. International Union of Pure and Applied Chemistry; 2006. Online version 3.0.1, 2019. https://doi.org/10.1351/goldbook.P04907

El negativo del cambio de entalpía en la reación fase gas (real o hipotética) entre un protón (o más propiamente un hidrón, H+) y la especie química correspondiente, normalmente una especie eléctricamente neutra para dar el ácido conjugado de la especie. (Abreviada como PA en inglés de manera informal)

Buscamos por internet y encontramos datos relacionados en la página del NIST

https://webbook.nist.gov/cgi/cbook.cgi?ID=C1333740&Units=SI&Mask=20#Ion-Energetics

Estos datos son del hidrógeno:



View reactions leading to H_2^+ (ion structure unspecified)

Quantity	Value	Units	Method	Reference	Comment
IE (evaluated)	15.42593 ± 0.00005	eV	N/A	N/A	*L*
Quantity	**Value**	**Units**	**Method**	**Reference**	**Comment**
Proton affinity (review)	422.3	kJ/mol	N/A	Hunter and Lias, 1998	*HL*
Quantity	**Value**	**Units**	**Method**	**Reference**	**Comment**
Gas basicity	394.7	kJ/mol	N/A	Hunter and Lias, 1998	*HL*

Figura 2.1

Necesitamos la afinidad protónica de todos y cada uno de los elementos de la tabla periódica. Pero, ¿cómo lo hacemos? ¿entramos en decenas de páginas y copiamos el dato uno a uno? No. Hacemos un webscraping. ¿Qué es el webscraping?

El webscraping se refiere a las técnicas de extracción de datos utilizada para recopilar información de páginas web de manera automatizada. En nuestro caso vamos a analizar el código HTML de una página y extraer los datos relevantes, como texto, imágenes o enlaces, mediante la utilización de scripts o programas.

¿Qué es el HTML?

HTML o HyperText Markup Language es el lenguaje de marcas de las páginas web. Un lenguaje de marcado o lenguaje de marcas es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

¿Qué es un script?

El script es la serie de instrucciones que le damos a nuestro programa Python y que es ejecutable. Los archivos .py que usamos aquí son nuestros scripts.

Vamos a hacer un script que abra múltiples páginas y vaya extrayendo el dato del código HTML que nosotros le indicamos.

Primero necesitamos definir qué páginas queremos scrapear (hacerle webscraping, extraer datos de ella)

De la página

https://webbook.nist.gov/cgi/cbook.cgi?ID=C1333740&Units=SI&Mask=20#Ion-Energetics

observamos que en el enlace aparece el código CAS del elemento, en este caso del hidrógeno. (lo hemos subrayado). En cada página de cada elemento varía esa parte de la ID.

La lista con todos los elementos y sus números CAS sí es algo fácil de encontrar. Lo metemos en un Excel:

	atomic number	A weight	Name	Sym	M.P	B.P.	Density	Earth	Group	Electron Configuration		
											CAS	CAS
Hydrogen (H) - CAS Number: 1333-74-0	1	1.008	Hydrogen	H	-259	-253	0.09	0.14	1	$1s^1$	1333-74-0	1333740
Helium (He) - CAS Number: 7440-59-7	2	4.003	Helium	He	-272	-269	0.18		18	$1s^2$	7440-59-7	7440597
Lithium (Li) - CAS Number: 7439-93-2	3	6.941	Lithium	Li	180	1,347	0.53		1	[He] $2s^1$	7439-93-2	7439932
Beryllium (Be) - CAS Number: 7440-41-7	4	9.012	Beryllium	Be	1,278	2,97	1.85		2	[He] $2s^2$	7440-41-7	7440417
Boron (B) - CAS Number: 7440-42-8	5	10.811	Boron	B	2,3	2,55	2.34		13	[He] $2s^2$ $2p^1$	7440-42-8	7440428
Carbon (C) - CAS Number: 7440-44-0	6	12.011	Carbon	C	3,5	4,827	2.26	0.09	14	[He] $2s^2$ $2p^2$	7440-44-0	7440440
Nitrogen (N) - CAS Number: 7727-37-9	7	14.007	Nitrogen	N	-210	-196	1.25		15	[He] $2s^2$ $2p^3$	7727-37-9	7727379
Oxygen (O) - CAS Number: 7782-44-7	8	15.999	Oxygen	O	-218	-183	1.43	46.71	16	[He] $2s^2$ $2p^4$	7782-44-7	7782447
Fluorine (F) - CAS Number: 7782-41-4	9	18.998	Fluorine	F	-220	-188	1.70	0.03	17	[He] $2s^2$ $2p^5$	7782-41-4	7782414
Neon (Ne) - CAS Number: 7440-01-9	10	20.180	Neon	Ne	-249	-246	0.90		18	[He] $2s^2$ $2p^6$	7782-41-4	2023453

Figura 2.2

Definimos una columna adicional en Excel en la que cambiamos la parte del enlace del código por la del sendo elemento que le corresponda. Obtenemos una columna que tiene todos los enlaces de la página de datos de energías de ión del NIST

=CONCATENAR("https://webbook.nist.gov/cgi/cbook.cgi?ID=C";P5;"&Units=SI&Mask=20#Ion-Energetics")

M	N	O	P	Q
tron Configuration				
		CAS	CAS	web
		1333-74-0	1333740	https://webbook.nist.gov/cgi/cbook.cgi?ID=C1333740&Units=SI&Mask=20#Ion-Energetics
		7440-59-7	7440597	https://webbook.nist.gov/cgi/cbook.cgi?ID=C7440597&Units=SI&Mask=20#Ion-Energetics
$2s^1$		7439-93-2	7439932	https://webbook.nist.gov/cgi/cbook.cgi?ID=C7439932&Units=SI&Mask=20#Ion-Energetics
$2s^2$		7440-41-7	7440417	https://webbook.nist.gov/cgi/cbook.cgi?ID=C7440417&Units=SI&Mask=20#Ion-Energetics
$?p^1$		7440-42-8	7440428	https://webbook.nist.gov/cgi/cbook.cgi?ID=C7440428&Units=SI&Mask=20#Ion-Energetics
$?p^2$		7440-44-0	7440440	https://webbook.nist.gov/cgi/cbook.cgi?ID=C7440440&Units=SI&Mask=20#Ion-Energetics

Figura 2.3

Como hemos hablado previamente Python puede leer Excel, pero no es el formato ideal para almacenar datos (aunque a esta escala nos valdría). Transformamos la hoja de cálculo a formato JSON.

JSON (JavaScript Object Notation, pronounced /ˈdʒeɪsən/ or /ˈdʒeɪˌsɒn/) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values)

JSON o json es un formato de archivo estándar abierto y un formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos de datos que consisten en pares y matrices de atributo-valor (u otros valores serializables).

Lo usamos como un archivo de almacenamiento de datos, sabemos cómo referenciarlo y extraer datos de él con Python.

Usamos una página web que nos lo hace:

https://products.aspose.app/cells/conversion/excel-to-json

Transforma nuestra hoja de Excel en un archivo json.

Entendemos que para documentos sensibles no debamos usar cualquier plataforma para hacer esto, pero para este trabajo esta página nos vale.

El formato json es peor para visualizar los datos como tablas pero permite más posibilidades (las cuales no explotamos en este TFG, pero sí conocemos aunque sean tan simples como que el json permite almacenar muchos más datos que al trabajarlos en Excel en nuestro portátil de 2009 hacen colgarse el sistema)

Así se ve el documento json, abierto dentro de PyCharm, la terminal de Python de nuestra elección.

```
{
"Hoja1":[
 {
  "Column15": "CAS",
  "Column16": "CAS",
  "web": "https:\/\/webbook.nist.gov\/cgi\/cbook.cgi?ID=C1333740&Units=SI&Mask=20#Ion-Energetics"
 },
 {
  "Column2": "Hydrogen (H) - CAS Number: 1333-74-0",
  "Column3": 1,
  "A weight": 1008,
  "Name": "Hydrogen",
  "Sym": "H",
  "M.P": -259,
  "B.P.": -253,
  "Density": "0.09",
  "Earth": "0.14",
  "Group": 1,
  "Electron Configuration": "1s1",
  "Column15": "1333-74-0",
  "Column16": 1333740,
  "web": "https:\/\/webbook.nist.gov\/cgi\/cbook.cgi?ID=C7440597&Units=SI&Mask=20#Ion-Energetics"
```


Este documento json lo metemos en la misma carpeta donde tengamos el script para que pueda leerlo.

Comenzamos con el script:

```
import json
import requests
from bs4 import BeautifulSoup
import re
```

A principio del código siempre deben estar las importaciones de las librerías a usar. Una librería es una serie de funciones prehechas por otros usuarios. Suponen gran parte de la potencia de Python. Esto no es lo primero que escribimos al empezar a programar pero sí debe ir al principio del script, o por lo menos previo a llamar (hacer uso de) a esas librerías. Esto es así con todo, el código no se escribió de arriba abajo; se fue construyendo más como un cuadro o dibujo, añadiendo detalles según van siendo necesarios.

Json es la librería para leer json.

Requests para abrir páginas web.

BeautifulSoup una librería de webscraping.

Re es una librería de expresiones regulares para buscar y manipular cadenas de texto.

```
def strip_non_numbers(text):
    # Use regular expression to remove everything but numbers and decimal points
    return re.sub(r'[^\d.±]', '', text)
```

Definimos una función para extraer el texto eliminando todo excepto números y puntos de decimal. Nos servirá más adelante. Una función es un trozo de código que estructuramos como función para poder usarlo múltiples veces más delante de forma conveniente.

```
def scrappo(pagee):
    # Send a GET request to the webpage

    response = requests.get(pagee)

    # Parse the HTML content using Beautiful Soup
    soup = BeautifulSoup(response.text, 'html.parser')

    # Find all occurrences of the desired text within the parsed content
    desired_text = 'Proton affinity (review)'
    #desired_text = 'Gas basicity'
    #desired_text = 'IE (evaluated)'
    found_elements = soup.find_all(text=desired_text)
```

Definimos una función que busca el texto (‘Proton affinity (review)’) en el HTML de la página.

Manda un request a la página. Recoge el texto de la página (con toda su estructura HTML, no el texto limpio). Busca el desired_text en el texto recogido. Para esto hemos tenido que inspeccionar el HTML de la página y buscar cómo aparece nuestro dato.

