UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Ejercicios introductorios a la programación en Python para el análisis de datos geológicos georreferenciados

MATERIAL DIDÁCTICO

Que para obtener el título de

Ingeniera Geóloga

P R E S E N T A

Brenda Ortiz Soto

ASESOR DE MATERIAL DIDÁCTICO

Dr. Darío Emmanuel Solano Rojas

Ciudad Universitaria, Cd. Mx., 2022

ÍNDICE

RESUMEN

El presente trabajo tiene como finalidad presentar a la comunidad de Ingeniería en Ciencias de la Tierra una herramienta de gran utilidad en el ámbito profesional como en el ámbito académico. Sí, nos referimos a Python, el lenguaje de programación que ha alcanzado gran popularidad mundial en los últimos años.

¿Por qué proponemos Python? Python es un lenguaje que es fácil de aprender (comparado con otros lenguajes), es gratis y lo puedes utilizar en cualquier computadora en todo el mundo. Python es utilizado en el desarrollo de aplicaciones web, Inteligencia Artificial (AI), Aprendizaje Automático, sistemas operativos, aplicaciones móviles y videojuegos.

Este trabajo incluye una introducción a Python, ventajas y desventajas de usarlo, una guía de cómo instalar Python en tu computadora y un compendio de ejercicios básicos de análisis y representación de datos geológicos en Jupyter Notebooks.

Los ejercicios propuestos están pensados para que el usuario navegue una serie de pasos que le faciliten el análisis e interpretación de información de manera computacional en Python. Dichos pasos básicos son: 1. Se presentan los datos, 2. Se realiza "limpieza" de los datos, 3. Se seleccionan los datos de interés para su análisis 4. Se representan los datos en gráficos que facilitan su interpretación.

Realizar el análisis e interpretación de información en Python representa una gran ventaja, ya que podemos reutilizar y reciclar el código que generemos. Además, el usuario se vuelve capaz de utilizar código escrito por alguien más, incluida la comunidad internacional de geociencias, lo que amplía el panorama de los problemas que puede resolver. Dicho todo esto podemos darnos cuenta que el uso de una herramienta como Python nos abrirá muchas puertas.

ABSTRACT

The main goal of this work is to introduce to the community of Earth Science Engineering at UNAM a useful tool used not only in industry but in academia as well. Yes, I mean Python, the programming language that has become internationally popular in recent years.

Why Python? Python is a programming language that is easy to learn (as compared to other programming languages); also, it is free of charge, and it can be used on any computer around the world. Python is used, for instance, in developing web apps, artificial intelligence, Machine Learning, Operative Systems, mobile apps, and video games.

This work includes an introduction to Python, pros, and cons, a guide of Python installation, as well as a compendium of introductory exercises in Jupyter notebooks of geologic data analysis and presentation.

The proposed exercises are designed to follow a series of basic steps guiding the user towards computational analysis and interpretation of geological data. Such steps are: 1. Data are presented and loaded, 2. Data cleaning, 3. Selection of the data of interest, 4. Graphical representation of data to facilitate interpretation. Performing such data analysis and interpretation in Python represents a great advantage in terms of code re-use and recycling. Furthermore, the user becomes capable of using code written by somebody else, including the international geoscience community, which broadens the solvable-problem horizon.

Having said all that, we can realize that using Python can potentially open many doors for us.

AGRADECIMIENTOS

Al Dr. Darío E. Solano Rojas por su apoyo y motivación para elaborar este material didáctico.

Al proyecto PAPIME: "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra", Proyecto PAPIME PE101020 por el patrocinio económico.

A mi Universidad, a la Facultad, por brindarme todas las herramientas necesarias para llegar hasta donde estoy hoy.

A mis sinodales por revisar, corregir y asesorarme en todo este trabajo.

A mi familia, que siempre estuvo conmigo y me apoyo durante toda mi etapa escolar. Gracias a mis papás por apoyarme incondicionalmente a cumplir mi sueño de estudiar Ingeniería Geológica en otra ciudad. Gracias mamá y papá por todo el esfuerzo que hicieron para mantenerme en otra ciudad, por dar todo para que yo pudiera llegar hasta donde estoy. A mi hermana Mareny por todos estos años de convivencia, por crecer juntas, por escucharme en todo momento, por ser mi cómplice, por compartir tus conocimientos de geofísica conmigo, pero sobre todo por estar conmigo en todos los buenos y malos momentos.

A mis amigos, por compartir su tiempo conmigo, por todas las risas, tardes de tareas, por las prácticas de campo, recorridos en campo, porque cada uno me enseño algo durante este camino, por volverse personas incondicionales.

A cada uno de mis profesores que tuve en el Anexo de Ingeniería que me enseñaron las bases matemáticas e ingenieriles, que me hicieron desvelarme, pero de quiénes aprendí. A mis profesores de las materias del Principal, por su paciencia, por compartir sus conocimientos y pasión por la ingeniería geológica conmigo. A mis profesores de las materias de Humanidades, por formarme en el área cultural.

1. INTRODUCCIÓN

El sector industrial requiere de ingenieros con habilidades programáticas para la resolución de problemas y optimización de tiempos. En la Facultad de Ingeniería de la UNAM, es muy común que los egresados de la carrera de Ingeniería Geofísica dominen estas habilidades, ya que durante el transcurso de su formación profesional cursan varias asignaturas en las cuales programan. En cambio, son pocos los ingenieros geólogos e ingenieros en minas y metalurgia que egresan de la facultad con conocimientos sólidos en programación. Recientemente, el lenguaje de programación Python ha alcanzado gran popularidad mundial ya que es fácil de aprender. Este trabajo pretende ser una introducción al lenguaje de programación en Python para los estudiantes en las carreras de Ingeniería en Ciencias de la Tierra, con ejemplos de las áreas de Geología y Minería, pero va enfocado a estudiantes que aún no aprenden a programar o que apenas van iniciando. Con este material didáctico podrás analizar datos geológicos e interpretarlos para formular tus propias conclusiones.

1.1 Tendencias de la programación en las geociencias

Las computadoras son cada vez más poderosas y capaces de procesar más datos. Por lo tanto, son capaces de resolver problemas cada vez más complejos, sobre todo cuando se trata de la implementación repetitiva de algoritmos y del manejo de grandes conjuntos de datos. Las Ciencias de la Tierra no están exentas del avance de la tecnología y el incremento del volumen de datos a utilizar para la exploración, los cuales tienden a ser cuantiosos. Por supuesto, una exploración geológica basada en más datos tendrá menos incertidumbre. Además, los datos geológicos tienen características geométricas y dimensionales particulares, lo cual hace complicado su análisis "a mano". En la actualidad, tanto la industria como la academia requieren de profesionales que tengan nociones básicas de programación para manejar grandes cantidades de datos en rutinas estandarizadas.

Existen muchos lenguajes de programación, todos con ventajas y desventajas. Por ejemplo, MATLAB (The MathWorks, 2015) es un lenguaje de programación muy utilizado en las ingenierías, capaz de realizar desde operaciones aritméticas simples hasta modelado y clasificaciones utilizando inteligencia artificial. Sin embargo, es una herramienta que

requiere pagar una licencia, la cual no siempre está asegurada en el campo laboral. En este caso, cualquier código que se pueda tener en MATLAB, va a enfrentar la posibilidad de volverse obsoleto por la imposibilidad de ejecutarlo o reutilizarlo. Si bien es cierto que existen otros paquetes con cierta compatibilidad con MATLAB, no todas las herramientas son soportadas. Por lo tanto, es recomendable que el lenguaje de programación a utilizar para realizar un proyecto sea de licencia abierta, lo cual permite tener acceso a todas sus funcionalidades (*spoiler alert*: vamos a usar Python).

Python es el lenguaje que marca el estándar para la industria en nuestros días. A pesar de que muchos otros lenguajes de programación tienen capacidades similares a Python, todo parece indicar que Python llegó para quedarse (aunque probablemente eso se dijo, en su tiempo, de otros lenguajes de programación). Particularmente, la comunidad geológica internacional se ha esforzado por adoptar a Python como el lenguaje estándar. El objetivo de este material, por supuesto, no es formar programadores que ganen competencias de programación a *hackatones*, si no dar un panorama general del potencial de aplicar la programación a la Ciencias de la Tierra.

1.2 Marco teórico de referencia

Aprender un lenguaje de programación se siente igual que aprender un idioma nuevo: al inicio, puede llegar a sentirse -mucha- frustración. Por ejemplo, la primera vez que pides una hamburguesa en un restaurante de comida rápida en Estados Unidos, esperas que te reciban con un "*good afternoon, sir*". En la práctica, es muy probable que el restaurante esté lleno, y que el encargado de la caja diga de manera directa: "*for here or to go*?". Por supuesto, ninguna clase de inglés te puede preparar para una situación tan específica. Más aún, después de aprender a pedir hamburguesas, tal vez exista la necesidad de que pidas un sándwich, pollo frito, etc., lo cual implicará nuevos retos en saber qué ingredientes ordenar, cómo pedirlos, saber si el restaurante los da gratis, los cobra extra, o tal vez si los debes de tomar tú mismo de la barra de salsas. Al inicio, enfrentar esas situaciones da pena, toma tiempo, consume energía y esfuerzo mental. Eventualmente, la práctica hace que ordenar

cualquier tipo de comida se vuelva cotidiano y fácil, hasta que llegan nuevos retos para comunicarse en un ambiente diferente como el banco, la tintorería, el súper, etc., etc. Aprender un idioma es un proceso permanente, en donde nuevos retos se agregan con el tiempo, pero con la convicción de que esa incomodidad es temporal, eventualmente se llega a dominar el léxico especial para cada situación. Lograr la fluidez en un lenguaje de programación conlleva exactamente el mismo sentimiento, con una pequeñísima ventaja que tiene nuestra generación: Google.

La razón por la cual Python es el lenguaje de elección para la industria hoy en día, es porque Python es gratuito, portable, poderoso y muy fácil de usar (Lutz, 2007; J. M. Perkel, 2015). Además, es aplicable para resolver problemas de la vida diaria (J. M. Perkel, 2015), y tiene ejemplos para casos muy específicos, gracias a que Python tiene una comunidad enorme, la cual desarrolla herramientas para aplicaciones específicas, de manera que no hay que empezar las cosas desde cero (Python Software Foundation, 2020).

Una de las características más importantes de la comunidad de Python, es el impulso que se le ha dado a las mujeres programadoras. Algunas de las comunidades más importantes impulsoras del movimiento de mujeres programadoras a nivel internacional son la Hackbright Academy en San Francisco, Ladies Learning Code y PyLadies. Este último grupo tiene una comunidad en crecimiento en México (En Twitter las encuentras como @MxPyladies). En general, la comunidad de Python se ha caracterizado por procurar a grupos que tradicionalmente no se dedican a la programación.

No es coincidencia que grandes sitios webs que usamos todos los días, como Google, Youtube, Bitly, SurveyMonkey, hayan surgido a través de o adoptado Python en algún momento de su crecimiento. Incluso ArcGIS, uno de los softwares más conocidos de nuestra generación, depende fundamentalmente de Python para sus aplicaciones a gran escala (Python Software Foundation, 2020). Python tiene un sinnúmero de aplicaciones en las áreas de desarrollo web e internet, videojuegos, ciencia y cálculos numéricos, educación, negocios, desarrollo de software, entre muchos otros (Python Software Foundation, 2020).

Python viene de Monty Python, y no de la serpiente. El creador de Python, Guido van Rossum, nombró a este lenguaje de programación en 1991 en honor a un programa de comedia de la BBC, llamado *Monty Python's Flying Circus* (Lutz, 2007). La serpiente, sin embargo, es ahora la mascota que aparece en todos los libros de referencia de este lenguaje de programación. El nombre Python hace referencia tanto al lenguaje como al "intérprete". Es decir, un código está escrito en el lenguaje Python, y al momento de ejecutarlo, el intérprete de Python lo convierte en lenguaje máquina para seguir las instrucciones del código. Dicho intérprete se instala dependiendo de cuál es el "sabor" de Python que nos interesa. La instalación mínima de Python requiere un ejecutable y un conjunto de librerías ligados a él, en el siguiente capítulo se discute.

1.3 Lo fundamental a utilizar de Python

La comunidad científica, independientemente de su ramo, utiliza un grupo esencial de paquetes (J. M. Perkel, 2015), los cuales son: NumPy (arreglos matemáticos), SciPy (álgebra lineal, ecuaciones diferenciales, procesado de señales), SymPy (matemáticas simbólicas), Matplotlib (para graficado) y Pandas (análisis de datos). Cada paquete se especializa en cumplir una función específica sobre la que, a su vez, es posible construir otros paquetes (Fig. 1).

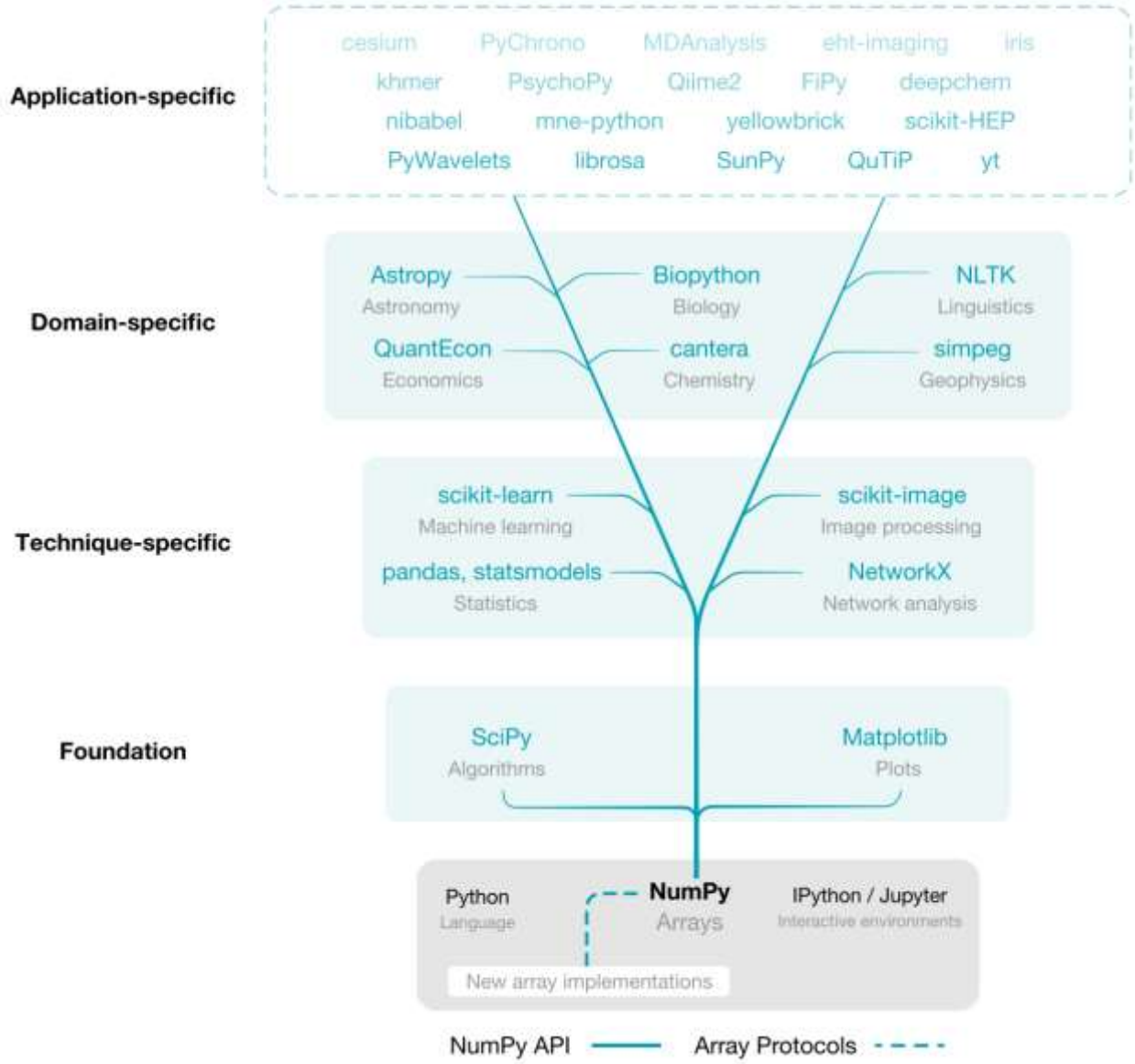


Fig. 1. Estructura del ecosistema de Python (Harris et al., 2020).