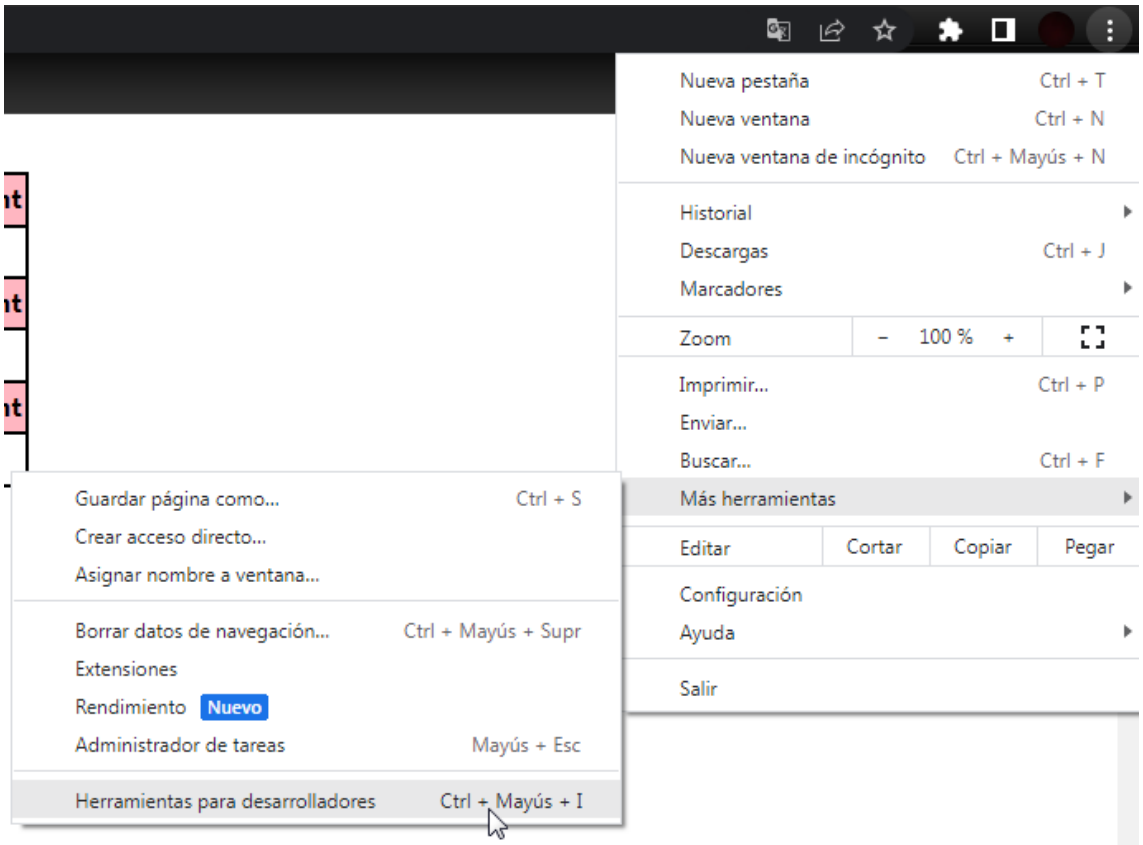


Figura 2.4

Para entrar al HTML de la página.

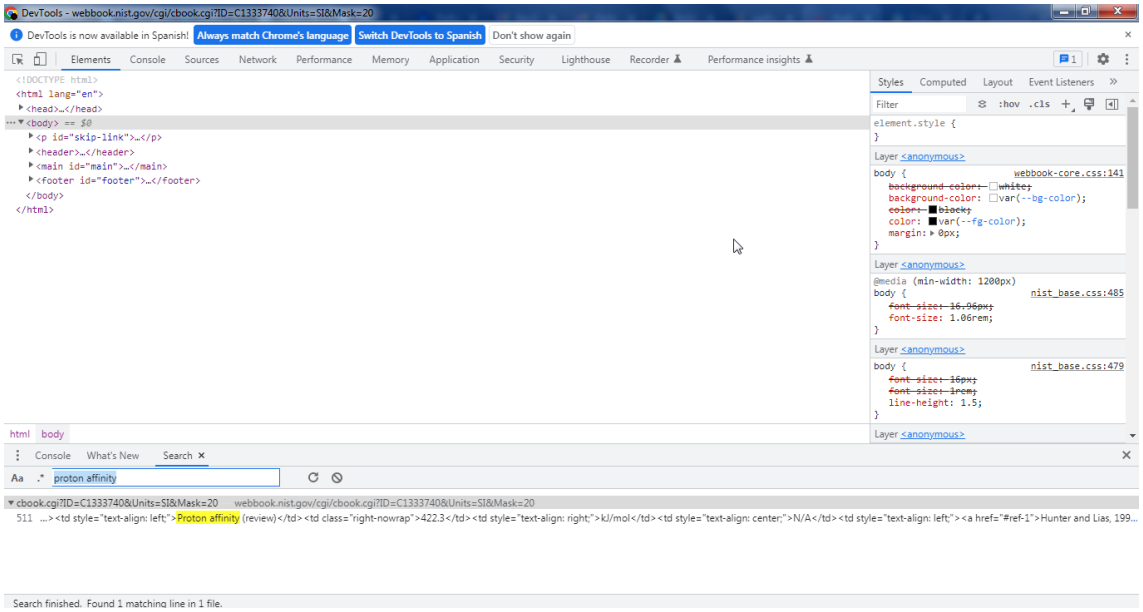


Figura 2.5

Buscamos el término ‘proton affinity’ porque es fijo en la página, el valor en concreto varía. Vamos a utilizarlo para pedirle al script que busque el elemento posterior a este, que es el valor del proton affinity.

Nuestros conocimientos de webscraping son muy básicos, pero hemos conseguido hacerlo de esta manera. El objetivo inicial no era necesariamente scrapear datos, sino hacer un plot de datos. El scraping surgió como una necesidad y resolvimos el problema.

```
    # Print the entire line and the next tag following it
    if found_elements:
        #print("Lines containing '{}' and the next tag in the webpage:".format(desired_text))
        for element in found_elements:
            # Navigate up the HTML tree to the parent element
            parent_element = element.parent
            # Print the parent element
            #print(parent_element)

            # Find the next sibling tag
            next_tag = str(parent_element.find_next_sibling())
            #next_tag2 = str(next_tag.find_next_sibling())
            if next_tag:
                value=str(strip_non_numbers(next_tag))
                print(value)
    else:
        print("--------------".format(desired_text))
```

Este código sigue dentro de la función scrappo()

Si encuentra el elemento buscado anteriormente (en algunos elementos no está recogido, es importante que el código sepa continuar si no lo encuentra) define ese elemento como el parent_element y define como next_tag el elemento posterior a ese parent.element. Si lo encuentra, lo pasa por la función que lo deja solo en valores numéricos y puntos decimales y no los devuelve, lo printea en la consola. En caso contrario de que lo encuentre (Else:) printea “-----“

```
with open('dataaa.json') as f:  # Replace 'your_json_file.json' with the path to your JSON file
    data = json.load(f)
```

Abre el json, lo carga y lo guarda como “data” para referirlo dentro de este script.

```
continua=True
ii=0
while continua:
    try:

        url = (data['Hoja1'][ii]['web'])
        ii=ii+1
        scrappo(url)
    except:
        break
```

Define una variable como True para crear un bucle infinito que sólo se rompe y termina cuando no consigue realizar lo que está dentro del Try

Se define contador ii con valor 0.

Carga la url del json, el valor ii de la columna ‘web’.

Suma uno al ii, para que luego entra a la web 1 (la 2º) y así sucesivamente.

Aplica la función scrappo a esa url. Esta función printea el valor.

Cuando ya no hay más urls el Try no se consigue y la función hace Break y acaba el script.

Podría pensarse que quizá también podría haberse hecho con un bucle For con tantas iteraciones como longitud de la columna ‘web’, pero no sabíamos cómo obtener la longitud de una “columna” en json. Lo hicimos de la manera que nos resultó más intuitivo, pero debe haber múltiples.

Veamos cómo funciona. El script en funcionamiento lo hemos dejado para el final porque no tiene ningún input manual ni de copia-pega, no es un script que tenga sentido para un usuario distinto al creador.

Figura 2.6

Van apareciendo los datos en columna uno a uno, a velocidad de uno por segundo más o menos.

Una vez el script ha terminado, seleccionamos la columna entera y la copiamos y pegamos al Excel de antes para tenerla guardada y poder usar los datos para el plot.

Ya hemos hecho un webscraping de unos datos que necesitábamos.

De igual manera que hemos hecho para la *protton affinity* también podemos obtener datos de la *gas basicity*

¿Qué posibilidades tiene esto?

En este caso hemos scrapeado unos datos estáticos de una institución. No resulta fácil buscar ejemplos en lo que se recopilen datos de internet en la investigación en química como una persona fuera de ése ámbito, pero sí sabemos que recopilar datos es una de las tareas más lucrativas para las grandes empresas.

Hemos explorado un script que abre páginas web, y esto no sólo abre la posibilidad de leer sino de escribir en la página; rellenar formularios, hacer descargas, realizar búsquedas, etc.

Con este tipo de scripts se puede:

Recopilar datos de distintas páginas web para organizar y presentar la información; por ejemplo:

- Los datos meteorológicos de distintos lugares para presentarlos juntos y trabajar con ellos.
- Comparar precios de distintas páginas.

- Analizar datos económicos y sociopolíticos de redes sociales, algo que es todo un mundo y se nos escapa.

Buscando por internet vemos:

El webscraping tiene una amplia gama de posibilidades y usos en diversos campos:

Recopilación de datos: Permite recopilar datos de múltiples fuentes en la web, como precios de productos, información de empresas, datos meteorológicos, noticias, y más. Esta información puede ser utilizada para análisis, investigación, toma de decisiones, entre otros propósitos.

Monitoreo de precios y competidores: Las empresas pueden utilizar webscraping para monitorear los precios de sus productos y los de la competencia en línea, lo que les permite ajustar sus estrategias de precios en consecuencia.

Análisis de opiniones y comentarios: Permite recopilar opiniones y comentarios de usuarios de diferentes plataformas, como redes sociales, foros o sitios de reseñas, para realizar análisis de sentimientos, identificar tendencias o evaluar la reputación de una marca.

Generación de leads: Se puede utilizar para extraer información de contacto de clientes potenciales, como direcciones de correo electrónico o números de teléfono, a partir de sitios web y bases de datos en línea.

Seguimiento de noticias: Permite recopilar y clasificar noticias de diferentes fuentes para estar al tanto de los acontecimientos actuales y tendencias en diversos campos, como política, economía, tecnología, entre otros.

Creación de conjuntos de datos para machine learning: Es útil para recopilar grandes cantidades de datos etiquetados necesarios para entrenar modelos de machine learning en áreas como reconocimiento de imágenes, procesamiento de lenguaje natural, entre otros.

Automatización de tareas: Puede utilizarse para automatizar tareas repetitivas en línea, como completar formularios, realizar búsquedas o descargar archivos de sitios web.

Investigación académica: Los investigadores pueden utilizar webscraping para recopilar datos relevantes para sus estudios, como información bibliográfica, estadísticas o resultados de encuestas, entre otros.

Existen softwares y plataformas que te facilitan este tipo de tareas de automatización y búsqueda de datos en internet. No obstante la que probamos para realizar esta tarea (Bardeen) exige pago para mandar un algoritmo a múltiples páginas.

Con estos datos obtenidos hacemos distintos plots:

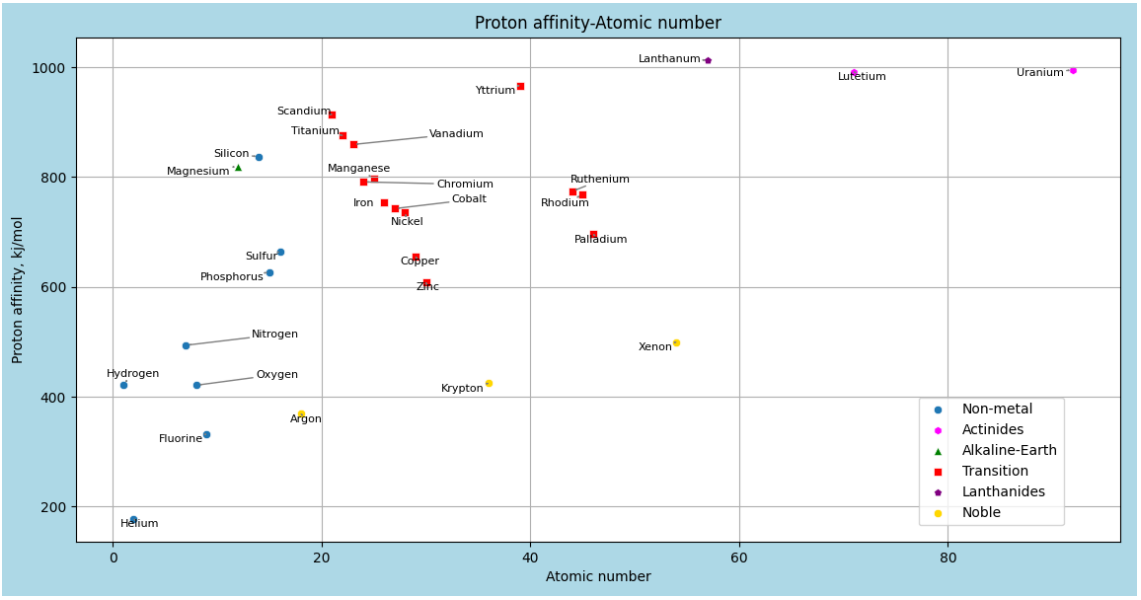


Figura 2.7

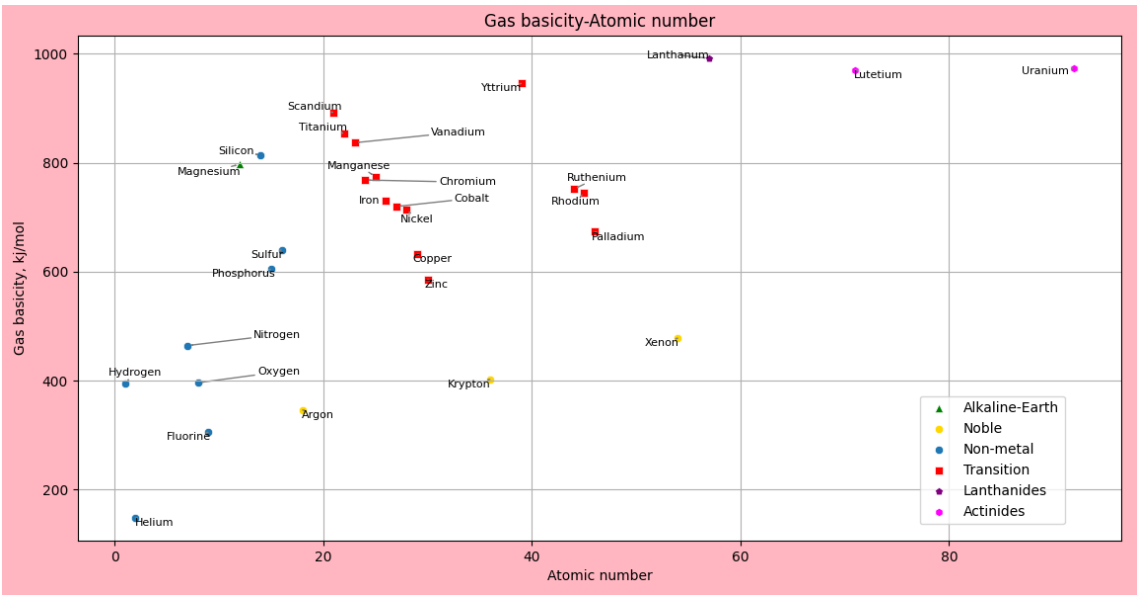


Figura 2.8

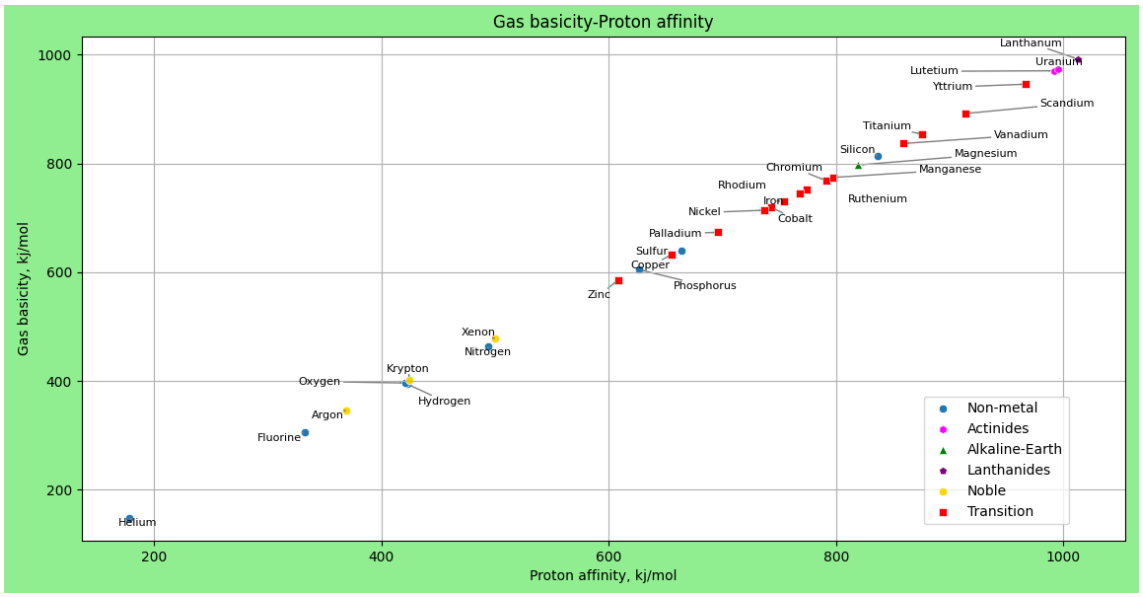

Gas basicity-Proton affinity
Gas basicity, kj/mol
Proton affinity, kj/mol
1000
800
600
400
200
200
400
600
800
1000
Lanthanum
Uranium
Lutetium
Yttrium
Scandium
Titanium
Vanadium
Silicon
Magnesium
Chromium
Manganese
Rhodium
Ruthenium
Iron
Nickel
Cobalt
Palladium
Sulfur
Copper
Phosphorus
Zinc
Xenon
Nitrogen
Krypton
Oxygen
Hydrogen
Argon
Fluorine
Helium
Non-metal
Actinides
Alkaline-Earth
Lanthanides
Noble
Transition

Figura 2.9

3.Diagrama líquido-vapor.

Nos planteamos realizar un diagrama líquido vapor en función de la P o la T.

Decidimos hacerlo de la mezcla butano-isobutano por ser una mezcla común en los hidrocarburos y sin azeótropos. Decidimos usar la ecuación de Antoine por su simplicidad y por encontrar sus parámetros en el NIST.

https://webbook.nist.gov/cgi/cbook.cgi?ID=C106978&Mask=4&Type=ANTOINE

https://webbook.nist.gov/cgi/cbook.cgi?ID=C75285&Mask=4&Type=ANTOINE

Comenzamos testeando la ecuación, relacionando la P con la T en las sustancias puras.

Veamos el código para esto. Iremos más rápido porque ya hemos hablado de los plots en la sección anterior. *(la sección del plot queda en los anexos)*

```
import seaborn as sns
import matplotlib.pyplot as plt
from adjustText import adjust_text
from statistics import mean

import numpy as np
import scipy.stats as st
from sklearn.linear_model import LinearRegression


import matplotlib.pyplot as plt
#from ipywidgets import interact, widgets
```

Librerías importadas. A menudo copiamos librerías usadas en scripts similares aunque posteriormente no las necesitemos. Por si acaso. Trabajar resulta cada vez más fácil y nos permite llegar a cosas más complejas porque nos apoyamos sobre lo anterior.

```
#Antoine equation
#log10(P) = A - (B / (T + C))
    #P = vapor pressure (bar)
    #T = temperature (K)
```

Nos dejamos apuntada la expresión. Todo lo escrito después de una # no se interpreta como código, esto sirve para hacer anotaciones.

```
# Antoine equation for butane: log10(P) = A - (B / (T + C))
# Constants for butane 272.66 - 425K
A=4.35576
B=1175.581
C=-2.071


# Antoine equation constants for isobutane 261.31 - 408.12  K
Ai = 4.3281
Bi = 1132.108
Ci = 0.918
```

Introducimos las constantes de Antoine como variables.

```
# Function to calculate vapor pressure (P) from temperature (T) using Antoine equation
def antoine(T):
    return 10**(A - (B / (T + C)))

def antoine2(T):
    return 10**(Ai - (Bi / (T + Ci)))
```

Definimos la función de Antoine para el butano y el isobutano. Es una función que devuelve la presión en función de la temperatura.

```
# Temperature range (in Celsius)
T_range = np.linspace(220,400,500)

# Calculate vapor pressures corresponding to the temperature range
P_range = antoine(T_range)
P_range2 = antoine2(T_range)
```

Introducimos el rango de temperaturas que vamos a plotear. Desde 220 hasta 400, con 500 puntos en ese intervalo.