NumPy es la base sobre la que se erige el ecosistema científico para Python. NumPy es la librería de programación basada en arreglos de Python, la cual proporciona una sintaxis compacta y expresiva para acceder, manipular y operar datos en formato de vectores, matrices, y arreglos n-dimensionales (Harris et al., 2020). NumPy es tan importante para la ciencia, que tuvo un papel fundamental para el descubrimiento de las ondas gravitacionales y la primera imagen de un hoyo negro (Harris et al., 2020). NumPy no forma parte de las librerías estándar de Python. Sin embargo, el equipo que desarrolla NumPy trabaja muy de cerca con los desarrolladores de Python (Harris et al., 2020).

SciPy es una librería de rutinas numéricas para Python, la cual provee los elementos fundamentales para modelar o resolver problemas científicos (Virtanen et al., 2020). SciPy incluye algoritmos para optimización, integración, interpolación, problemas de eigenvalores, ecuaciones algebraicas, ecuaciones diferenciales, y muchos otros problemas. SciPy se construyó sobre NumPy, por lo que tiene acceso a los arreglos y estructuras de datos de NumPy, y a su vez, permite que otras librerías se construyan encima de él. SciPy también tuvo un papel importante en el descubrimiento de las ondas gravitacionales y en la primera imagen de un hoyo negro.

Pandas, por su parte, se enfoca en leer, manipular y preparar datos para un análisis práctico en Python. Matplotlib, es una librería para crear visualizaciones estéticas, animadas e interactivas en Python, la cual se utiliza para presentar conjuntos de datos y resultados de una manera gráfica. A pesar de que cada uno de los módulos mencionados se han convertido en el estándar de la programación en Python, cada uno es independiente de la distribución oficial de Python. Desde sus inicios, SciPy y Matplotlib se encuentran estrechamente ligados con NumPy. Juntos, y sumados a un ambiente interactivo como Jupyter y a un graficador como Matplotlib, proveen una base para la programación basada en arreglos de Python.

1.4 Notebooks interactivos para compartir código

La necesidad de compartir piezas de código de manera rápida siempre ha existido. Sin embargo, no es poco común recibir un programa y que al momento de ejecutarlo en una computadora diferente de donde fue creado, falle. Existen varias razones comunes por las que un código puede fallar al ejecutarse en una computadora diferente, entre ellas, que la versión del programa sea diferente, que las librerías necesarias no estén instaladas, o que el *path* no esté bien definido. También es común que después de resolver los problemas básicos de compatibilidad entre computadoras, se requiera modificar el código que nos compartieron, y es ahí en dónde radica la verdadera dificultad de usar código ajeno. Sobre todo, es difícil saber si el resultado final de una rutina es correcto si no se pueden ver los productos intermedios.

La necesidad de compartir código y permitir a alguien más ejecutarlo y modificarlo, mostrando paso por paso el avance de la rutina, y además dejando notas para facilitar las cosas al nuevo usuario del código, produjo el desarrollo de una herramienta interactiva llamada IPython (I=*interactive*) (She, 2014). Posteriormente, la comunidad de usuarios y desarrolladores, quienes usualmente trabajan en más de un lenguaje de computación, desarrollaron una herramienta más poderosa que pudiera trabajar con código de lenguajes como Julia (Ju), Python (Py), y R, lo cual originó la herramienta más utilizada en nuestros días: Jupyter.

Jupyter es una herramienta web gratuita y de código abierto, conocida como un *notebook* (o cuaderno de apuntes) computacional (J. Perkel, 2018). Jupyter permite a sus usuarios combinar código, salidas de las operaciones computacionales, texto explicativo y recursos multimedia en un solo documento. Dicha herramienta es tan poderosa que se ha popularizado tanto en la comunidad científica como en la industria. La plataforma más popular para compartir código en línea, GitHub, encontró un crecimiento de *notebooks* de Jupyter, compartidos por sus usuarios, de solamente 200,000 *notebooks* en 2015 hasta 2.5 millones en 2018 (J. Perkel, 2018). Jupyter es una herramienta tan potente que incluso ha

iniciado una tendencia nueva en la publicación de artículos científicos: que los autores incluyan un Jupyter Notebook con su código en vez de copiar-pegar como texto plano.

Un Jupyter Notebook tiene dos componentes. Primero, el usuario introduce su código en celdas a través de una página en un navegador web (p. ej. Google Chrome, Mozilla, Safari). Después, el navegador le pasa el código a un kernel que corre en la computadora, aunque no lo veamos, el cual regresa los resultados al navegador web para ser desplegados. El kernel típicamente vive en la computadora del usuario, aunque también existen alternativas que pueden correr en la nube (como *Google's Colaboratory project*). Existen herramientas adicionales como JupyterLab y JupyterHub, las cuales aumentan el potencial y la usabilidad de Jupyter, pero primero lo primero. Por lo tanto, en este trabajo decidimos adoptar Jupyter Notebooks para la elaboración de tutoriales o manuales interactivos para el manejo de datos de Ciencias de Tierra en Python. En la tabla 2, puedes encontrar la relación de ejercicios propuestos en este material.

1.5 ¿Cómo ejecuto código?

Existen varias maneras de ejecutar un código de Python. La más sencilla requiere utilizar el *interactive prompt* o línea de comando interactiva, a la cual se puede acceder desde IPython, en la terminal, o en Spyder. Sin embargo, la gran desventaja de programar de esa manera es que todo el código se ejecuta tan pronto como se escribe, y cuando se requiere ejecutar un conjunto de acciones en un orden determinado, puede volverse caótico. Por lo tanto, usualmente resulta más conveniente hacer uso de un editor de texto, el cual servirá para guardar las líneas de código, que después se ejecutarán en el orden en el que fueron escritas. Usualmente, un código de Python tiene la terminación .py o .ipynb.

1.6 Mi primer programa en Python

El ejemplo clásico del primer programa en cualquier lenguaje de programación es imprimir en pantalla "Hola mundo". Si ya instalaste Anaconda, ve a algunas de las opciones mencionadas para acceder a la línea de comando, y escribe: *print("Hola mundo")* y presiona *Enter* o da clic en ejecutar. Listo, requisito cumplido. Nota que la sintaxis es particularmente simple comparada con algunos otros lenguajes de programación.

1.7 Reglas y recomendaciones básicas para nombrar carpetas y archivos

Existen algunas reglas básicas de nomenclatura que se tienen que cumplir para programar. Las reglas son aplicables tanto al nombre de los archivos como al *path*:

-No usar acentos ni caracteres especiales, por ejemplo: '+$%^"?)(*

-No usar espacios, se pueden usar guiones bajos en su lugar. Como en: mi_archivo.py

-Usar solamente minúsculas (dentro de lo posible)

-Usar la ruta más corta posible

-No comenzar los nombres de los archivos con números

-Usar puntos solamente para la extensión de los archivos

-Usar nombres descriptivos para los archivos

-Usar el sufijo v01, v02, v03, etc. para las versiones de un mismo archivo

Un programador experimentado podría darle la vuelta a estas reglas y debatir que no es necesario cumplirlas todas, ni en todos los casos. Algún otro programador que, además de

experimentado, haya llegado a un nivel de iluminación superior a través de la purificación conferida por sus preciosas lágrimas derramadas a las 4 de la mañana porque su código para la entrega final no corría, podrá dar fe de que cumplir estas reglas puede ahorrarle muchos dolores de cabeza a alguien que va comenzando. Simplemente, comparemos la ruta a este *script*:

/Volumes/yo_mero/geoestadistica/nube_puntos_v04.py

contra esta otra ruta:

/Volumes/soy Batman XD/My Cloud Drive/UNAM/FI/8o. semestre/Geoestadística/tareas y ejercicios/códigos de prueba/intento final?/mi código final final ahora sí finalísimo ya porfavor!.py

Es gracioso porque es real. Algunas librerías o programas no fueron hechos para lidiar con espacios, caracteres especiales, letras que solo existen en español, o con rutas complicadas o del sistema (en donde a veces Python no tiene permisos de ejecutar o editar archivos). Además, entre más complicado es el nombre de un archivo, más difícil será llegar a él y leerlo o ejecutarlo desde un *script* de Python.

1.8. Aplicaciones en las geociencias

Las carreras de Ciencias de la Tierra generalmente utilizan sets de datos de gran tamaño. Dichos sets de datos provienen de logueo de núcleos de roca, información de registros geofísicos de pozo, de técnicas geofísicas (tomadas en superficie), información de orientación de planos de discontinuidad, concentraciones de elementos químicos, etc. Muy frecuentemente, las tareas relacionadas a la limpieza, tratamiento y ploteo de datos son

intensivas, repetitivas, y requieren seguir una serie determinada de pasos. Además, las tecnologías emergentes de apoyo para la exploración geológica (e.g. LiDAR, fotogrametría a partir de adquisiciones fotográficas con drones, percepción remota multiespectral), han abierto la puerta para la generación de grandes conjuntos de datos potencialmente aplicables a la exploración geológica. Por lo tanto, la necesidad de programar rutinas de procesado sistemático de datos se ha incrementado.

Los datos geológicos tienen particularidades que no siempre se apegan a ejemplos generalizados de programación. Por ejemplo, los datos geológicos pueden tener orientación expresada en geometría esférica, pueden pertenecer a espacios n-dimensionales con $n>3$, pueden tener distribuciones multimodales, y, aunado a las otras características mencionadas, pueden tener una coordenada tridimensional en el espacio geográfico. Por lo tanto, se cae en muchos casos especiales para el tratamiento de datos. Aprender a manejar diferentes programas para cada caso es imposible, además de que en muchas ocasiones se requiere de una licencia de paga.

Dados los muchos retos que implica el procesar y analizar datos geológicos, es necesario apoyarse de una comunidad a la que le gusten las rocas y las computadoras. Justamente, el eslogan de *The Software Underground* (https://softwareunderground.org) es: *The place for scientists that like rocks and computers.* Dicha comunidad tiene canales de conversación en Slack (herramienta de comunicación en equipo), recopilaciones de librerías disponibles en línea, bases de datos, blogs, calendarios de eventos, e incluso mercancía. Algunas otras comunidades que se especializan en el manejo de datos geoespaciales con Python, son el *Earth Lab* (https://www.earthdatascience.org/tutorials/python/), *Pangeo* (https://pangeo.io/#what-is-pangeo) y *Earthpy* (http://earthpy.org). En la Tabla 1, se mencionan librerías desarrolladas por la comunidad de Ciencias de la Tierra. Evidentemente, existe una cantidad de código, rutinas y librerías preexistentes que se pueden adoptar y modificar en Ciencias de la Tierra.

	Librería	Descripción	Vínculo
Geoestadística	pyKriging	Kriging n-dinensional	https://github.com/capaulson/pyKriging
	HPGL	LIbrería para geoestadística de alto rendimiento	https://github.com/hpgl/hpgl
	PyGSLIB	Estimaciones de recurso mineral	https://opengeostat.github.io/pygslib/index.html
	GeostatsPy	*Geostatistical Software Library* reimplentada en Python	https://github.com/GeostatsGuy/GeostatsPy
	GeoStat-Framework	Simulaciones geoestadísticas	https://github.com/GeoStat-Framework
Análisis espacial	geonotebook	Jupyter n*otebooks* de la NASA para visualización y análisis geoespacial	https://github.com/OpenGeoscience/geonotebook
	Verde	Procesado de datos geoespaciales	https://github.com/fatiando/verde
Geoquímica	Reaktoro	Modelación de sistemas químicamente reactivos	https://reaktoro.org/
	GeoPyTool	Aplicación para graficado geoquímico	https://github.com/GeoPyTool/GeoPyTool

	pyrolite	Transformación geoquímica y visualización	https://github.com/morganjwilliams/pyrolite
Geología Estructural	mplStereonet	Redes estereográficas en Python basadas en Matplotlib	https://github.com/joferkington/mplstereonet
	apsg	Análisis de geología estructural avanzada y visualización basada en Matplotlib	https://github.com/ondrolexa/apsg

Tabla 1. Relación de algunas de las librerías en Python para la implementación de rutinas de procesado y análisis de datos geológicos.

1.9 Encuestas de opinión

El sector industrial actual requiere de profesionales especializados en el tratamiento sistemático de datos geológicos a través de rutinas de programación. Históricamente, profesionales de la carrera de ingeniería geofísica tienen habilidades más desarrolladas en términos de programación. Sin embargo, los profesionales encargados del procesado de datos enfocados a la exploración geológica egresados de las escuelas a nivel nacional, típicamente ingenieros geólogos e ingenieros en minas y metalurgia, carecen de los fundamentos básicos de programación. A pesar de que los egresados y alumnos actuales de diferentes escuelas a nivel nacional consideran que saber programar es importante para el ejercicio de sus labores profesionales, no existe ningún material de apoyo para apoyar el aprendizaje de la programación enfocado al tratamiento y análisis de datos geológicos.