Calculamos las presiones como la función aplicada a ese rango de temperaturas para los parámetros del butano y del isobutano (antoine2)

```
# Plotting
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
plt.plot(T_range, P_range, label="Vapor Pressure of Butane", color="blue")
plt.plot(T_range, P_range2, label="Vapor Pressure of IsoButane", color="red")
plt.xlabel("Temperature (K)")
plt.ylabel("Pressure (bar)")
plt.title("P-T Diagram for Butane and Isobutane (Antoine Equation)")
plt.legend()
plt.grid(True)
plt.gcf().set_facecolor('paleturquoise')  # Change background color of the figure
plt.gca().set_facecolor('white')  # Change background color of the inside figure

plt.show()
```

Hacemos el plot, esto ya lo explicamos antes.

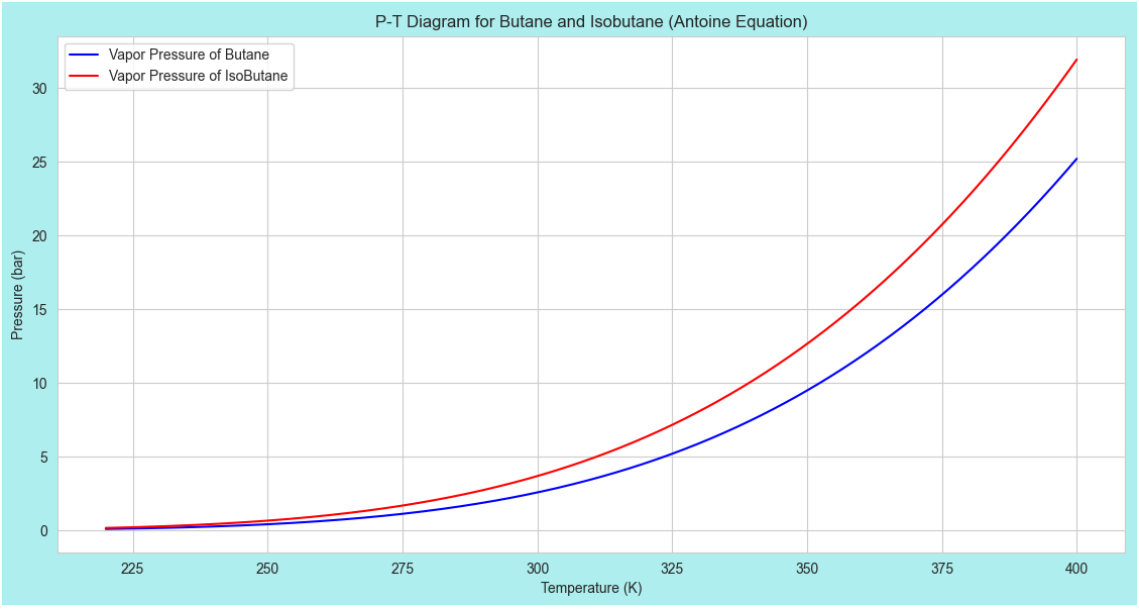


Figura 3.1

Ahora que hemos comprobado que la ecuación funciona vamos a graficar la mezcla binaria en función de la presión.

```python
def antoine(T,x):
    Pbutane=(1-x)*(10**(Ab - (Bb / (T + Cb))))
    Pisobutane=(x)*(10**(Ai - (Bi / (T + Ci))))
    return Pisobutane+Pbutane

def antoine_dew(T,x):
    Pbutane=(1-x)*(10**(Ab - (Bb / (T + Cb))))
    Pisobutane=(x)*(10**(Ai - (Bi / (T + Ci))))
    y=Pisobutane/(Pisobutane+Pbutane)
    return y
```

Definimos estas dos funciones de dos variables. Ahora mismo la T es fija, pero más adelante haremos algo al respecto de eso.

La x es la fracción del isobutano en el líquido, el componente más volátil (como hemos observado antes que tiene mayor presión de vapor).

Tenemos una función antoine que nos da la presión total según la T y la x.

Y una función antoine_dew (rocío) que nos da la y, la fracción de isobutano en el vapor.

```python
# Temperature range (in Celsius)
#T_range = np.linspace(200, 500, 500) # (start,end, number of points)
#T=350
T=350
x = np.linspace(0, 1, 500)
# Temperature range (in Kelvin)
#T_range = np.linspace(250,400,500)
# Calculate vapor pressures corresponding to the temperature range
P_range = antoine(T,x)
# at pure vapour if vapor pressure drops and reaches system pressure that is the dew point
```

Definimos la temperatura fija, el rango de la x y el rango de la P como función de la T y la x.

```
y=antoine_dew(T,x)
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
plt.plot(x,P_range, label="bubble point, x", color="blue") # bubble point
plt.plot(y,P_range, label="dew point, y", color="green") # dew point
plt.xlabel("x,y")
plt.ylabel("Pressure, bar")
plt.title("PXY butane and isobutane,"+ str(T) +" K, isobutane most volatile")
plt.legend()
plt.grid(True)
plt.gcf().set_facecolor('plum')  # Change background color of the figure
plt.gca().set_facecolor('white')  # Change background color of the inside figure
plt.show()
```

Definimos la y como la y obtenida la función del rocío.

Hacemos el plot. Recordemos que P_range,x e y son una extensa lista de valores. Al plotear unos con otros obtendremos una serie de puntos que harán las líneas.

Representamos (plt.plot)

La x y la P . En azul, punto de burbuja.

La y con la P. En verde, punto de rocío.

Para cada x hay una y (y>x excepto al principio y al final) y ambas se representan con la P en el eje vertical. Y es función de x.

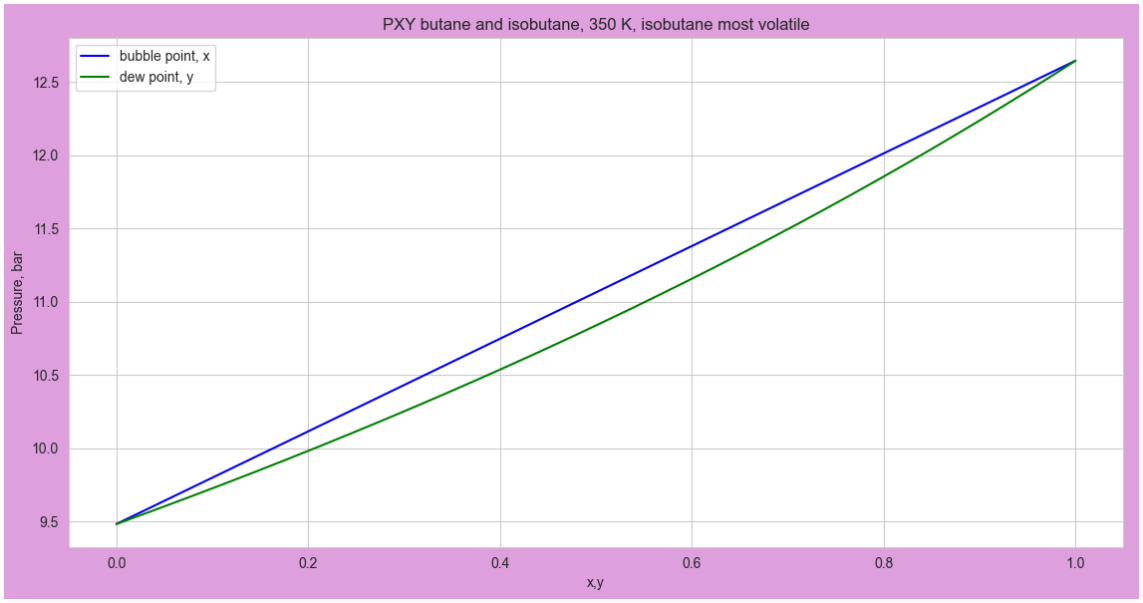


Figura 3.2

Podemos variar la temperatura en el script alterando esa variable.

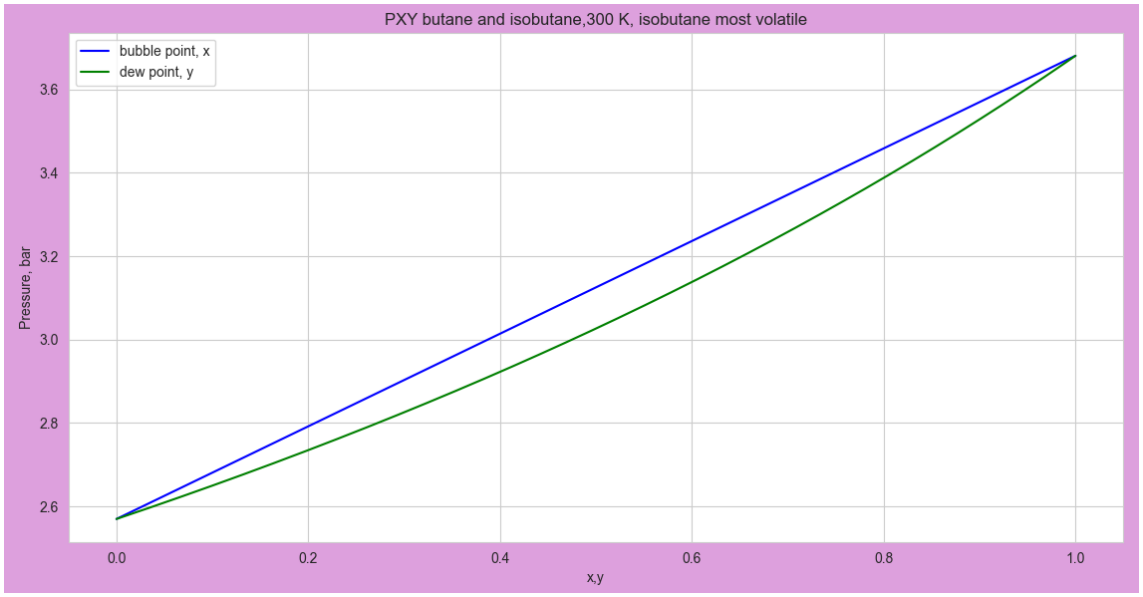


Figura 3.3

¿Y si pudiéramos hacer un plot en el que pudiéramos tocar la T en tiempo real?

Descubrimos la librería de widgets Bokeh, que nos permite esto. Esto llevó un tiempo buscando código y documentación al respecto.

Se trata de una librería que lo traduce a Javascript y abre un servidor local web donde aparece la gráfica con el widget. Esto implica ejecutar el código y posteriormente abrir la caja de comandos y abrir el servidor para verlo. Conseguimos automatizar todo esto dentro el script de Python (que puede abrir y ejecutar en la caja de comandos) y ese es el script que vamos a enseñar.

Esta es lo que aparece en la consola de Python al ejecutarlo:

```
C:\Users\usuario\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/usuario
2024-04-18 14:11:06,599 Starting Bokeh server version 3.1.1 (running on Tornado 6.4)
2024-04-18 14:11:07,569 User authentication hooks NOT provided (default user enabled)
2024-04-18 14:11:07,588 Bokeh app running at: http://localhost:5006/antoine3
2024-04-18 14:11:07,588 Starting Bokeh server with process id: 9432
2024-04-18 14:11:12,775 Starting Bokeh server version 3.1.1 (running on Tornado 6.4)
2024-04-18 14:11:14,124 Cannot start Bokeh server, port 5006 is already in use
2024-04-18 14:11:16,030 WebSocket connection opened
2024-04-18 14:11:16,031 ServerConnection created
```

Figura 3.4

Este es el resultado.

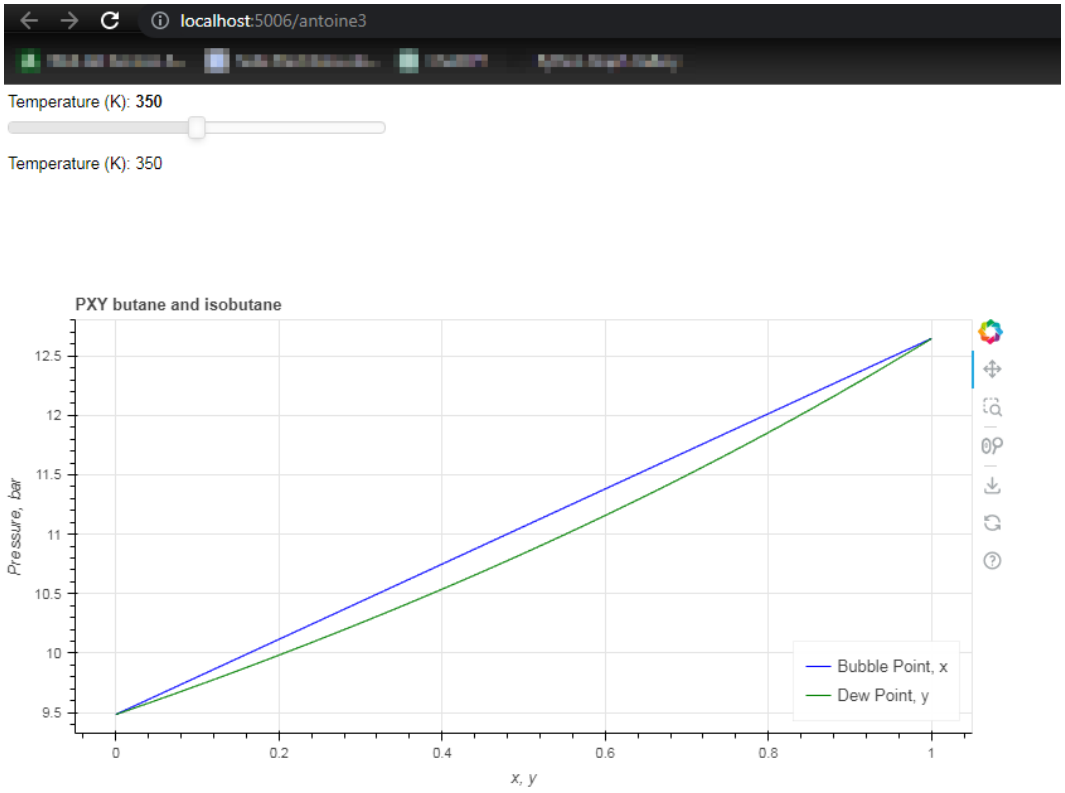


Figura 3.5

Moviendo el widget de la temperatura se ajusta la gráfica:

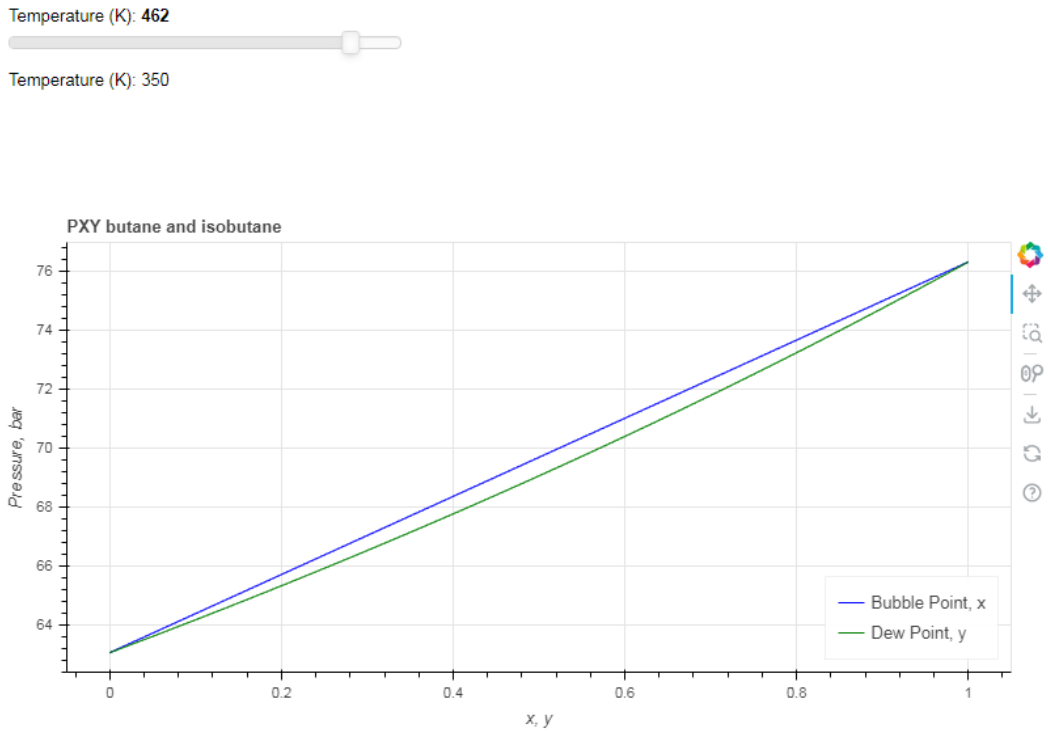


Figura 3.6

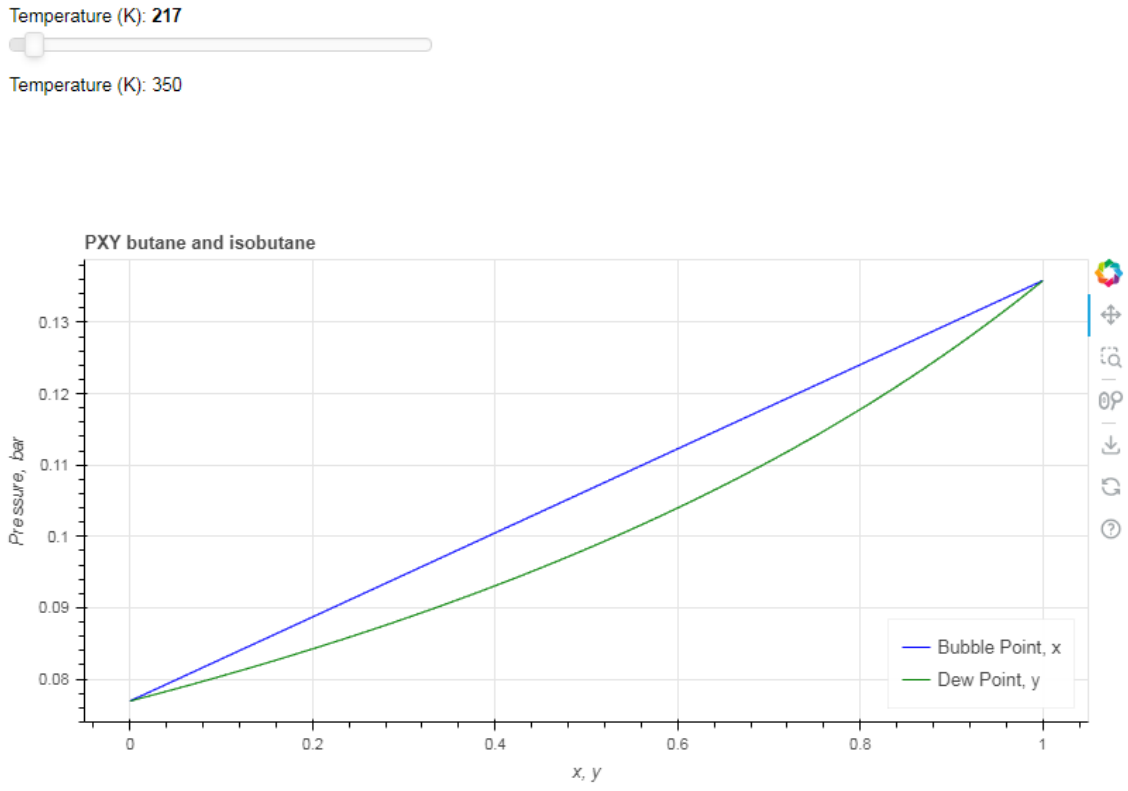


Figura 3.7

Este código aparece en los anexos.

4.Visualización de la regla de la palanca:

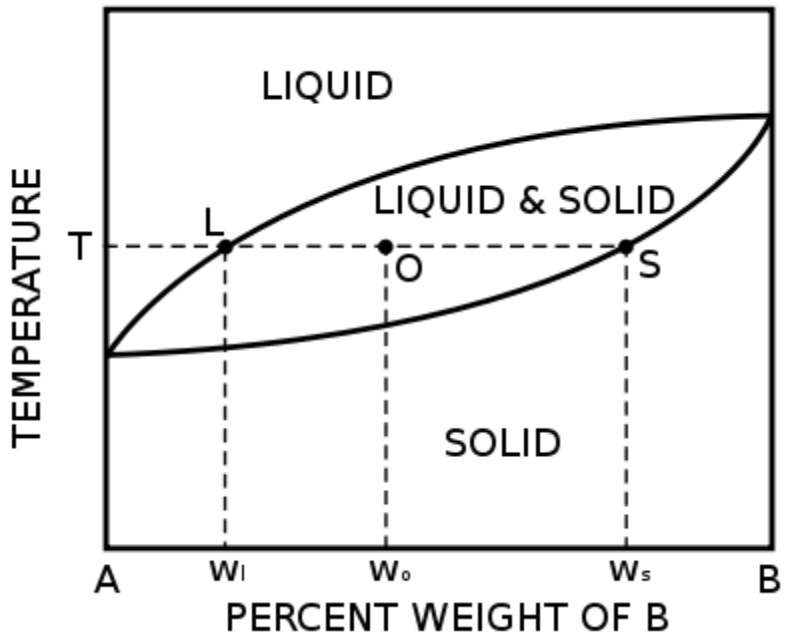


Figura 4.1

La regla de la palanca en termodinámica es una manera intuitiva de entender cómo las proporciones de dos sustancias en equilibrio afectan las condiciones termodinámicas. Se basa en la idea de que cuando una sustancia se evapora o se condensa, la otra sustancia en el sistema experimenta un cambio compensatorio en su proporción. Por ejemplo, en una mezcla líquido-vapor, si la proporción de vapor aumenta (por evaporación), la proporción de líquido disminuye para mantener el equilibrio. Este fenómeno se relaciona con la ley de Raoult y la ley de Dalton para mezclas de líquidos y gases respectivamente, donde la presión parcial de cada componente en el equilibrio depende de su fracción molar y presión de vapor. En esencia, la regla de la palanca ilustra cómo los cambios en una fase afectan las proporciones de las fases restantes para mantener el equilibrio termodinámico.