Las generaciones actuales de profesionales reportan que la programación sería de utilidad en áreas que típicamente manejan grandes conjuntos de datos geológicos, como lo son la

exploración minera a partir de datos geoquímicos y la exploración petrolera a partir de mediciones de registros geofísicos de pozos. Las tecnologías emergentes de apoyo para la exploración geológica (p. ej. LiDAR, fotogrametría a partir de adquisiciones fotográficas con drones, percepción remota multiespectral, etc.), además, han abierto la puerta para la generación de grandes conjuntos de datos potencialmente aplicables a la exploración geológica. Por lo tanto, la necesidad de programar rutinas de procesado sistemático de datos se ha incrementado.

Los alumnos de las carreras de ingeniería en Ciencias de la Tierra solamente llevamos fundamentos de programación en nuestro paso por Ciencias Básicas, y no tenemos un entrenamiento formal para el tratamiento de datos geológicos. Dichos datos tienen particularidades que no se apegan a ejemplos generalizados.

Considerando que el lenguaje de programación que marca el estándar actual de la industria es Python, este trabajo se enfoca en producir material didáctico que introduzca a los alumnos al manejo de datos geológicos, y de esa manera contribuir a formar profesionales con la capacidad y habilidad de procesar datos de manera programática en el desarrollo de sus actividades profesionales en la industria.

El análisis y aseveraciones anteriores se basan principalmente en una encuesta realizada mediante el recurso en línea "Formularios de Google" para conocer la situación actual de la programación aplicada a las Ciencias de la Tierra.

Se encuestó a 3 sectores: alumnos, egresados y docentes. Los resultados obtenidos se tomaron de 170 alumnos, 74 egresados y 30 docentes, sumando un total de 274 personas encuestadas.

- Egresados: fueron encuestados aquellos que han laborado o se encuentran laborando actualmente en el área de Ciencias de la Tierra, en el sector gubernamental, así como en el sector privado y que pertenecen a distintas escuelas en el país donde se imparten carreras de Ciencias de la Tierra.

- Alumnos: fueron encuestados alumnos pertenecientes a las carreras de Ingeniería Geológica, Geofísica y Minas y Metalurgia que estén cursando asignaturas de sexto semestre en adelante en la Facultad de Ingeniería, UNAM.

- Docentes: fueron encuestados profesores que imparten asignaturas para las carreras de Ingeniería Geológica, Geofísica y Minas y Metalurgia.

La encuesta realizada arrojó los siguientes resultados:

- Para los egresados, el 82% de los encuestados asegura que la programación es una herramienta necesaria en el campo laboral, y el 55% afirma que sabe programar. La mayoría aprendió en la escuela y en cursos extras.
- Para los estudiantes de Ingeniería Geológica: el 58% no sabe programar y el 42% afirma saber programar. De los que sí programan/programaron, lo hicieron en C, Python, R, Java y Matlab.
- Para los estudiantes de Ingeniería en Minas y Metalurgia: el 58% no sabe programar y el 42% afirma saber programar. De los que sí programan/programaron lo hicieron en C, Matlab y Python.
- Para los estudiantes de Ingeniería Geofísica: el 62% sabe programar y el 38% no sabe programar. De los que sí programan/programaron lo hicieron en C, Matlab, Fortran y Python.
- De los alumnos que afirman saber programar, un 82% aprendió en las materias de ciencias básicas, en las materias de la carrera, así como en cursos intersemestrales cortos, mientras que el 18% restante aprendió en cursos externos a la UNAM.
- Para los docentes, el 63% ha programado en su vida profesional y la gran mayoría lo ha hecho en Fortran y Python. El 26% considera que Python es más fácil comparado con otros lenguajes de programación, el 21% que tiene el mismo nivel de dificultad, mientras que el 53% restante no tiene idea del nivel de dificultad. Sólo el 10% de los encuestados ha pedido a sus alumnos que programen.

2. AMBIENTE COMPUTACIONAL PARA LA PROGRAMACIÓN

En este capítulo se presenta el ambiente computacional adecuado para trabajar con Python en nuestro equipo. Se propone utilizar Ubuntu como sistema operativo, también encontrarás todos los pasos para que puedas instalarlo, así como el procedimiento de descarga de Python y la preparación de nuestro equipo con el fin de utilizar el entorno de Jupyter Notebooks y así la realización de los ejercicios propuestos en este material didáctico.

2.1 Ubuntu, ventajas, desventajas y requerimientos

Ubuntu es un sistema operativo de código abierto que fue lanzado por primera vez el 20 de octubre de 2004 y es una distribución de GNU/Linux la cual está basada en Debian y que se distribuye bajo una licencia libre.

Ubuntu corre en computadoras de escritorio y servidores. A continuación, se mencionan algunas ventajas y desventajas del uso de Ubuntu.

Ventajas:

- Es gratis. - Lo cual lo vuelve muy accesible (también puedes hacer donaciones a los desarrolladores).
- Compatibilidad de escritorios. - Si no te gusta algún detalle del ambiente puedes reemplazarlo por uno diferente, ya que utiliza el escritorio GNOME (así como todas las distribuciones basadas en GNU/Linux). Por lo tanto, puedes hacer diversas modificaciones para que sean visualmente de tu agrado.
- Es mucho más seguro. - Todos los sistemas son susceptibles a malware. Sin embargo, si lo comparamos con Windows, los riesgos asociados por "malware" son menores. Esto se debe a que existe una mayor cantidad de usuarios en Windows lo que lo hace más susceptible a que se creen virus para el software antes mencionado. Conclusión: Linux no se rompe tan fácil como Windows.
- Existen otras variantes (*flavours*). - Ubuntu cuenta con diversas variantes que se ajustan a las necesidades específicas de cada usuario. Por ejemplo, Kubuntu que presenta un entorno moderno y elegante; para sistemas con baja configuración,

existe Lubuntu y Xubuntu. Otro ejemplo es Edubuntu el cual está enfocado para el uso en escuelas e instituciones educativas, entre otros.

- Línea de comando. - Aquí es donde los usuarios avanzados pueden contribuir en el desarrollo de código y documentación para arreglar *bugs* así como agregar características.
- Comunidad de Soporte. - Tal como otros proyectos de GNU/Linux, Ubuntu cuenta con una fuerte comunidad de soporte y un foro donde la comunidad puede resolver tus dudas (https://www.ubuntu.com/support).
- Software disponible. - Puedes descargar cientos de aplicaciones mediante "Software de Ubuntu".
- Recursos. - Ubuntu consume menos recursos que Windows (incluso con hardware antiguo) y por lo tanto es más rápido que Windows.

Desventajas:

- Existe software en Windows que no tiene una alternativa en Ubuntu (aunque generalmente se puede ejecutar si ocupas herramientas de virtualización como Virtualbox o Wine).
- Es menor la cantidad de personas que pueden darle soporte a tu equipo que en Windows. Pero no te preocupes, puedes solucionarlo apoyándote de la comunidad de soporte antes mencionada.

Requerimientos para la instalación de Ubuntu

Según la página oficial de Ubuntu, los requerimientos mínimos son:

- 2 GHz procesador *dual core*
- 4 GB en RAM
- 25 GB de espacio en disco duro para la instalación de Ubuntu y el software básico, para el usuario mínimo otros 25, así que en total 50 GB
- VGA con capacidad de resolución 1024x768
- CD/DVD o una memoria USB
- Acceso a Internet

2.2 Python

Python es un lenguaje de programación orientado a objetos, amigable, fácil de aprender y de código abierto. A continuación, se mencionan algunas características notables de Python:

- Utiliza una sintaxis elegante, haciendo que los programas que escribes sean más fáciles de leer.
- Es un lenguaje fácil de usar (ideal para principiantes) y que simplifica el funcionamiento de los programas. Haciendo que Python sea ideal para el desarrollo de prototipos y otras tareas de programación.
- Cuenta con una extensa biblioteca para tareas básicas como conectarse a los servidores, búsqueda de texto con expresiones regulares o lectura y modificación de archivos.
- El modo interactivo de Python facilita la prueba de fragmentos de código.
- Se extiende fácilmente agregando nuevos módulos implementados en un lenguaje compilado como C o C++.
- Puedes integrarlo en una aplicación para proporcionar una interfaz programable.
- Es ejecutable en cualquier sistema operativo, incluyendo GNU/Linux, Mac Os X y Windows.
- Es software libre en dos sentidos. Descargar y usar Python, así como incluirlo en una aplicación no tiene costo, además puedes modificarlo y re distribuirlo porque a pesar de que el lenguaje tiene derechos de autor, está disponible bajo una licencia de código abierto.

A continuación, características del lenguaje:

- Cuenta con una variedad de tipos de datos: números, *strings* (cadenas), listas y diccionarios.
- Python permite programación orientada a objetos.
- El código puede ser agrupado en módulos y paquetes (*packages*).
- Tiene tipado dinámico, es decir, el tipo de dato se determinará en tiempo de ejecución, según el valor que se le asigne a una variable.
- La administración automática de memoria de Python le libera de tener que asignar y liberar memoria manualmente en su código.

2.3 Anaconda

Anaconda distribution es una distribución de Python que funciona como un gestor de entorno y un gestor de paquete, posee una colección de más de 720 paquetes de código abierto.

Se agrupa en 4 sectores: Anaconda Navigator, Anaconda Project, las librerías de Ciencia de datos y Conda. Para nuestros fines utilizaremos Anaconda Navigator.

2.4 Github

Github es una plataforma de desarrollo. Desde código abierto hasta para negocios, puedes hospedar y revisar código, gestionar productos y crear software junto a otros 50 millones de desarrolladores.

Para poder tener acceso a la misma, debes tener una cuenta, sólo necesitas un correo electrónico y crear un nombre de usuario y una contraseña. En este material se incluyen algunos enlaces en Github, por lo que te sugerimos abras una cuenta en Github (si aún no la tienes).

2.5 Instalación de Ubuntu

Existen diversas formas para instalar el Sistema Operativo de Ubuntu en nuestra computadora, a continuación, se describen:

1. Máquina virtual. - Con esta forma no necesitas particionar ni tocar el disco duro principal y puedes usar los dos sistemas operativos al mismo tiempo. Un punto importante es que requieres de un equipo que cumpla con ciertas características para que puedas utilizar Ubuntu en una máquina virtual y que tu computadora no se vuelva lenta. Una ventaja es que la puedes exportar para que otro usuario trabaje la "máquina" como tú la dejaste.

   Algunos ejemplos de máquinas virtuales gratuitas son: Virtualbox, VMware Workstation Player, Virtual PC, Hyper-V, entre otras.

2. Partición del disco duro. - Esta forma te permite tener dos sistemas operativos instalados en tu computadora, es decir, el software que viene instalado de fábrica y Ubuntu. Por lo tanto, al encender tu equipo, tú decides con que sistema operativo

quieres trabajar. En este caso, se explica la instalación de Ubuntu en una computadora con Windows.

Las características de mi equipo son:

- Sistema Operativo: Windows 10, 64 bits
- Procesador: Intel Celeron® CPU N3050 @ 1.60GHZ x 2
- RAM: 4 Gb

Puedes encontrar el tutorial de instalación en el anexo

3.EJERCICIOS DE PROGRAMACIÓN EN PYTHON

A través de Jupyter Notebooks podrás visualizar diversos ejercicios donde se realiza el análisis de datos geológicos.

Aprenderás a importar los datos desde un archivo .csv, a importar las librerías necesarias para llevar a cabo los ejercicios, así como a "limpiar" los datos para posteriormente analizar datos de interés y realizar diversos gráficos.

A continuación, encontrarás un resumen de los datos geológicos para cada ejercicio.

E0: Primeros pasos	En este ejercicio conocerás el entorno de trabajo de Júpiter Notebook e imprimirás tu primer línea de código
E1: Lectura de datos desde una tabla	En este ejercicio podrás analizar la base de datos de minerales registrados ante el IMA
E2: Cálculo de medidas de tendencia central	En este ejercicio analizarás datos sobre sismos de magnitud mayor a 5.5 en el mundo

E3: Covarianza y correlación de datos	En este ejercicio podrás calcular la covarianza y correlación de datos de tiempo de erupción de un geiser
E4: Estandarización de datos y semivariograma en Python	En este ejercicio estandarizarás datos y visualizarás semivariogramas a partir de datos de permeabilidad y porosidad de rocas
E5: Graficando perfil geológico	En este ejercicio graficarás un perfil geológico a partir de coordenadas UTM
E6: Red de Schmidt y red de Wulff	En este ejercicio podrás graficar puntos y planos en ambas redes
E7: Manipulación y visualización de una nube de puntos en Python	En este ejercicio podrás visualizar nubes de puntos que corresponden con datos de un afloramiento en Ciudad Universitaria
E8: Segmentación de nubes de puntos basada en color y K-*means* en Python	En este ejercicio podrás segmentar la nube de puntos basada en color así como clasificar por K-*means* (técnica no supervisada en *Machine Learning*)
E9: Búsqueda de planos de discontinuidad a partir de una nube de puntos	En este ejercicio podrás utilizar las coordenadas cartesianas de la nube de puntos para calcular planos representativos de una discontinuidad en términos de rumbo y echado

Tabla 2. Relación de ejercicios y resumen de actividades a realizar

Los ejercicios propuestos los encuentras en el Apéndice H o también puedes acceder al siguiente enlace para descargarlos: https://github.com/brendorts/ejercicios-python-

4.CONCLUSIONES Y RECOMENDACIONES

Es importante que los alumnos de las carreras de Ciencias de la Tierra se introduzcan al manejo de datos geológicos de manera programática en Python, considerando que es el lenguaje de programación que marca el estándar actual de la industria. Teniendo ese objetivo en mente, se presenta este panorama general junto con los ejercicios de este material didáctico, y así contribuir a formar profesionales con la capacidad y habilidad de procesar datos de manera programática en el desarrollo de sus actividades profesionales en la industria y la academia. En los ejercicios desarrollados para este trabajo se busca proveer al alumno de las herramientas básicas para agilizar la adopción, modificación y ejecución de códigos preexistentes, pero donde se requiere que el alumno tenga fundamentos básicos de programación. Las librerías e implementaciones deseables que los alumnos conozcan son NumPy, SciPy, Matplotlib, y Pandas, a través de una interfaz interactiva como Jupyter. La instalación básica de Python, Jupyter, y las librerías elementales mencionadas se pueden lograr a través de Anaconda. Las librerías especializadas desarrolladas independientemente por una comunidad activa de amantes de las rocas y las computadoras utilizan dicha estructura básica de Python.