Veamos el resultado final:

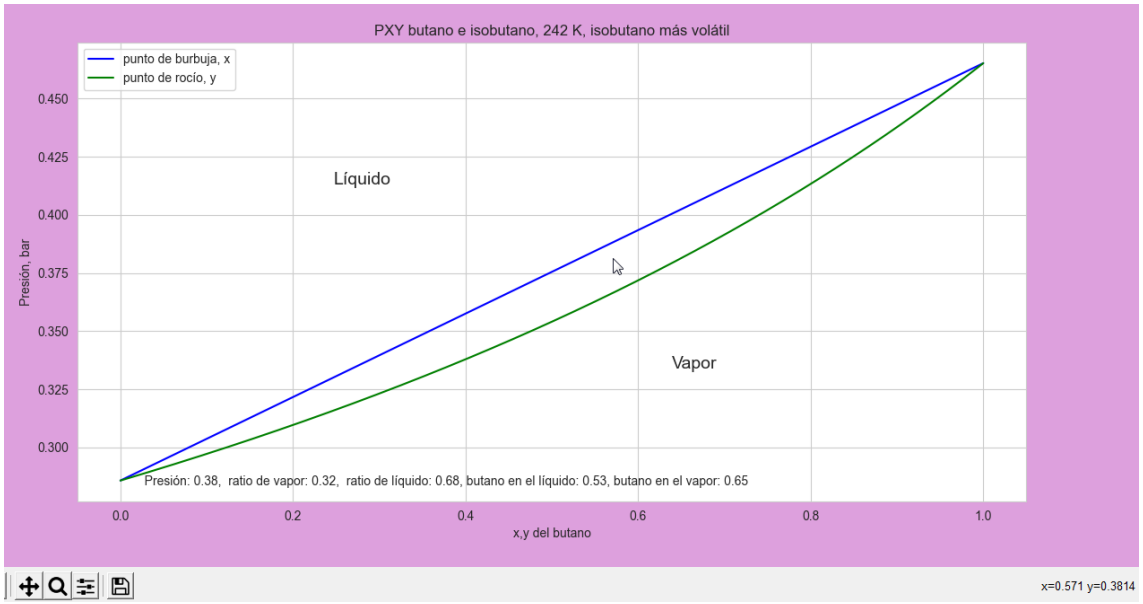


Figura 4.1

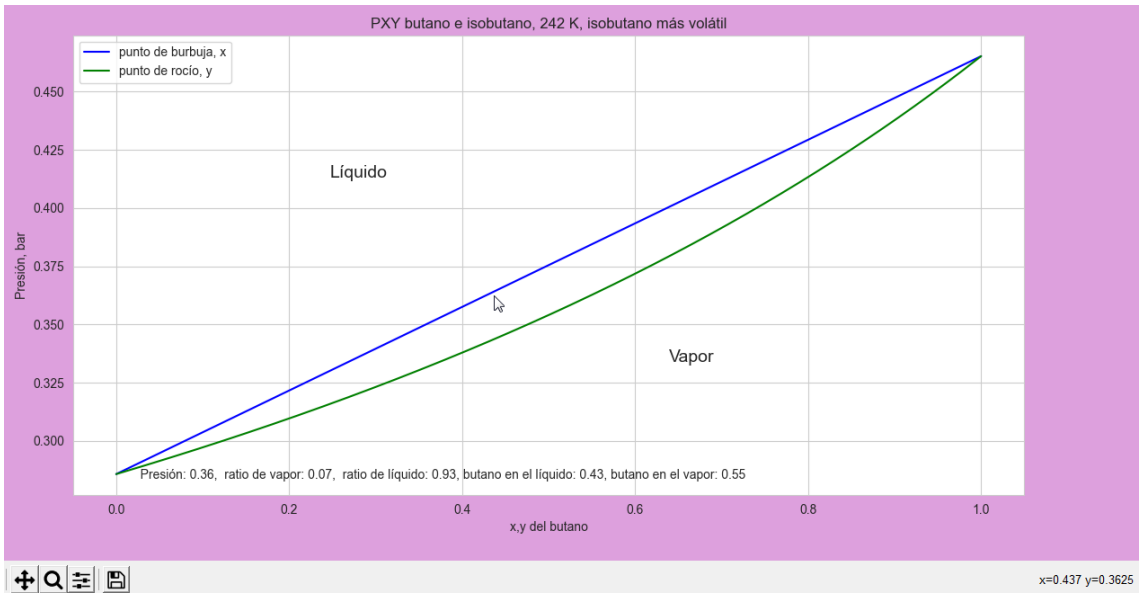


Figura 4.2

Este código también es derivado de los anteriores, pero tiene algunas cuestiones matemáticas interesantes.

Definimos unas funciones inversas a las de antoine y antoine_dew. Usamos el software Mathematica para ello.

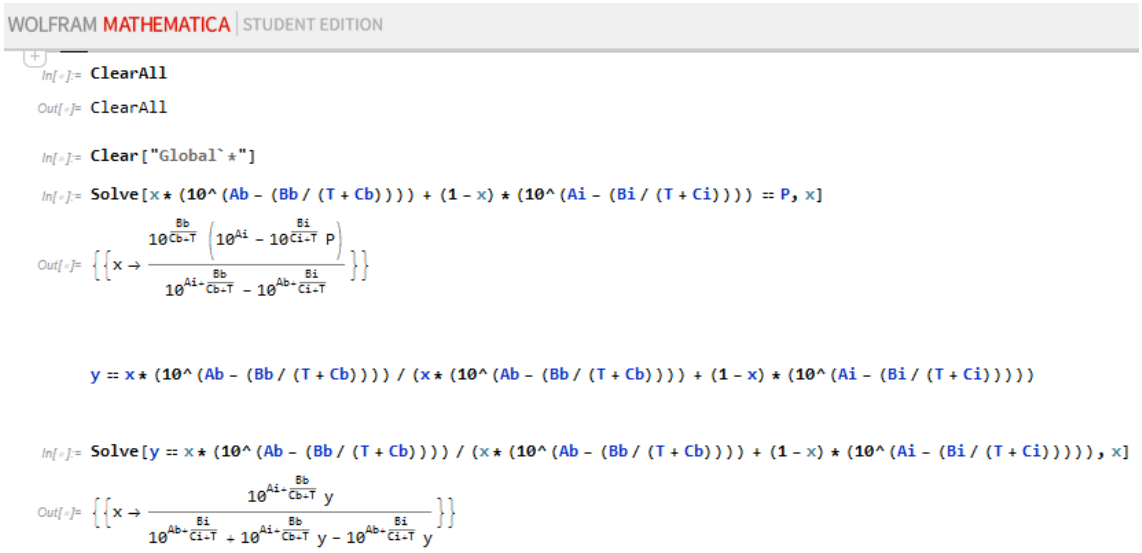


Figura 4.3

```
def inverse_antoine(T,P):
    try:
        x=(10**(Bi/(Ci+T))*((10**Ab)-10**(Bb/(Cb+T))*P))/((10**(Ab+(Bi/(Ci+T))))-10**(Ai+(Bb/(Cb+T))))
    except:
        x=0
    return x

def inverse_antoine_dew(T,y):
    x=((10^(Ab+Bi/(Ci+T)))*y)/((10^(Ai+Bb/(Cb+T)))+(10^(Ab+Bi/(Ci+T)))*y-(10^(Ai+Bb/(Cb+T)))*y)
    return x
```

Estas funciones serán necesarias posteriormente para la palanca.

```
# Temperature range (in Celsius)
#T_range = np.linspace(200, 500, 500) # (start,end, number of points)
#T=350 Antoine intended between 272 and 408
T=242
# se usa esta temperatura tan baja porque se observa mejor el equilibrio y la palanca, el espacio es más grande

x = np.linspace(0, 1, 500)
P = np.linspace(9.5, 12.7, 500)

# Temperature range (in Kelvin)
T_range = np.linspace(250,400,500)

# Calculate vapor pressures corresponding to the temperature range
P_range = antoine(T,x)

# at pure vapour if vapor pressure drops and reaches system pressure that is the dew point
```

Además de la x y la presión función de (T,x) ahora también definimos un rango de presión independiente y un rango de temperatura independiente.

```
# Plotting
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
#plt.plot(inverse_antoine(T,P),P, label="bubble point, x", color="red")#inversa
plt.plot(x,P_range, label="punto de burbuja, x", color="blue")#estándar
#plt.plot(antoine_dew(T,inverse_antoine(T,P)),P,label="dew point, y", color="violet")#inversa
plt.plot(antoine_dew(T,x),P_range, label="punto de rocío, y", color="green")#estándar
plt.xlabel("x,y del butano")
plt.ylabel("Presión, bar")
plt.title("PXY butano e isobutano, "+str(T)+" K, isobutano más volátil")
plt.legend()
plt.grid(True)
```

Se comprueba que las funciones inversas dan el mismo plot que las que llamamos estándar.

El ploteo de [x, P(x,T)] es igual al ploteo de [x(T,P),P]. En un caso la x es independiente y en el otro es la P la variable independiente.