Si como lector has llegado al final de este documento, y decidiste darle una oportunidad a Python, te recomiendo que hagas lo siguiente:

1. Dirígete a anaconda.com desde tu navegador

2. Descarga la distribución adecuada para tu computadora y sistema operativo

3. Haz tu primer programa con *print("Hola mundo")*

Cuando logres ejecutar esos pasos con éxito, estarás listo para seguir con los ejercicios de este material. *Bon voyage!*

REFERENCIAS CITADAS

1. (S/f-b). Cmu.edu. Recuperado el 30 de mayo de 2021, de https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat
2. About the Ubuntu project. (n.d.). Ubuntu. Recuperado el 31 de marzo de 2022, de https://ubuntu.com/about
3. BeginnersGuide - Python Wiki. (s/f). Python.org. Recuperado el 30 de octubre de 2020, de https://wiki.python.org/moin/BeginnersGuide
4. Chemkaeva, D. (s/f). IMA Database of Mineral Properties ⁻Data set⁻.
5. Enterprise open source support. (s/f). Ubuntu. Recuperado el 30 de enero de 2022, de https://www.ubuntu.com/support
6. Geostatspy. (s/f). PyPI. Recuperado el 30 de enero de 2022, de https://pypi.org/project/geostatspy/
7. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array Programming with NumPy. *Nature*, *585*(June), 357–362. https://doi.org/10.1038/s41586-020-2649-2
8. Koldunov, N. (s/f). EarthPy - python for geosciences. Earthpy.Org. Recuperado el 30 de enero de 2022, de http://earthpy.org/tag/python-for-geosciences.html
9. Lin, J. W.-B. (2012). Why Python Is the Next Wave in Earth Sciences Computing. Bulletin of the American Meteorological Society, 93(12), 1823–1824. doi:10.1175/bams-d-12-00148.1
10. Lutz, M. (2007). *Learning Python. Icarus*. https://doi.org/10.1016/0019-1035(89)90077-8
11. Matplotlib — Visualization with Python. (s/f). Matplotlib.org. Recuperado el 30 de enero de 2022, de https://matplotlib.org/
12. Michael J. Pyrcz, The University of Texas at Austin. (s/f). Professor Michael J. Pyrcz, the university of Texas at Austin. Professor Michael J. Pyrcz, The University of Texas at Austin. Recuperado el 30 de enero de 2022, de https://michaelpyrcz.com/
13. Perkel, J. (2018). By Jupyter, it all makes sense. *Nature*, *563*(November), 145–146. Recuperado de https://colab.research.google.
14. Perkel, J. M. (2015). Pick up Python. *Nature*, *518*(7537), 125–126. https://doi.org/10.1038/518125a
15. Plotly: The front end for ML and data science models. (s/f). Plotly.com. Recuperado el 29 de diciembre de 2021, de https://plotly.com/
16. plotlygraphs. (2019, julio 3). Static image export. Plotly.com. https://plotly.com/python/static-image-export/
17. Poux, F. (2020). 5-Step Guide to generate 3D meshes from point clouds with Python. Retrieved 1 March 2021, from https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba
18. Poux, F., Neuville, R., Nys, G., & Billen, R. (2018). 3D Point Cloud Semantic Modelling: Integrated Framework for Indoor Spaces and Furniture. Remote Sensing, 10(9), 1412. doi: 10.3390/rs10091412

19. Pyrcz, M. (s/f-b). sample_data.csv at master · GeostatsGuy/GeoDataSets.
20. Pyrcz, M.J., Jo. H., Kupenko, A., Liu, W., Gigliotti, A.E., Salomaki, T., and Santos, J., 2021, GeostatsPy Python Package, PyPI, Python Package Index, https://pypi.org/project/geostatspy/.
21. Python Software Foundation. (2020). Python org. Recuperado el 18 de septiembre de 2020, de https://www.python.org/
22. She, H. (2014). Interactive Notebooks: Sharing the Code. *Nature*, *515*(10), 151–152.
23. Solano, D. (s.f.). Videos [Canal de Youtube]. YouTube. Recuperado el 19 de octubre de 2020, de https://www.youtube.com/channel/UCmeMXDwmAyZfmgi3ZiXffYw/videos
24. The MathWorks, I. (2015). MATLAB Release 2015b. Natick, Massachusetts, United States.
25. US Geological Survey. (s/f). Significant Earthquakes, 1965-2016 [Data set].
26. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Vázquez-Baeza, Y. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2
27. Welcome to. (s/f). Python.org. Recuperado el 30 de enero de 2022, de https://www.python.org/doc/
28. Zhou, Q., Park, J., & Koltun, V. (2021). Open3D – A Modern Library for 3D Data Processing. Recuperado el 2 de marzo de 2021, de http://www.open3d.org/

LISTA DE APÉNDICES

APÉNDICE A

ENCUESTAS REALIZADAS

EGRESADOS

	Carrera	Año de egreso	Escuela de egreso	Rango de edad (años)	¿Sabes programar?	¿Qué tan útil consideras programar en la vida profesional? 1:poco, 5: mucho	¿Dónde aprendiste a programar? (Puedes seleccionar una o varias opciones)	¿Consideras que saber programar es una herramienta necesaria en el mundo laboral?
1	Ingeniería Geológica/Geología	2017	UNAM	21-30	No			Sí
2	Ingeniería Geológica/Geología	2018	UNAM	21-30	Sí	5	Escuela	Sí
3	Ingeniería Geológica/Geología	2016	UNAM	31-40	Sí	4	Cursos extra	Sí
4	Ingeniería Geofísica/Geofísica	2015	UNAM	21-30	Sí	2	Escuela	Sí
5	Ingeniería Geológica/Geología	2011	UNAM	21-30	No			Sí
6	Ingeniería en Minas y Metalurgia/Afines	2018	UNAM	21-30	No			Sí
7	Ingeniería Geofísica/Geofísica	2018	BUAP	21-30	Sí	3	Escuela	Sí
8	Ingeniería en Minas y Metalurgia/Afines	2017	UES	21-30	No			No
9	Ingeniería Geológica/Geología	2016	Uaz	21-30	Sí	4	Por mi cuenta	Sí
10	Ingeniería Geofísica/Geofísica	2014	BUAP	21-30	No			No
11	Ingeniería Geofísica/Geofísica	2017	UNAM	21-30	Sí	3	Escuela	Sí
12	Ingeniería en Minas y Metalurgia/Afines	0	UNAM	21-30	Sí	5	Escuela	Sí
13	Ingeniería Geológica/Geología	2015	UNAM	21-30	No			Sí
14	Ingeniería Geológica/Geología	2016	UG	21-30	No			Sí
15	Ingeniería Geofísica/Geofísica	2014	UNAM	21-30	Sí	4	Escuela, trabajo	Sí
16	Ingeniería Geofísica/Geofísica	2012	UNAM	21-30	Sí	3	Escuela	No
17	Ingeniería Geológica/Geología	2018	UNAM	21-30	Sí	5	Cursos extra, trabajo	Sí
18	Ingeniería Geofísica/Geofísica	2018	UNAM	21-30	No			No

19	Ingeniería Geológica/Geología	2007	UNAM	31-40	Sí	4	Cursos extra, por mi cuenta	Sí
20	Ingeniería Geológica/Geología	2019	UNAM	21-30	No			Sí
21	Ingeniería Geológica/Geología	2017	UNAM	21-30	No			Sí
22	Ingeniería Geológica/Geología	2019	UNAM	21-30	Sí	3	Cursos extra, por mi cuenta	Sí
23	Ingeniería Geológica/Geología	2008	UNAM	31-40	No			Sí
24	Ingeniería en Minas y Metalurgia/Afines	2019	UNAM	21-30	Sí	5	Por mi cuenta	Sí
25	Ingeniería Geofísica/Geofísica	2019	UNAM	21-30	Sí	5	Escuela, cursos extra	Sí
26	Ingeniería Geofísica/Geofísica	2014	UNAM	21-30	Sí	5	En la escuela, Por mi cuenta	Sí
27	Ingeniería Geológica/Geología	1975	IPN	Mayor de 50	No			Sí
28	Ingeniería en Minas y Metalurgia/Afines	1988	UNISON	Mayor de 50	Sí	4	Por mi cuenta	No
29	Ingeniería Geológica/Geología	2017	IPN	21-30	No			Sí
30	Ingeniería Geológica/Geología	2018	IPN	21-30	No			No
31	Ingeniería Geofísica/Geofísica	2013	UNAM	21-30	Sí	5	Escuela, por mi cuenta	Sí
32	Ingeniería Geológica/Geología	2015	IPN	21-30	No			Sí
33	Ingeniería Geofísica/Geofísica	2017	IPN	21-30	Sí	3	Escuela	Sí
34	Ingeniería Geológica/Geología	2019	UNAM	21-30	Sí	5	Escuela, por mi cuenta	Sí
35	Ingeniería Geológica/Geología	2017	UNAM	21-30	Sí	5	Escuela, por mi cuenta, trabajo	Sí
36	Ingeniería Geofísica/Geofísica	2018	IPN	21-30	Sí	5	Por mi cuenta	Sí
37	Ingeniería Geológica/Geología	2010	UNAM	21-30	No			Sí
38	Ingeniería en Minas y Metalurgia/Afines	2019	UNAM	21-30	Sí	5	Escuela	Sí
39	Ingeniería Geofísica/Geofísica	2015	UNAM	21-30	Sí	3	Escuela	Sí
40	Ingeniería Geofísica/Geofísica	2017	Universidad Nacional de San Agustin de Arequipa	21-30	Sí	4	Cursos extra, por mi cuenta	Sí
41	Ingeniería Geofísica/Geofísica	1999	UNAM	41-50	Sí	5	Escuela, por mi cuenta	Sí

42	Ingeniería en Minas y Metalurgia/Afines	2015	UNAM	21-30	Sí	5	Cursos extra, por mi cuenta	No
43	Ingeniería Geofísica/Geofísica	2019	UNAM	21-30	Sí	5	Escuela, cursos extra, por mi cuenta, en mi trabajo	Sí
44	Ingeniería en Minas y Metalurgia/Afines	2016	UNAM	31-40	No			Sí
45	Ingeniería Geológica/Geología	2017	IPN	21-30	No			Sí
46	Ingeniería en Minas y Metalurgia/Afines	2019	UNAM	21-30	No			Sí
47	Ingeniería Geofísica/Geofísica	2015	UNAM	21-30	Sí	3	Escuela, cursos extra	Sí
48	Ingeniería Geológica/Geología	2019	UNAM	21-30	Sí	2	Escuela, trabajo	No
49	Ingeniería Geofísica/Geofísica	2019	UNAM	21-30	Sí	4	Escuela, por mi cuenta, trabajo	No
50	Ingeniería Geofísica/Geofísica	2016	UNAM	21-30	Sí	5	Escuela, cursos extra, por mi cuenta	Sí
51	Ingeniería Geofísica/Geofísica	2012	UNAM	31-40	Sí	5	Cursos extra, por mi cuenta	Sí
52	Ingeniería Geofísica/Geofísica	2012	IPN	31-40	Sí	5	Por mi cuenta, trabajo	Sí
53	Ingeniería en Minas y Metalurgia/Afines	2012	UJED-FCQ	31-40	No			Sí
54	Ingeniería Geofísica/Geofísica	2014	Universidad Nacional San Agustín de Arequipa	21-30	Sí	4	Cursos extra	Sí
55	Ingeniería Geofísica/Geofísica	2019	UNAM	21-30	Sí	5	Escuela, cursos extra, por mi cuenta	Sí
56	Ingeniería Geológica/Geología	2010	IPN	31-40	No			Sí
57	Ingeniería Geológica/Geología	2010	UNAM	41-50	No			Sí
58	Ingeniería Geológica/Geología	1960	UNAM	Mayor de 50	No			Sí
59	Ingeniería Geológica/Geología	1960	UNAM	Mayor de 50	No			Sí
60	Ingeniería Geológica/Geología	2017	UNAM	21-30	No			Sí

61	Ingeniería Geofísica/Geofísica	2017	UNAM	21-30	No			Sí
62	Ingeniería en Minas y Metalurgia/Afines	2006	BUAP	31-40	No			Sí
63	Ingeniería Geológica/Geología	2018	UAZ	21-30	No			Sí
64	Ingeniería Geológica/Geología	2013	UNAM	31-40	Sí	4	Escuela	Sí
65	Ingeniería en Minas y Metalurgia/Afines	1984	Universidad de Santiago de Chile	Mayor de 50	Sí	5	Trabajo	Sí
66	Ingeniería en Minas y Metalurgia/Afines	1987	BUAP	Mayor de 50	Sí	5	Trabajo	Sí
67	Ingeniería Geofísica/Geofísica	2000	UNAM	41-50	Sí	5	Escuela, cursos extra, por mi cuenta, trabajo	Sí
68	Ingeniería Geológica/Geología	2007	IPN	31-40	No			No
69	Ingeniería Geológica/Geología	2013	IPN	21-30	No			Sí
70	Ingeniería Geológica/Geología	2008	IPN	31-40	Sí	5	Escuela, en trabajo	Sí
71	Ingeniería Geológica/Geología	2012	UAEH	31-40	No			No
72	Ingeniería Geológica/Geología	2012	Unal	31-40	No			No
73	Ingeniería Geológica/Geología	2010	IPN	31-40	Sí	2	Escuela	No
74	Ingeniería Geológica/Geología	2015	IPN	21-30	No			Sí

Tabla A1. Resultados de la encuesta realizada a egresados

ALUMNOS

	Carrera	Generación	Semestre en curso	¿Sabes programar?	¿En qué lenguaje programaste?	¿Qué tanto consideras que sabes programar? 1:poco, 5:mucho	¿Dónde aprendiste a programar?	¿Has cursado alguna asignatura donde hayas programado para resolver un problema (aplicado a las geociencias)?	¿Consideras que saber programar es de utilidad en el campo laboral?	¿Has escuchado hablar sobre Python?	¿Estarías dispuesto a llevar un curso donde aprendas Python?
1	Ingeniería Geológica	2017	Sexto	No					Sí	Sí	Sí
2	Ingeniería Geológica	2018	Sexto	No					Sí	Sí	Sí
3	Ingeniería Geológica	2017	Sexto	Sí	C	4	Anexo de Ingeniería (Ciencias Básicas)	No	No	Sí	No
4	Ingeniería Geológica	2015	Terminé créditos	Sí	R	3	Cursos fuera de la UNAM	No	Sí	Sí	Sí
5	Ingeniería Geológica	2014	Terminé créditos	Sí	Python, R, Java	5	Cursos fuera de la UNAM	No	Sí	Sí	No
6	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
7	Ingeniería Geológica	2007	Terminé créditos	Sí	C, Matlab, Python	3	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
8	Ingeniería Geológica	2018	Terminé créditos	No					Sí	Sí	Sí
9	Ingeniería Geológica	2017	Octavo	No					Sí	Sí	Sí

10	Ingeniería Geológica	2013	Terminé créditos	No					Sí	Sí	Sí
11	Ingeniería Geológica	2014	Terminé créditos	Sí	HTML	2	Cursos fuera de la UNAM	No	Sí	Sí	Sí
12	Ingeniería Geológica	2013	Terminé créditos	No					Sí	Sí	Sí
13	Ingeniería Geológica	2016	Séptimo	Sí	C	2	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
14	Ingeniería Geológica	2015	Terminé créditos	Sí	Python	4	Cursos fuera de la UNAM	No	Sí	Sí	Sí
15	Ingeniería Geológica	2015	Octavo	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
16	Ingeniería Geológica	2015	Sexto	No					Sí	Sí	Sí
17	Ingeniería Geológica	2016	Décimo	Sí	C, Python	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
18	Ingeniería Geológica	2015	Terminé créditos	No					No	Sí	No
19	Ingeniería Geológica	2017	Octavo	No					Sí	Sí	Sí
20	Ingeniería Geológica	2015	Décimo	No					Sí	Sí	No
21	Ingeniería Geológica	2017	Séptimo	No					Sí	No	Sí
22	Ingeniería Geológica	2016	Noveno	Sí	C	3	Principal (Materias de la Carrera)	No	Sí	Sí	Sí
23	Ingeniería Geológica	2016	Décimo	No					Sí	Sí	Sí
24	Ingeniería Geológica	2018	Sexto	No					Sí	Sí	Sí

25	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
26	Ingeniería Geológica	2017	Octavo	Sí	C, Matlab	3	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
27	Ingeniería Geológica	2017	Octavo	Sí	C	2	Principal (Materias de la Carrera)	No	No	Sí	Sí
28	Ingeniería Geológica	2017	Octavo	Sí	C	4	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
29	Ingeniería Geológica	2016	Noveno	Sí	C, Matlab	3	Anexo de Ingeniería (Ciencias Básicas)	No	No	Sí	No
30	Ingeniería Geológica	2015	Octavo	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
31	Ingeniería Geológica	2017	Octavo	Sí	C	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
32	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
33	Ingeniería Geológica	2015	Sexto	Sí	C, Matlab, Java	4	Cursos fuera de la UNAM	Sí	Sí	No	Sí
34	Ingeniería Geológica	2017	Séptimo	No					No	No	No
35	Ingeniería Geológica	2017	Séptimo	No					Sí	No	Sí
36	Ingeniería Geológica	2017	Séptimo	No					Sí	Sí	Sí
37	Ingeniería Geológica	2017	Octavo	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí

38	Ingeniería Geológica	2015	Terminé créditos	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
39	Ingeniería Geológica	2018	Noveno	No					Sí	Sí	Sí
40	Ingeniería Geológica	2010	Terminé créditos	No					No	Sí	Sí
41	Ingeniería Geológica	2015	Terminé créditos	Sí	C	2	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
42	Ingeniería Geológica	2016	Décimo	No					Sí	Sí	Sí
43	Ingeniería Geológica	2015	Noveno	No					Sí	Sí	Sí
44	Ingeniería Geológica	2018	Sexto	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
45	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
46	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
47	Ingeniería Geológica	2015	Séptimo	No					Sí	No	Sí
48	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
49	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
50	Ingeniería Geológica	2016	Décimo	No					Sí	Sí	Sí
51	Ingeniería Geológica	2017	Octavo	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	No
52	Ingeniería Geológica	2018	Sexto	No					Sí	Sí	Sí

53	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
54	Ingeniería Geológica	2013	Terminé créditos	No					Sí	Sí	Sí
55	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
56	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
57	Ingeniería Geológica	2009	Terminé créditos	No					Sí	Sí	Sí
58	Ingeniería Geológica	2011	Terminé créditos	Sí	C, Fortran	3	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	No	Sí	Sí	Sí
59	Ingeniería Geológica	2016	Octavo	Sí	C	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
60	Ingeniería Geológica	2017	Octavo	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
61	Ingeniería Geológica	2010	Terminé créditos	No					No	Sí	No
62	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí
63	Ingeniería Geológica	2017	Octavo	Sí	C, Matlab	3	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
64	Ingeniería Geológica	2017	Octavo	Sí	C	2	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
65	Ingeniería Geológica	2015	Terminé créditos	No					Sí	Sí	Sí

66	Ingeniería Geológica	2015	Terminé créditos	No					No	Sí	Sí
67	Ingeniería Geológica	2015	Terminé créditos	Sí	Python	3	Cursos fuera de la UNAM	Sí	Sí	Sí	Sí
68	Ingeniería Geofísica	2015	Décimo	Sí	Matlab, Fortran, Python	2	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
69	Ingeniería Geofísica	2018	Sexto	No					Sí	Sí	Sí
70	Ingeniería Geofísica	2015	Terminé créditos	No					Sí	Sí	Sí
71	Ingeniería Geofísica	2017	Octavo	Sí	C, Fortran, Python	4	Cursos fuera de la UNAM	Sí	Sí	Sí	Sí
72	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab, Fortran	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
73	Ingeniería Geofísica	2012	Terminé créditos	Sí	C, Matlab, Fortran, Python, R	4	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
74	Ingeniería Geofísica	2014	Terminé créditos	No					Sí	Sí	Sí
75	Ingeniería Geofísica	2015	Décimo	No					Sí	Sí	Sí
76	Ingeniería Geofísica	2015	Décimo	Sí	C, Matlab	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
77	Ingeniería Geofísica	2018	Séptimo	Sí	Python	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí

78	Ingeniería Geofísica	2007	Terminé créditos	Sí	Matlab, Fortran, Python	3	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
79	Ingeniería Geofísica	2014	Terminé créditos	No					Sí	Sí	Sí
80	Ingeniería Geofísica	2016	Décimo	Sí	C, Matlab, Fortran, Python	4	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
81	Ingeniería Geofísica	2016	Octavo	Sí	C, Fortran, Python	3	Cursos fuera de la UNAM	No	Sí	Sí	Sí
82	Ingeniería Geofísica	2016	Décimo	No					Sí	Sí	Sí
83	Ingeniería Geofísica	2016	Octavo	Sí	C, Matlab, Python	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
84	Ingeniería Geofísica	2016	Octavo	No					Sí	Sí	Sí
85	Ingeniería Geofísica	2017	Octavo	Sí	C, Matlab, Python	3	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
86	Ingeniería Geofísica	2018	Sexto	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
87	Ingeniería Geofísica	2015	Terminé créditos	No					Sí	Sí	Sí
88	Ingeniería Geofísica	2016	Octavo	No					Sí	Sí	Sí

89	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab	2	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
90	Ingeniería Geofísica	2016	Octavo	No					Sí	Sí	Sí
91	Ingeniería Geofísica	2012	Terminé créditos	Sí	C, Matlab, Fortran	4	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
92	Ingeniería Geofísica	2017	Octavo	No					Sí	Sí	Sí
93	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab, Fortran	2	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
94	Ingeniería Geofísica	2018	Sexto	Sí	C, Python, Swith	2	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
95	Ingeniería Geofísica	2014	Terminé créditos	Sí	C, Fortran, Python	5	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
96	Ingeniería Geofísica	2015	Octavo	No					Sí	Sí	Sí
97	Ingeniería Geofísica	2017	Octavo	No					Sí	Sí	Sí
98	Ingeniería Geofísica	2015	Décimo	No					Sí	Sí	Sí
99	Ingeniería Geofísica	2017	Sexto	Sí	C, Python	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí

100	Ingeniería Geofísica	2018	Sexto	Sí	Java	2	Cursos fuera de la UNAM	Sí	Sí	Sí	Sí
101	Ingeniería Geofísica	2016	Décimo	Sí	C, Fortran, Python	4	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
102	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab, Python	3	Cursos fuera de la UNAM	Sí	Sí	Sí	Sí
103	Ingeniería Geofísica	2017	Octavo	No					Sí	Sí	Sí
104	Ingeniería Geofísica	2018	Sexto	No					Sí	Sí	Sí
105	Ingeniería Geofísica	2012	Décimo	Sí	C, Matlab, Fortran, Python	4	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
106	Ingeniería Geofísica	2015	Terminé créditos	No					Sí	Sí	Sí
107	Ingeniería Geofísica	2017	Sexto	Sí	C, Matlab	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
108	Ingeniería Geofísica	2017	Sexto	Sí	C, Python	4	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
109	Ingeniería Geofísica	2016	Octavo	Sí	C, Matlab, Python	2	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
110	Ingeniería Geofísica	2016	Séptimo	No					Sí	No	Sí
111	Ingeniería Geofísica	2017	Octavo	Sí	C, Matlab	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí

112	Ingeniería Geofísica	2016	Décimo	Sí	C, Matlab, Fortran, Python	3	Principal (Materias de la Carrera)	No	Sí	Sí	Sí
113	Ingeniería Geofísica	2015	Séptimo	Sí	C, Matlab	2	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	No	Sí	Sí	Sí
114	Ingeniería Geofísica	2013	Séptimo	Sí	Matlab, Fortran, Python	3	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
115	Ingeniería Geofísica	2015	Terminé créditos	Sí	C, Matlab, Fortran	4	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
116	Ingeniería Geofísica	2015	Décimo	Sí	Matlab	3	Principal (Materias de la Carrera)	Sí	Sí	No	Sí
117	Ingeniería Geofísica	2014	Terminé créditos	No					No	Sí	Sí
118	Ingeniería Geofísica	2018	Sexto	No					Sí	Sí	Sí
119	Ingeniería Geofísica	2010	Terminé créditos	Sí	C, Matlab, Fortran	2	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	No	Sí	Sí	Sí
120	Ingeniería Geofísica	2015	Décimo	No					Sí	Sí	Sí
121	Ingeniería Geofísica	2013	Terminé créditos	Sí	Matlab	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí

122	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab, Python	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
123	Ingeniería Geofísica	2017	Octavo	No					Sí	Sí	No
124	Ingeniería de Minas y Metalurgia	2016	Décimo	No					Sí	No	Sí
125	Ingeniería de Minas y Metalurgia	2018	Sexto	No					Sí	Sí	Sí
126	Ingeniería de Minas y Metalurgia	2014	Noveno	Sí	Python	3	Cursos fuera de la UNAM	No	Sí	Sí	Sí
127	Ingeniería de Minas y Metalurgia	2016	Décimo	Sí	C	3	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
128	Ingeniería de Minas y Metalurgia	2015	Décimo	Sí	C	3	Principal (Materias de la Carrera)	No	Sí	No	Sí
129	Ingeniería de Minas y Metalurgia	2016	Décimo	No					Sí	Sí	Sí
130	Ingeniería de Minas y Metalurgia	2016	Noveno	Sí	Matlab	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
131	Ingeniería de Minas y Metalurgia	2018	Sexto	Sí	C, Matlab	2	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
132	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	No					Sí	Sí	Sí
133	Ingeniería de Minas y Metalurgia	2016	Octavo	No					Sí	Sí	Sí

134	Ingeniería de Minas y Metalurgia	2016	Décimo	No					Sí	Sí	Sí
135	Ingeniería de Minas y Metalurgia	2018	Séptimo	Sí	Matlab, Python	3	Intersemestrales (cursos impartidos por las sociedades, UNICA, PROTECO, ...)	Sí	Sí	Sí	Sí
136	Ingeniería de Minas y Metalurgia	2017	Terminé créditos	No					Sí	No	Sí
137	Ingeniería de Minas y Metalurgia	2010	Terminé créditos	Sí	Matlab	4	Cursos fuera de la UNAM	No	No	Sí	No
138	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	Sí	C	1	Cursos fuera de la UNAM	No	Sí	Sí	Sí
139	Ingeniería de Minas y Metalurgia	2014	Terminé créditos	Sí	C, Matlab	2	Anexo de Ingeniería (Ciencias Básicas)	No	Sí	Sí	Sí
140	Ingeniería de Minas y Metalurgia	2016	Décimo	No					Sí	Sí	Sí
141	Ingeniería de Minas y Metalurgia	2013	Décimo	No					No	No	Sí
142	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	No					Sí	Sí	Sí
143	Ingeniería de Minas y Metalurgia	2017	Sexto	No					Sí	No	Sí
144	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	No					Sí	Sí	Sí

145	Ingeniería de Minas y Metalurgia	2017	Sexto	Sí	C, Matlab, Python	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
146	Ingeniería de Minas y Metalurgia	2017	Sexto	Sí	Matlab	3	Principal (Materias de la Carrera)	No	Sí	No	Sí
147	Ingeniería de Minas y Metalurgia	2016	Noveno	No					Sí	Sí	Sí
148	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	No					No	Sí	No
149	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	No					Sí	Sí	Sí
150	Ingeniería de Minas y Metalurgia	2018	Sexto	No					Sí	Sí	Sí
151	Ingeniería de Minas y Metalurgia	2014	Terminé créditos	No					Sí	Sí	Sí
152	Ingeniería de Minas y Metalurgia	2019	Terminé créditos	No					Sí	Sí	Sí
153	Ingeniería de Minas y Metalurgia	2016	Décimo	Sí	C, Matlab, Python	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
154	Ingeniería de Minas y Metalurgia	2019	Terminé créditos	No					Sí	No	Sí
155	Ingeniería de Minas y Metalurgia	2015	Terminé créditos	No					Sí	Sí	Sí
156	Ingeniería Geológica	2016	Octavo	Sí	C, Python	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí

157	Ingeniería de Minas y Metalurgia	2010	Terminé créditos	Sí	C, Python	5	Cursos fuera de la UNAM	No	Sí	Sí	Sí
158	Ingeniería Geofísica	2010	Terminé créditos	Sí	C, Matlab, Fortran	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
159	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab, Fortran	4	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
160	Ingeniería Geofísica	2016	Décimo	Sí	Matlab, Fortran	3	Principal (Materias de la Carrera)	Sí	Sí	Sí	Sí
161	Ingeniería Geofísica	2015	Terminé créditos	Sí	Matlab, Python	1	Principal (Materias de la Carrera)	Sí	Sí	Sí	No
162	Ingeniería Geológica	2018	Terminé créditos	No					Sí	No	Sí
163	Ingeniería Geológica	2018	Terminé créditos	No					Sí	No	Sí
164	Ingeniería de Minas y Metalurgia	2012	Terminé créditos	Sí	C, Matlab	2	Anexo de Ingeniería (Ciencias Básicas)	Sí	Sí	Sí	Sí
165	Ingeniería de Minas y Metalurgia	2015	Noveno	Sí	C, Matlab	3	Principal (Materias de la Carrera)	No	Sí	Sí	Sí
166	Ingeniería Geofísica	2014	Terminé créditos	Sí	Matlab, R	4	Cursos fuera de la UNAM	Sí	Sí	Sí	Sí
167	Ingeniería Geofísica	2014	Décimo	No					Sí	Sí	Sí
168	Ingeniería de Minas y Metalurgia	2016	Octavo	No					No	Sí	Sí
169	Ingeniería Geofísica	2018	Sexto	No					Sí	Sí	Sí
170	Ingeniería Geofísica	2016	Octavo	Sí	Python	3	Cursos fuera de la UNAM	No	Sí	Sí	Sí

Tabla A2. Resultados de la encuesta realizada a alumnos

DOCENTES

	Departamento al que perteneces	Rango de edad (años)	¿Qué tan útil consideras programar en la vida laboral? 1: poco útil, 5: muy útil	¿Has programado en tu vida profesional?	¿En qué lenguaje programaste?	¿Qué nivel de dificultad tiene Python comparado con otros lenguajes (C, Matlab, Fortran, ...)?	¿Consideras que Python es una herramienta útil en el campo laboral? 1: poco útil, 5: muy útil	¿Ha programado para resolver algún problema de geociencias?	¿En la materia que impartes le pides a tus alumnos que programen?
1	Geología	Mayor a 50	5	No					No
2	Geología	41-50	1	No					No
3	Geología	Mayor a 50	4	Sí	R	No sé	4	Sí	No
4	Geología	Mayor a 50	4	Sí	C++, Fortan, Basic y Pascal (Delphi)	No sé	5	Sí	No
5	Geología	31-40	3	No					No
6	Geología	Mayor a 50	5	Sí	Fortran	Tienen el mismo nivel de dificultad	5	Sí	No
7	Geología	Mayor a 50	4	No					No
8	Geología	41-50	5	Sí	Fortran	No sé	3	Sí	Sí
9	Geología	Mayor a 50	3	Sí	FORTRAN	No sé	4	No	No
10	Geología	31-40	4	No					No
11	Geología, Geofísica	Mayor a 50	4	Sí	Varios (C++, VB, C#, Fortran)	Tienen el mismo nivel de dificultad	3	Sí	Sí
12	Geología	Mayor a 50	5	Sí	Fortran	Es más fácil	5	Sí	No
13	Geología	31-40	5	Sí	Python	Es más fácil	5	Sí	No
14	Geología, Geofísica, Minas y Metalurgia	Mayor a 50	4	No					No
15	Geología	41-50	5	Sí	C	Tienen el mismo nivel de dificultad	3	Sí	No
16	Geología	31-40	5	Sí	Matlab	No sé	5	No	No
17	Geofísica	31-40	4	Sí	Matlab	No sé	4	Sí	No

18	Minas y Metalurgia	41-50	3	Sí	R, Arduino	No sé	3	No	No
19	Geología	21-30	3	No					No
20	Geología	Mayor a 50	4	Sí	Fortran	No sé	4	Sí	No
21	Geología	41-50	5	Sí	Python	Es más fácil	5	Sí	Sí
22	Geología	Mayor a 50	5	Sí	Fortran	Es más fácil	5	No	No
23	Geología	21-30	3	No					No
24	Geología	21-30	4	No					No
25	Geología	21-30	4	Sí	Python	Es más fácil	4	Sí	No
26	Geología	Mayor a 50	2	Sí	Fortran	No sé	3	Sí	No
27	Geología	21-30	4	Sí	C	Tienen el mismo nivel de dificultad	4	Sí	No
28	Geología	Mayor a 50	3	No					No
29	Geología	Mayor a 50	5	Sí	Fortran	No sé	3	No	No
30	Geología	41-50	2	No					No

Tabla A3. Resultados de la encuesta realizada a docentes

APÉNDICE B

TUTORIAL DE PARTICIÓN DEL DISCO DURO

Lo primero que necesitas para realizar la partición en el disco duro es espacio (al menos 50 GB)

1. Dirígete a la barra de herramientas, da clic derecho y da clic en "Administración de equipos".

Figura B1. Escritorio y barra de herramientas

2. En "Almacenamiento", da clic en "Administración de discos"

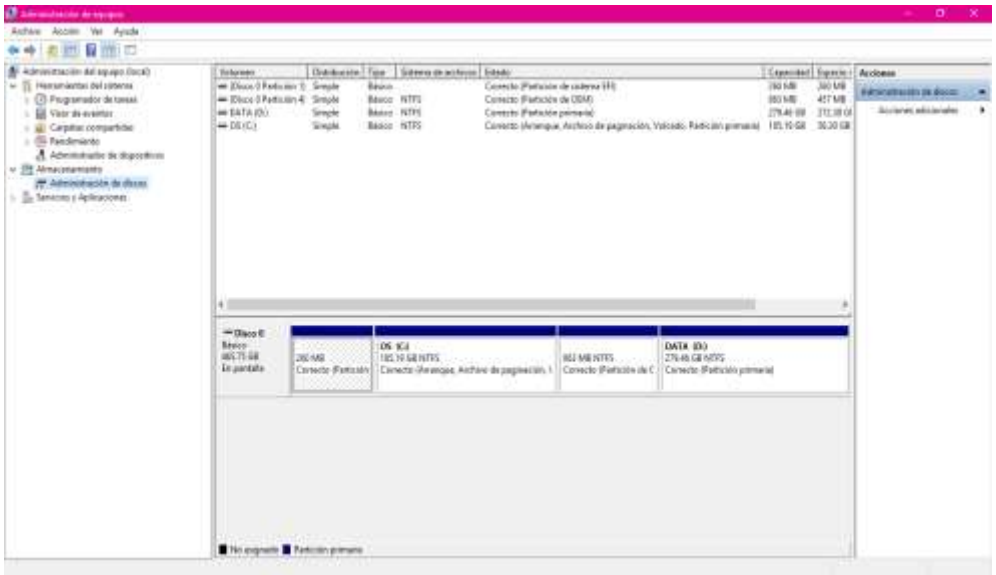


Figura B2. Administración de equipos

3. Selecciona el disco a particionar (en mi caso: Data (D:)), clic derecho y "Reducir volumen"

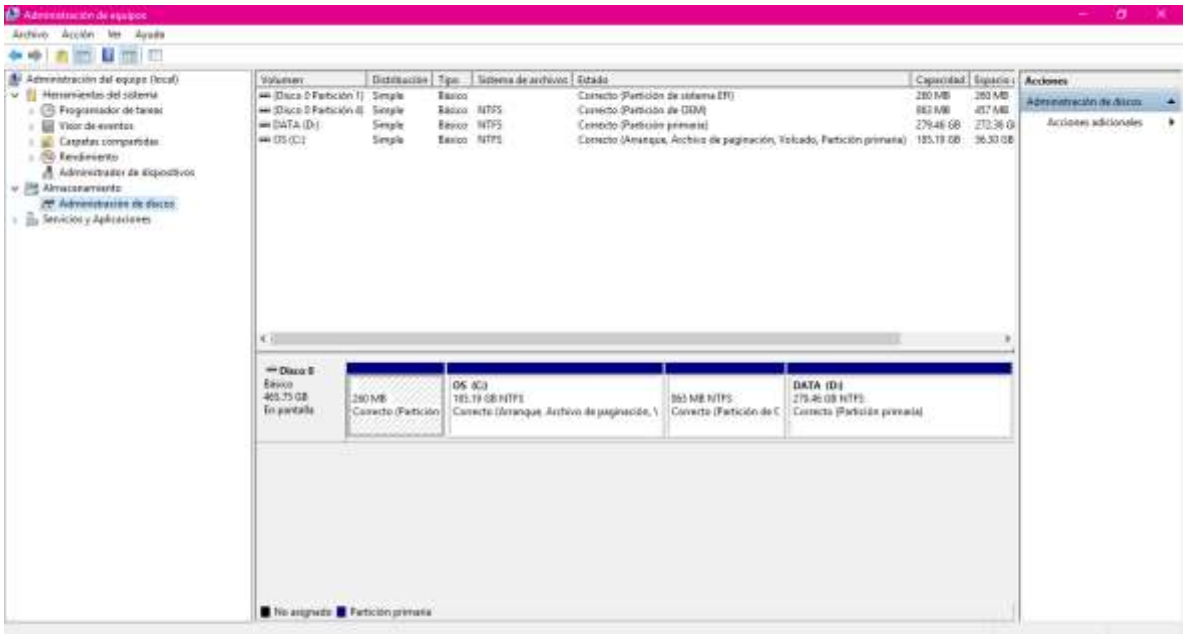


Figura B3. Selección de disco a particionar

4. Cambia el valor en "Tamaño del espacio que desea reducir" a 71680 (que es equivalente a 70 GB en MB). Arriba mencioné que necesitas como mínimo 50 GB, yo haré la partición a 70 GB. En caso de que requieras más espacio, puedes incrementar el valor, sólo recuerda convertir los GB en MB. Por último, selecciona "Reducir".

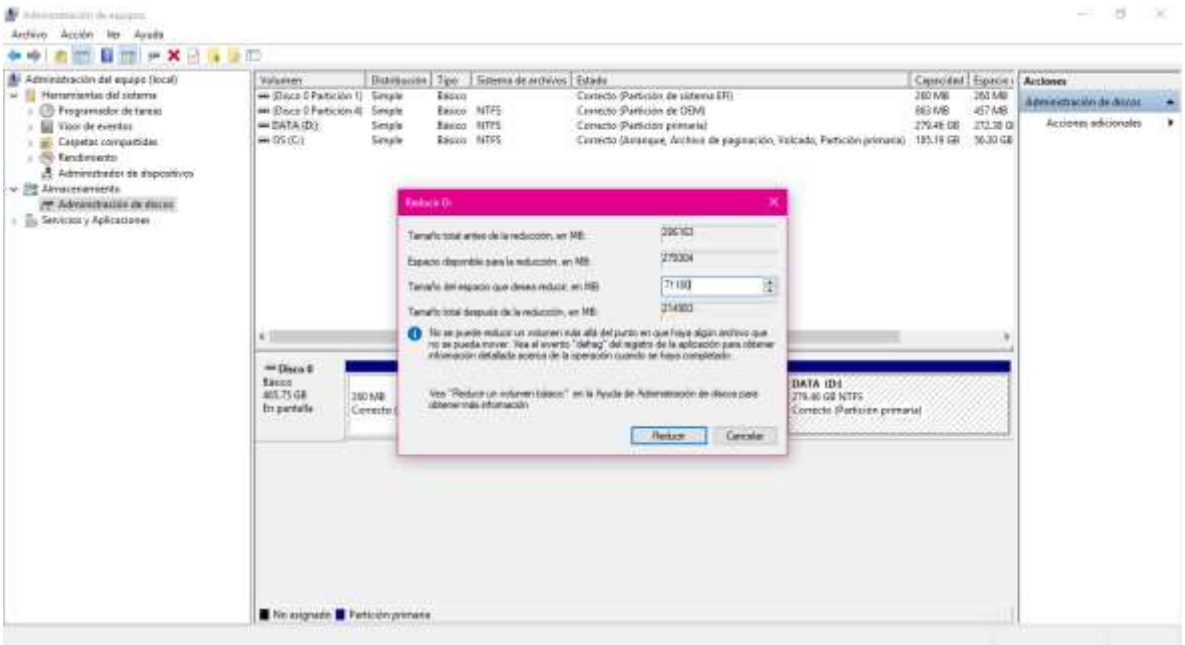


Figura B4. Selección de tamaño del espacio en disco a reducir

5. La nueva partición se muestra de color negro, da clic derecho y selecciona “Nuevo volumen simple”. Después da clic en “Asistente para nuevo volumen simple” y clic en “Siguiente”.