Igual con la línea de rocío; el ploteo de [y(T,x), P(x,T)] es igual al ploteo de [y(T, x(T,P)),P]. En un caso la x y la T son independientes y en el otro son la T y la P las variables independientes.

```
# Text annotation for cursor coordinates and products
text = plt.text(0.03, 0.03, '', transform=plt.gca().transAxes, ha='left', va='bottom')

# Additional text annotations
plt.text(0.3, 0.7, 'Líquido', transform=plt.gca().transAxes, ha='center', va='center',fontsize=14)
plt.text(0.65, 0.3, 'Vapor', transform=plt.gca().transAxes, ha='center', va='center',fontsize=14)
```

Posicionamos los textos de leyenda.

```
# Define x_cursor and y_cursor as global variables
x_cursor = None
y_cursor = None
# Function to update cursor coordinates and their products
def update_cursor(event):
    global x_cursor, y_cursor
    if event.inaxes:
        x_cursor = event.xdata
        y_cursor = event.ydata
        text.set_text(f'          Presión: {abs(y_cursor):.2f},  ratio de vapor: {(x_cursor-inverse_antoine(T,y_cursor))/abs(antoine_dew(T,inverse_antoine(T,y
        #text.set_text(f'ratio de vapor: {abs(x_cursor-inverse_antoine(T,y_cursor)):.2f}, ratio de líquido: {abs(x_cursor-antoine_dew(T,inverse_antoine(T,y_cur
        plt.draw()
# Connect event handler to mouse motion events
plt.gcf().canvas.mpl_connect('motion_notify_event', update_cursor)
```

Se definen unas variable vacías para ver la posición del cursor y una función para ver la posición en tiempo real del cursos y hacer los cálculos.

Recordemos que es una gráfica de x,y en el eje horizontal y P en el vertical.

La presión es la coordenada Y del cursor.

Presión: {abs(y_cursor):.2f},

El ratio de vapor es la coordenada X del cursor menos la x(T,y_cursor) partido

```
ratio de vapor: {(x_cursor-inverse_antoine(T,y_cursor))/abs(antoine_dew(T,inverse_antoine(T,y_cursor))-inverse_antoine(T,y_cursor)):.2f}
```

La x del cursor menos la X en función de la T y la y_cursor (esto es el punto de burbuja en función de la posición del ratón). La x_cursor menos la X de la gráfica en función de la y_cursor, divido entre la diferencia entre la línea de rocío en función de la y_cursor y la línea de burbuja en función de la y_cursor

```
ratio de líquido: {(-x_cursor+antoine_dew(T,inverse_antoine(T,y_cursor)))/abs(antoine_dew(T,inverse_antoine(T,y_cursor))-inverse_antoine(T,y_cursor)):.2f}
```

El ratio de líquido es la diferencia entre la x_cursor y la línea de rocío dividido entre las líneas de burbuja y rocío. Viene definido en negativo (-a+b) porque en origen la lenteja estaba de al revés. Posteriormente lo cambiamos todo.

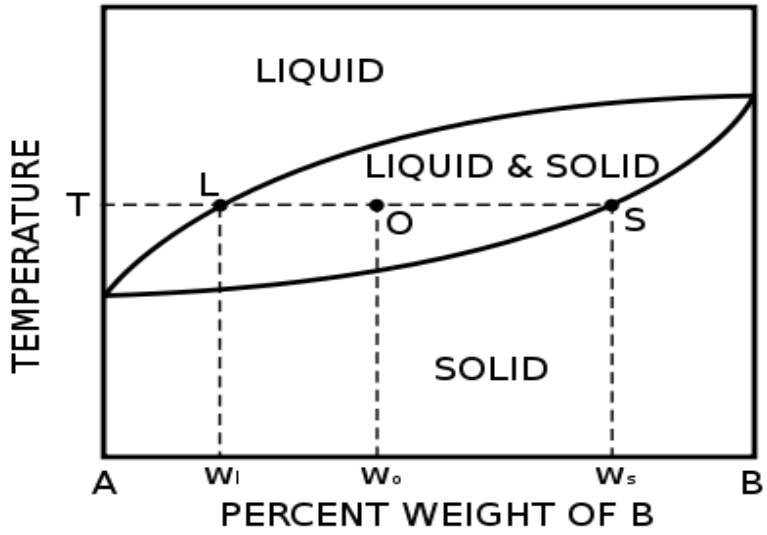
```
butano en el líquido: {abs(inverse_antoine(T,y_cursor)):.2f}
```

Butano en el líquido es la x de la línea de burbuja en función de la y_cursor.

```
butano en el vapor: {abs(antoine_dew(T,inverse_antoine(T,y_cursor))):.2f}
```

Butano en el vapor es la y de la línea de rocío en función de la y_cursor.

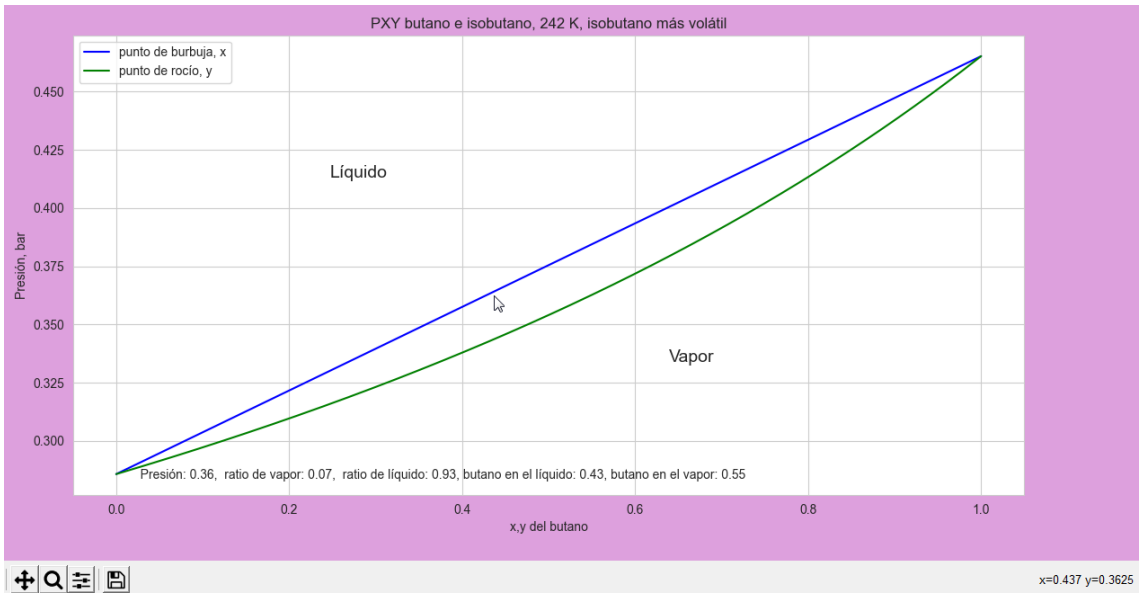
Resulta complejo de explicar. Básicamente, las líneas en origen están definidas en función de la x. Para comparar la x_cursor con estas líneas definimos las líneas en función de la y_cursor para obtener la x correspondiente a la línea. Comparando la x_cursor y la x de la línea es como hacemos calculamos las distancias de las líneas horizontales de la regla de la palanca.



Esto llevó un rato de abstracción y pensamiento matemático, calcular la X en función de la Y (esto fue lo que nos hizo definir las ecuaciones inversas, cosa que requirió del software Mathematica). Inicialmente buscamos documentación para calcular distancias entre líneas y la posición del ratón. La posición del ratón sí fue algo que encontramos documentación al respecto pero las líneas ploteadas no quedan guardadas como dato y estaba planteamiento no podía resolverse así.

```
plt.gcf().set_facecolor('plum')  # Change background color of the figure
plt.gca().set_facecolor('white')  # Change background color of the inside figure
plt.show()
```

Después se definen los colores de la gráfica y se muestra.



Bibliografía:

Al Sweigart. *Automate the Boring Stuff with Python.* ISBN-10-1593275994. ISBN-13-978-1593275990

Peter Atkins. *Atkins' Physical Chemistry 11e* .ISBN-10. 0198769865 · ISBN-13-978-

0198769866

Smith.h, William F. halkias; Hashemi, Javad (2006), *Foundations of Materials Science and Engineering* (4th ed.), McGraw-Hill, pp. 318–320, ISBN 0-07-295358-6.

Anexos:

Plot de datos.

Con el webscraping anterior obtuvimos los datos de la Proton Affinity y de la Gas Basicity (ajustando un poco el script).

View reactions leading to H_2^+ (ion structure unspecified)

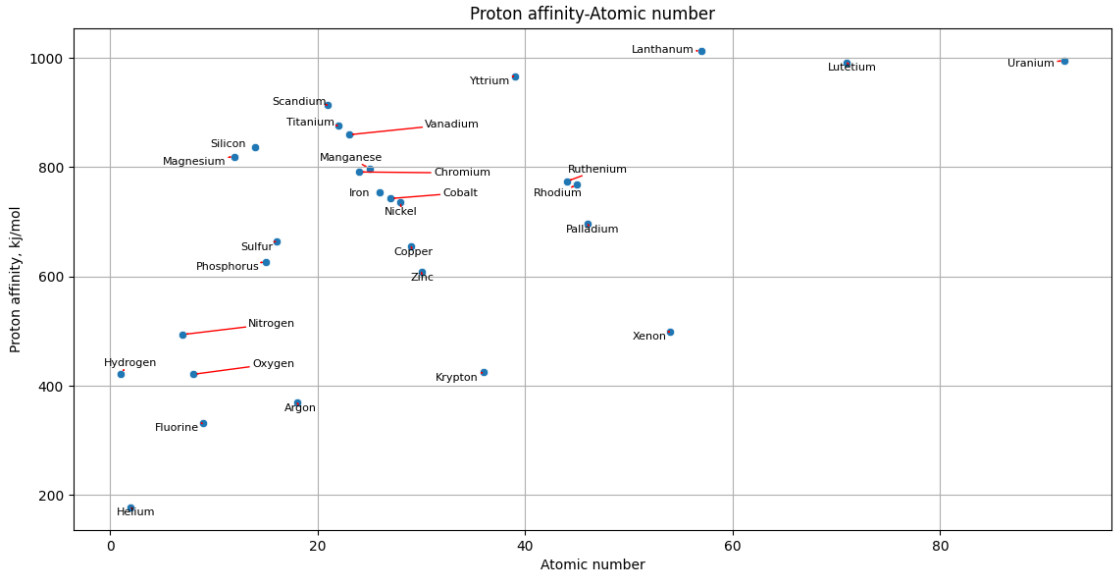
Quantity	Value	Units	Method	Reference	Comment
IE (evaluated)	15.42593 ± 0.00005	eV	N/A	N/A	*L*
Quantity	**Value**	**Units**	**Method**	**Reference**	**Comment**
Proton affinity (review)	422.3	kJ/mol	N/A	Hunter and Lias, 1998	*HL*
Quantity	**Value**	**Units**	**Method**	**Reference**	**Comment**
Gas basicity	394.7	kJ/mol	N/A	Hunter and Lias, 1998	*HL*

Con estos datos vamos a hacer varios plots para observarlos en relación mutua y con la masa atómica y tabla periódica.