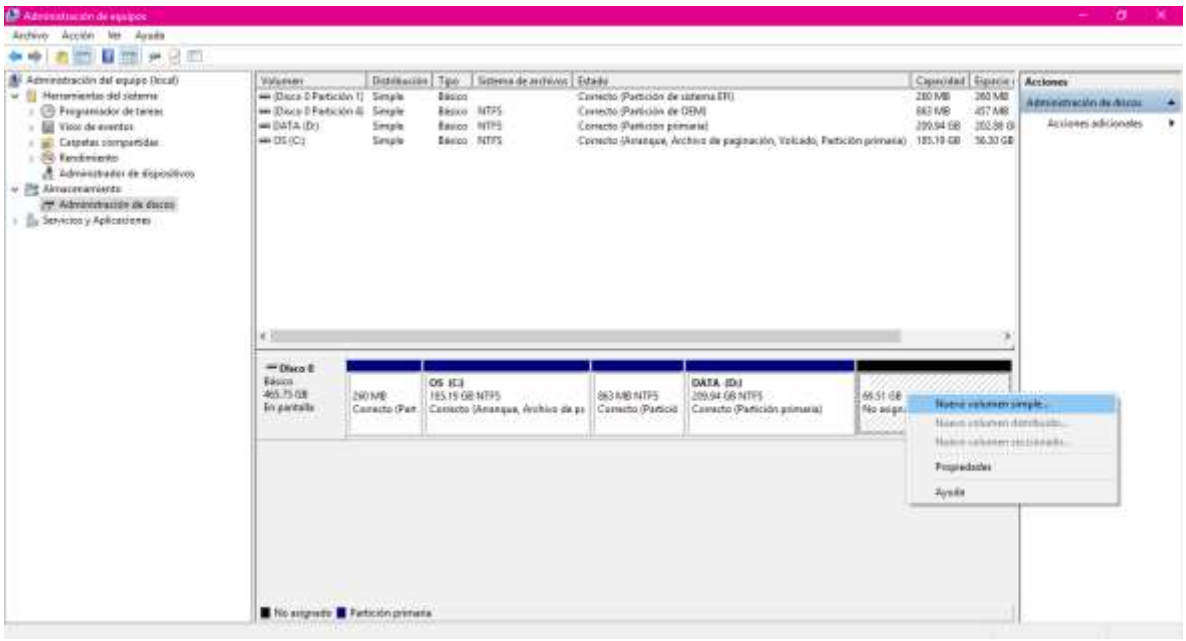


Figura B5. Generación de nuevo volumen simple

6. En “Especificar el tamaño del volumen” no mover nada y dar clic en “Siguiente”.

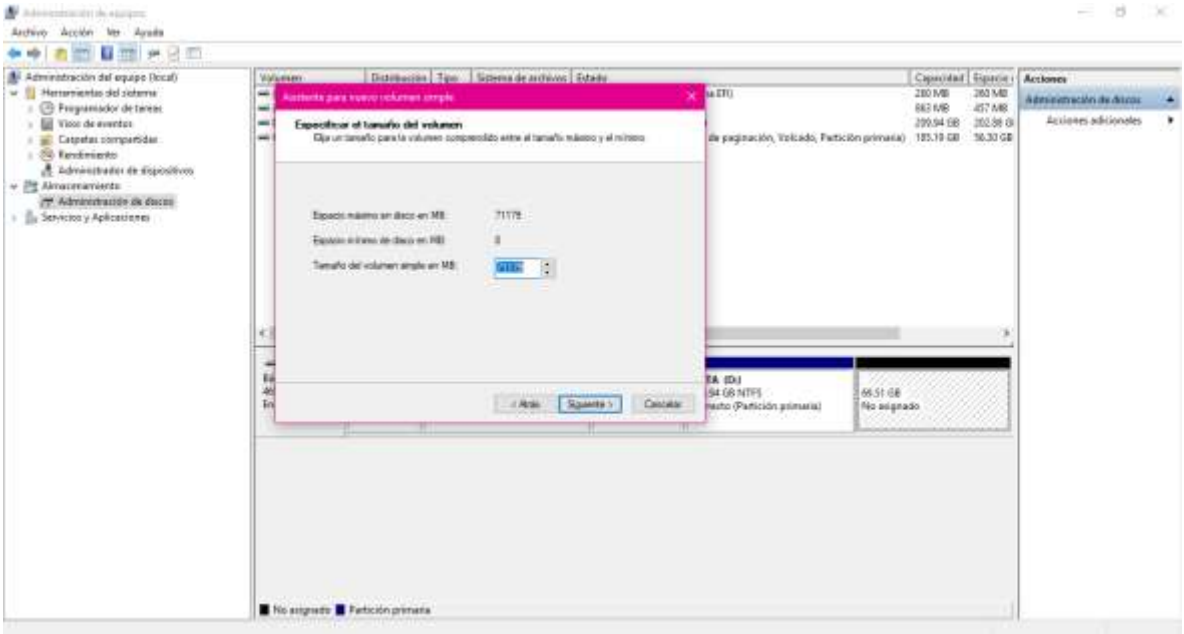


Figura B6. Especificación de tamaño del volumen

7. Asigna una letra a la nueva partición (opcional) y da clic en “Siguiente”.

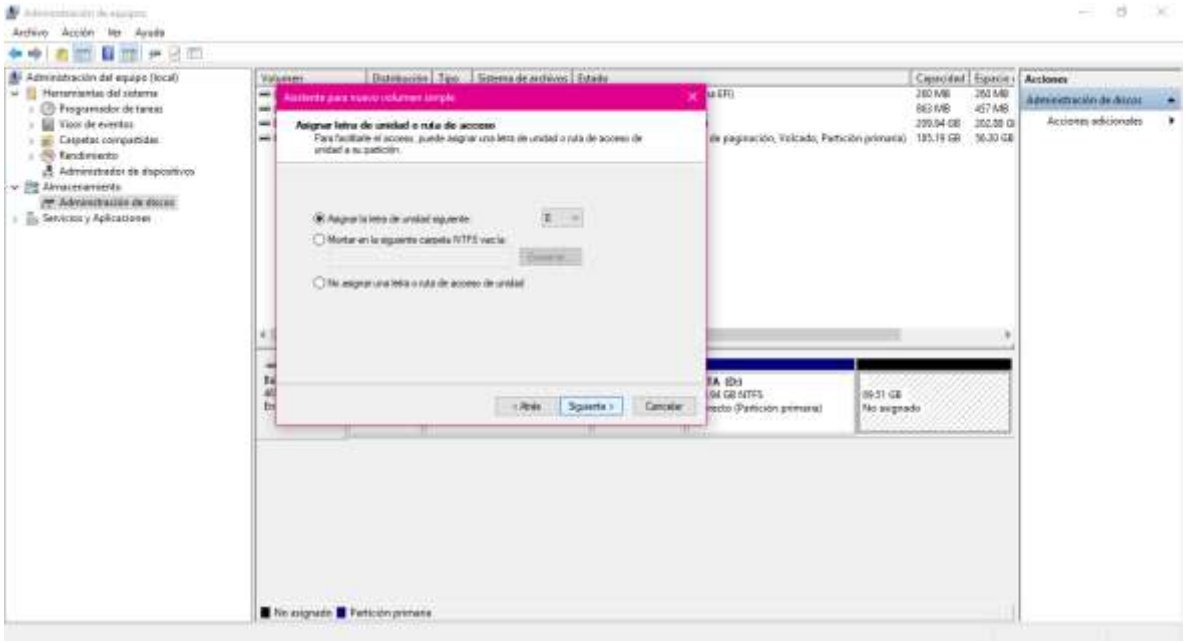


Figura B7. Asignación de letra a unidad o ruta de acceso de unidad de partición

8. Asigna un nombre (opcional) y da clic en “Siguiente”.

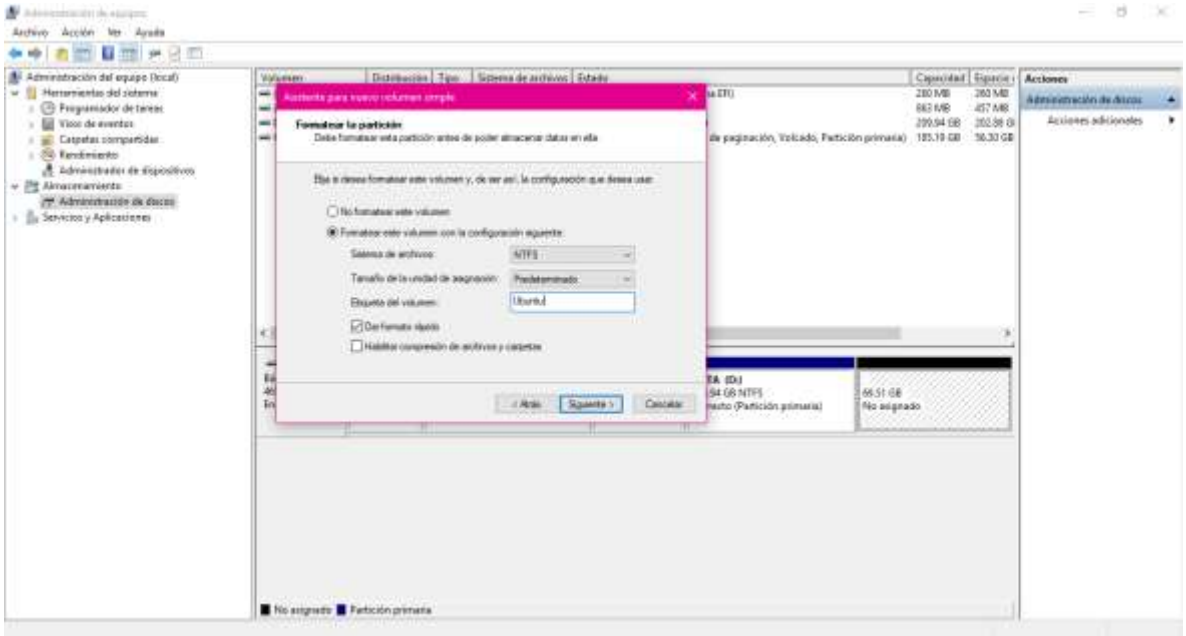


Figura B8. Asignación de letra a unidad o ruta de acceso de unidad de partición

9. Por último, da clic en “Finalizar”.

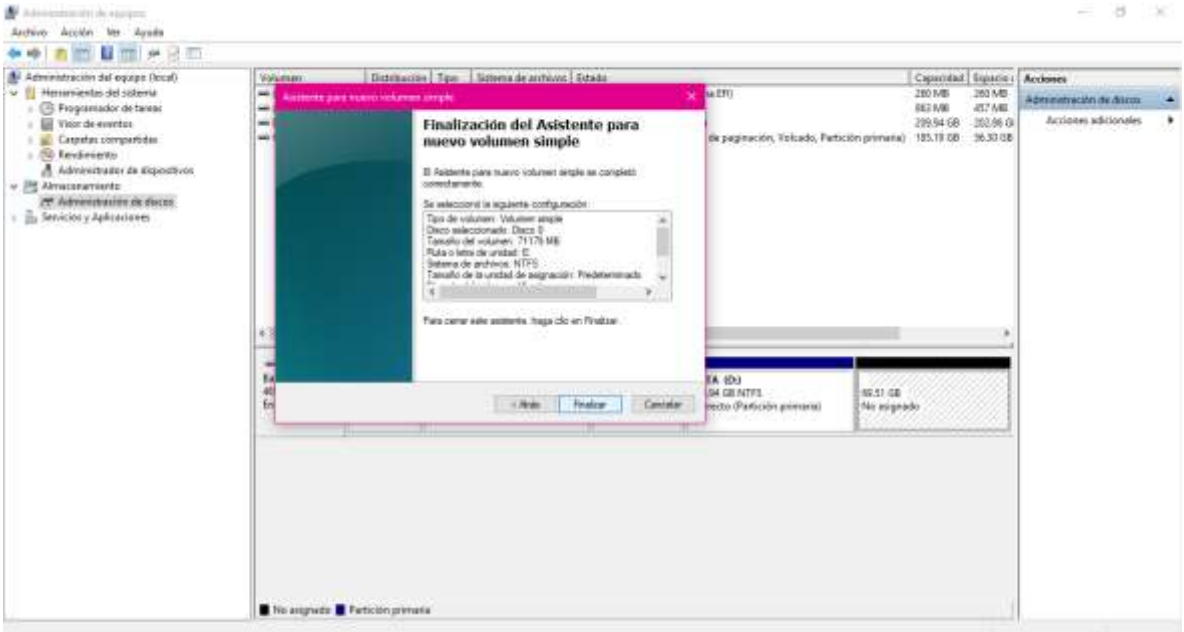


Figura B9. Finalización del asistente para nuevo volumen simple

10. Ya cuentas con una nueva partición en el disco duro, si quieres corroborarlo, dirígete a “Este equipo” y ahí aparecerá la nueva partición.

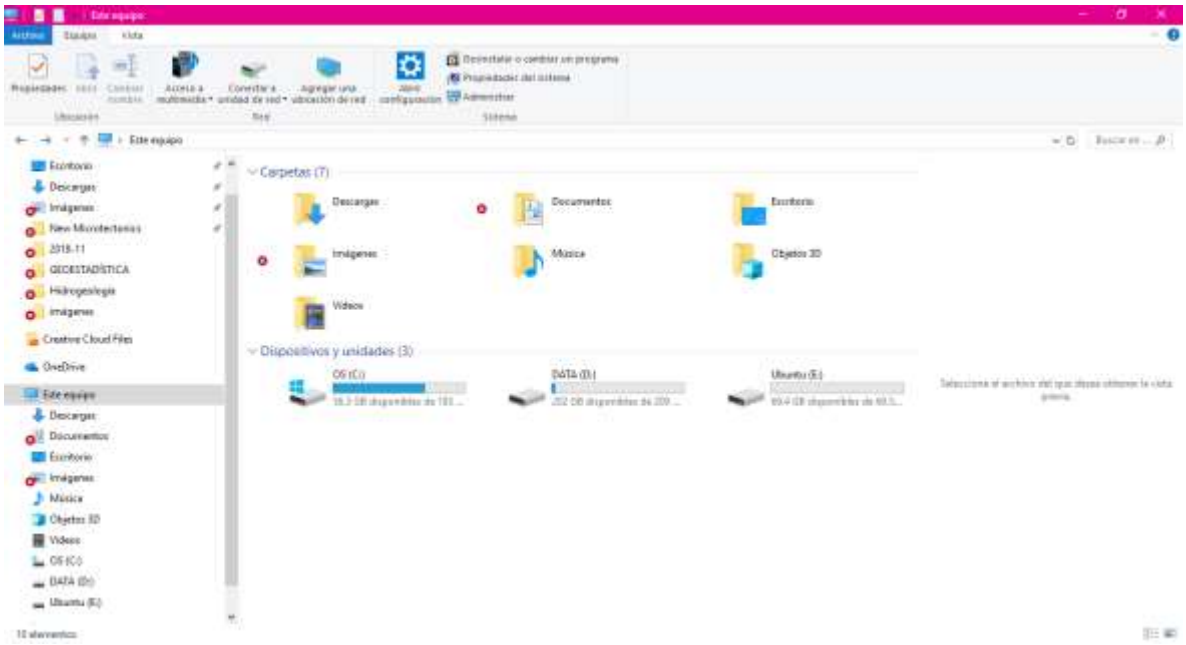


Figura B10. Comprobación de partición del disco duro

11. También lo puedes revisar en “Administración de equipos”

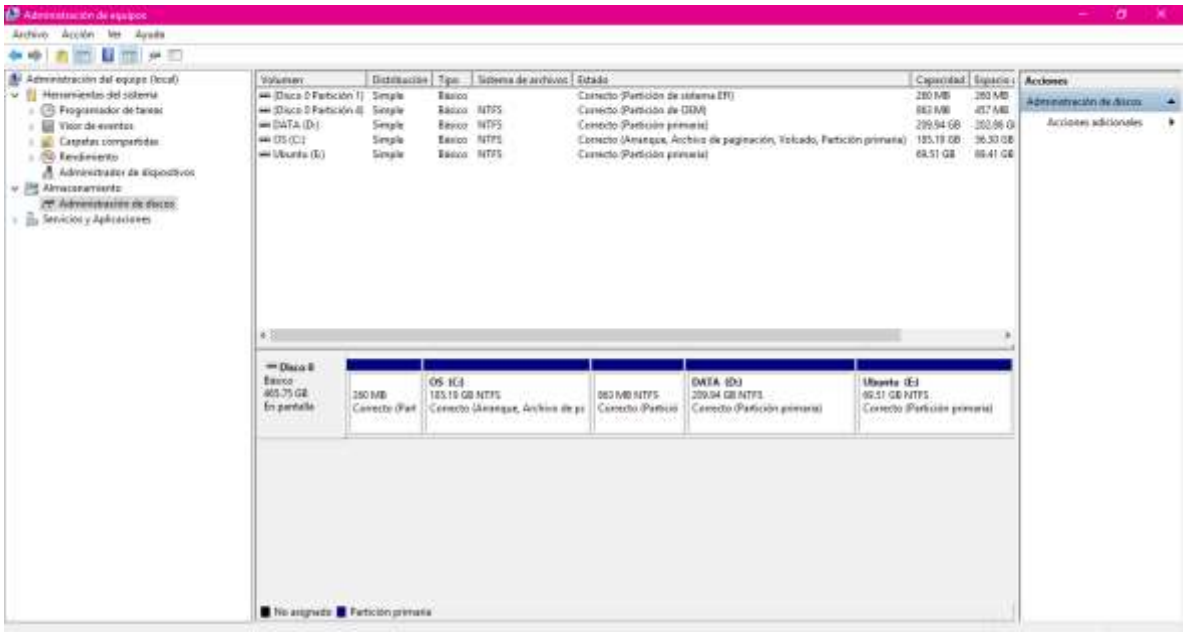


Figura B11. Comprobación de partición del disco duro desde "Administración de equipos"

APÉNDICE C

TUTORIAL DE DESCARGA DE UBUNTU

1. Dirígete a https://ubuntu.com/download/desktop y da clic en "Descargar". Se guardará un archivo con extensión .iso, es importante que sepas en que carpeta se descargará el archivo.