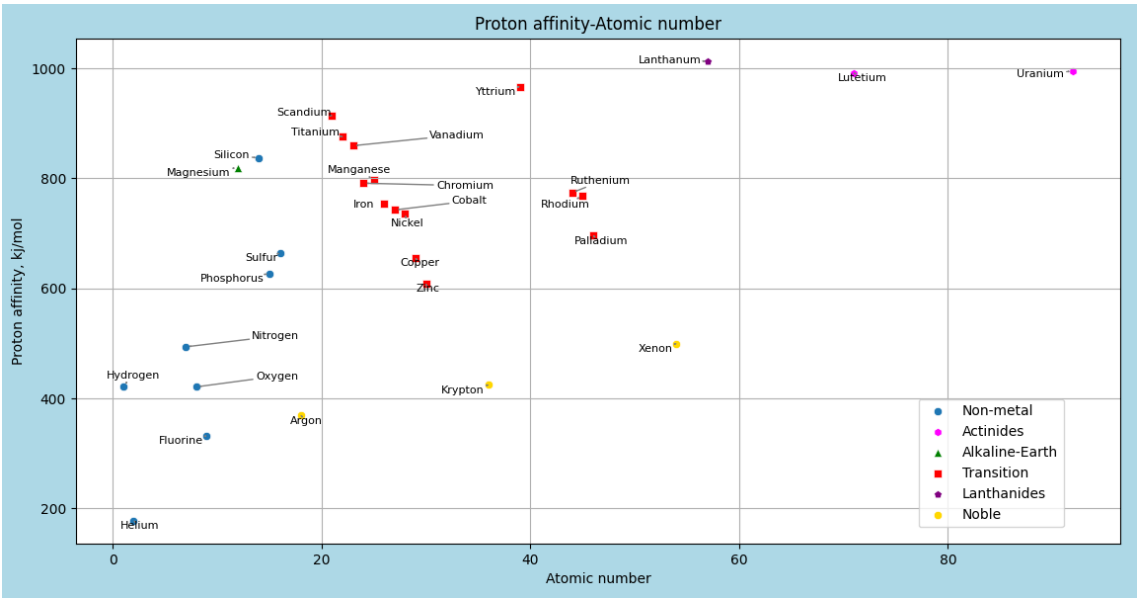
Un plot es una representación gráfica.

Estos son los plots que hicimos:

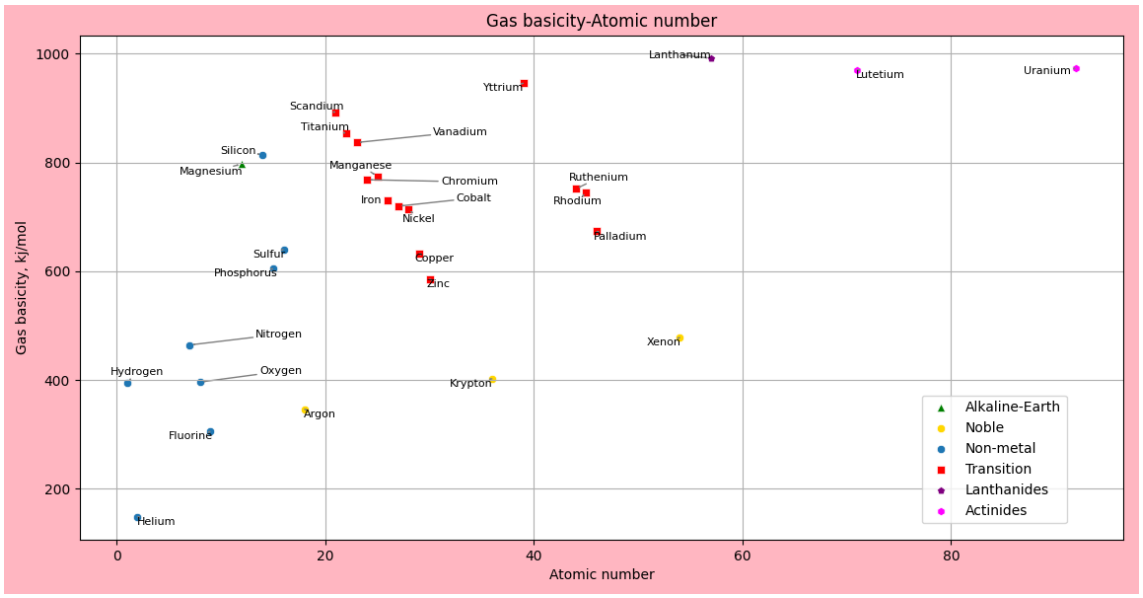
Plot 1: Afinidad protónica en función del número atómico. Este fue el primer script y está más en bruto.



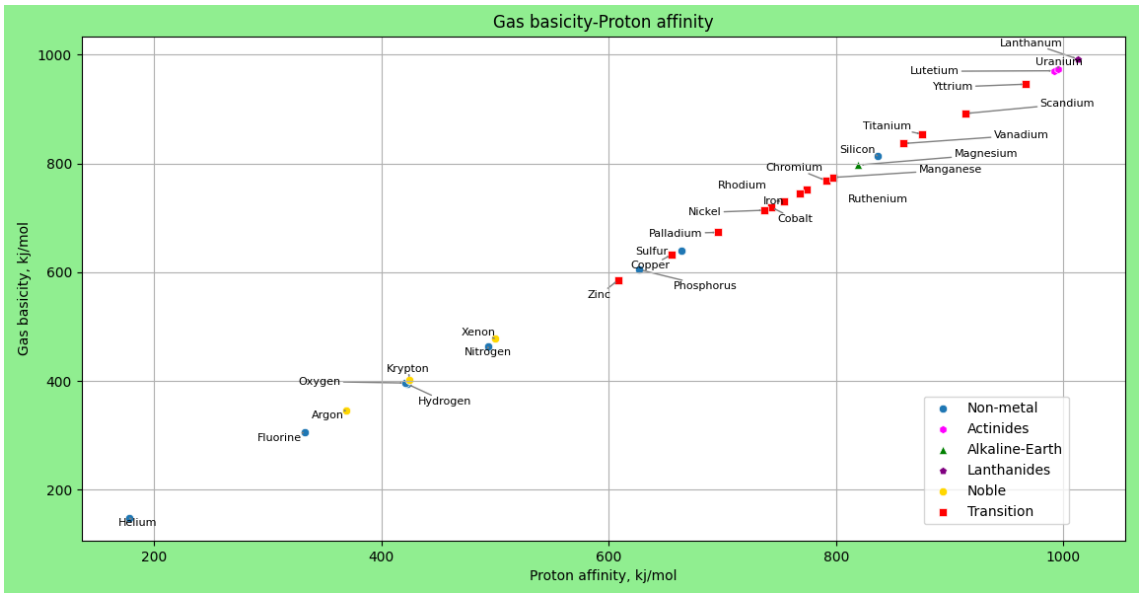
Plot 2: Afinidad protónica en función del número atómico. Este script está más cuidado, presenta una leyenda de los tipos de elementos en la esquina y tiene el borde coloreado. Los otros scripts son el mismo con otros ejes de datos. Hicimos pruebas con distintos esquemas de color y nos decantamos por estos.



Plot 2-gas-basicity: Gas basicity en función del número atómico. No traducimos el término Gas basicity porque no es una propiedad que controlemos ni conozcamos bien (igual con la afinidad protónica, pero en este caso la traducción es más inequívoca)



Plot 4: La Gas basicity en función de la Proton affinity.



Veamos el script.

```
# library & dataset
import seaborn as sns
import matplotlib.pyplot as plt
from adjustText import adjust_text
```

Al principio del script siempre están las importaciones de las librerías.

Matplotlib.pyplot es la librería para hacer plots.

Seaborn es una librería relativa a los plots, usada aquí para añadir detalles al plot.

adjustText es una librería específica que encontramos buscando una solución a como ajustar el texto para que no quedase encima de los puntos, aparece en una línea específica.

En este script nos basta con introducir los datos así (esto es un texto bruto, una string), por la cantidad de datos no hay necesidad de almacenarlos en un json ni leerlos de una hoja de cálculo.

Sí hemos preseleccionado e introducido sólo los elementos que tenían su proton affinity y gas basicity tabulado con sus sendos datos. Eliminamos en las columnas de la hoja de cálculo las filas con “------“ en lugar de un valor y las copiamos enteras. Los espacios vacíos no son un problema.

```
raw_types=('''
Non-metal
Non-metal
Non-metal
Non-metal
Non-metal
Alkaline-Earth
Non-metal
Non-metal
Non-metal
Noble
Transition
Transition
```

Hemos añadido una columna en la que clasificamos cada elemento en función de su tipo según su columna en la tabla periódica. Para nosotros químicos esto es algo harto conocido. Resulta difícil quitarse la inercia de trabajar en inglés cuando los datos de base son en inglés.

```
proton_affinity=list(raw_proton_affinity.split())
proton_affinity_data=[float(x) for x in proton_affinity]

gas_basicity=list(raw_gas_basicity.split())
gas_basicity_data=[float(x) for x in gas_basicity]

elements=list(raw_elements.split())
types=list(raw_types.split())

atomic_number=list(raw_atomic_number.split())
atomic_number_data=[float(x) for x in atomic_number]
```

Las strings bruto las hacemos listas haciendo un list del Split de cada string. Esto es; separamos el texto en palabras individuales y hacemos una lista con ellas.

Definimos la lista de datos como la lista que hace el bucle de hacer un float (número decimal computable, hasta ahora era tipo texto) de cada elemento de la lista previa.

```
# Sample data
#x = [1, 2, 3, 4, 5]
#y = [2, 3, 5, 7, 11]

#print(len(atomic_number))

#print(len(proton_affinity))
#print(len(elements))
#print(len(types))

# Create scatterplot
#sns.scatterplot(x=atomic_number_data, y=proton_affinity_data, marker='o')
```

Este código inerte es remanente de las pruebas necesarias para llegar al final. Se probó el plot con unos datos de prueba y se comprobó la longitud de cada lista, que sean todas iguales, para asegurarnos de que cada elemento tiene sus sendos parámetros y tipo, que nos hemos equivocado. Si no fueran iguales habríamos cometido algún error. Por ejemplo, el hidrógeno y sus parámetros y tipo están todos en la posición [0] de sus respectivas listas.

Por eso a la hora de llamar el elemento[i] se le relaciona su afinidad[i] y su tipo[i].

```
#x=atomic_number_data

x=proton_affinity_data
y=gas_basicity_data
```

Definimos como x e y los parámetros que vamos a representar. En este caso la afinidad y la basicidad.