Figura C1. Descarga de Ubuntu y requerimientos del sistema

APÉNDICE D

TUTORIAL DE DESCARGA DE RUFUS

Necesitas una memoria USB (al menos 8 GB) que no tenga ningún archivo almacenado.

1. Dirígete https://rufus.ie/ y descarga "Rufus 3.1 Portátil"

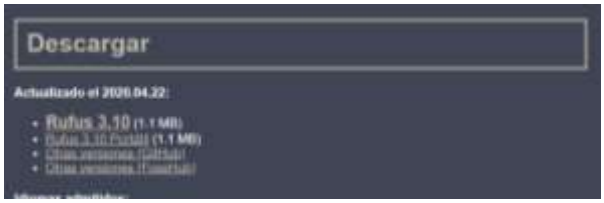


Figura D1. Descarga de Rufus Portátil

2. En "Dispositivo", escoge tu memoria USB.

Figura D2. Propiedades de la unidad

3. En elección de arranque, elige "Disco o imagen ISO", en la opción Seleccionar dar clic y seleccionar el archivo ISO que descargamos previamente.

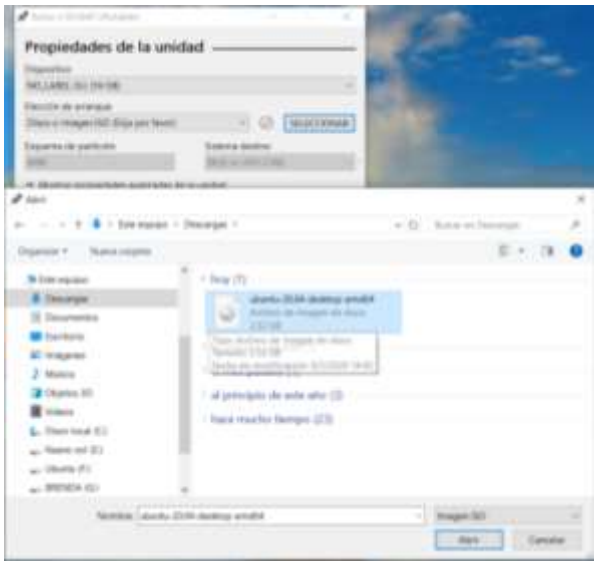


Figura D3. Selección de arranque

4. Da clic en “Empezar”

Figura D4. Arranque

5. Ahora se encuentra en tu memoria USB. Puedes corroborarlo en “Este equipo”

APÉNDICE E

TUTORIAL DE INSTALACIÓN DE UBUNTU

1. Apaga tu equipo. Enseguida inserta la memoria USB.
2. Enciende tu computadora y accede a la *BIOS*. En este enlace puedes ver en la columna *BIOS key* con que tecla acceder dependiendo del fabricante de tu equipo: https://www.disk-image.com/faq-bootmenu.htm

3. Dirígete al menú “Boot”. En “Boot Option Properties”, cambia la opción 1 por la opción 2. Esto para correr Ubuntu desde la USB.
4. *Save* & *exit* (F10)
5. Apaga y enciende tu computadora.
6. Selecciona el idioma y da clic en “Continuar”

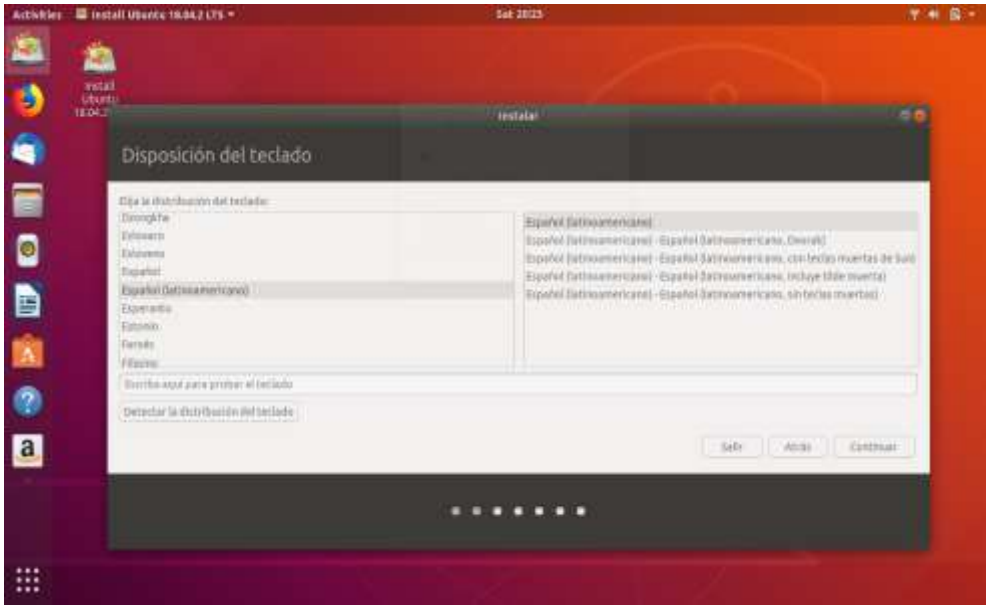


Figura E1. Elección de distribución del teclado

7. Selecciona “instalación normal” y da clic en “Continuar”

Figura E2. Actualizaciones y otro software

8. En “tipo de instalación”, seleccionar “Instalar Ubuntu junto a Windows Boot Manager” y “Continuar”

Figura E3. Selección del tipo de instalación

9. Selecciona la primera opción y da clic en “Instalar ahora”

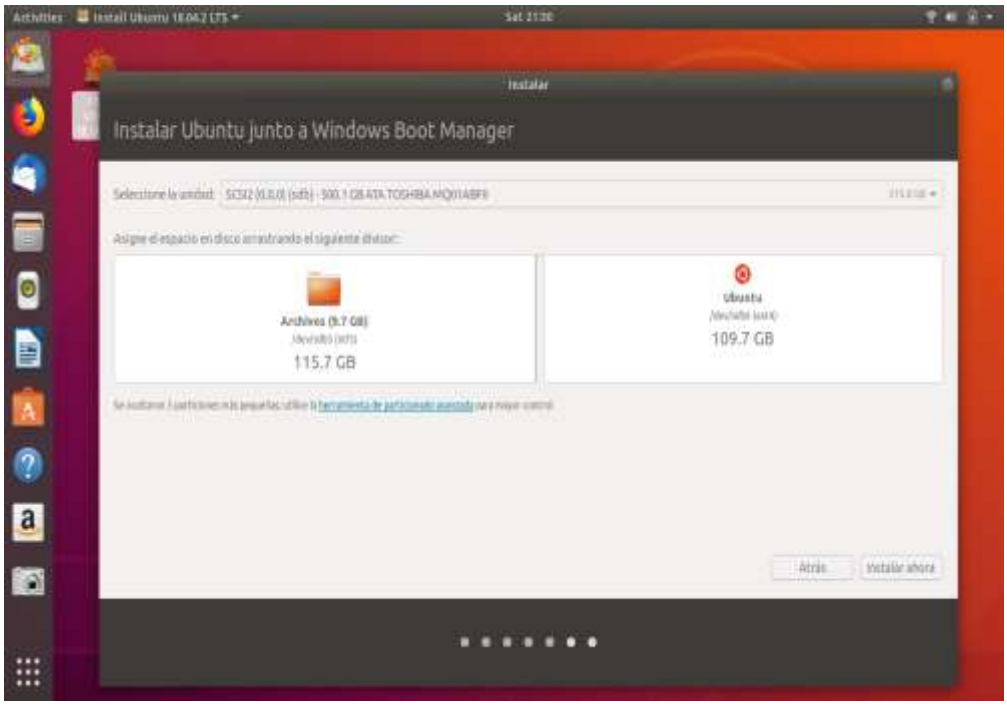


Figura E4. Instalación de Ubuntu junto a Windows Boot Manager

10. Da clic en “Continuar”.

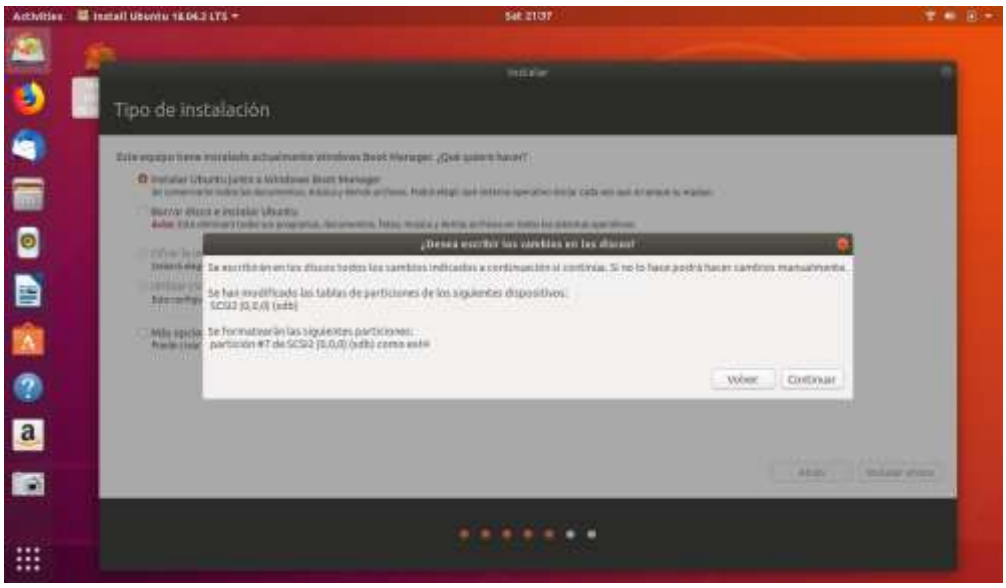


Figura E5. Confirmación de escritura de cambios en los discos

11. Ahora completa con tus datos y establece una contraseña.

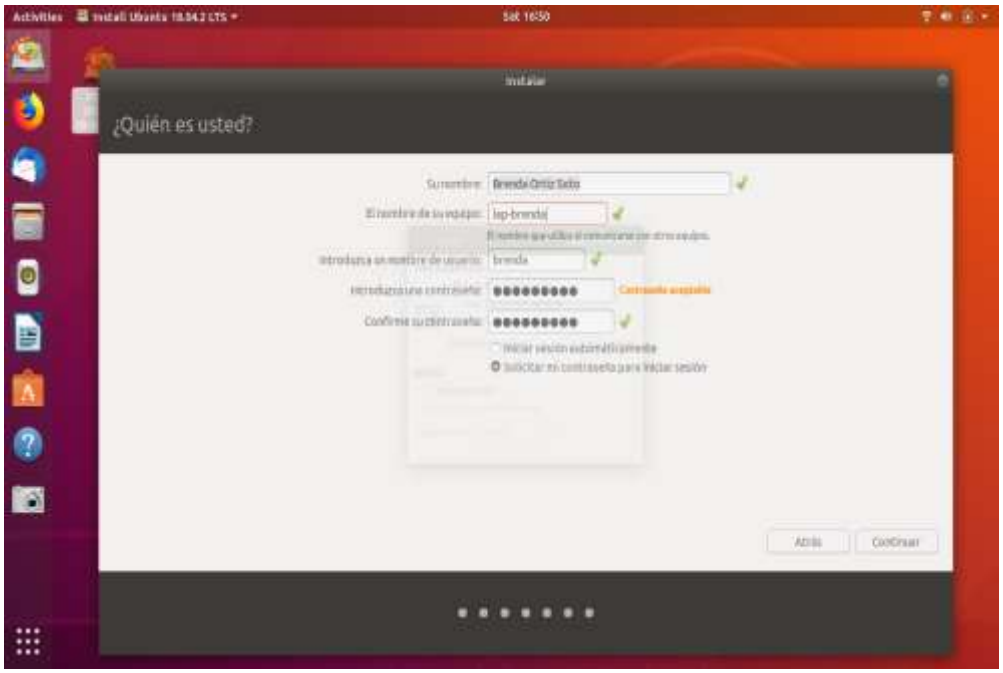


Figura E5. Asignación de usuario y contraseña

12. Espera a que termine la instalación.

Figura E6. Instalación en curso y bienvenida a Ubuntu

APÉNDICE F

TUTORIAL DE INSTALACIÓN DE ANACONDA

Para fines prácticos, la instalación de Python que vamos a utilizar será a través de Anaconda (https://www.anaconda.com/). La razón es simple: el equipo de personas detrás de Anaconda ya sufrió, e incluso tal vez derramó algunas lágrimas, para poder asegurar la compatibilidad de Python de diferentes paquetes en diferentes sistemas operativos. Al instalar Anaconda, se instala todo el software y librerías cercanamente relacionado a Python, de manera que, por ejemplo, se tiene acceso inmediato a Jupyter, SciPy, NumPy, Spyder, Pandas y Matplotlib.

1. Dirígete a https://www.anaconda.com/products/individual y descarga la versión más reciente (3.7) dependiendo tu sistema operativo.

Figura F1. Instaladores de Anaconda

2. Guarda el archivo en alguna carpeta que tú asignes, por ejemplo, en "Descargas" y copia el nombre del archivo, en mi caso es: Anaconda3-2020.02-Linux-x86_64.sh
3. Abre la terminal e ingresa el siguiente comando:

   sudo bash ~/nombre_de_la_carpeta_donde_esta_tu_archivo/el_archivo_que_descargaste

   En mi caso:

   sudo bash ~/Descargas/Anaconda3-2020.02-Linux-x86_64.sh
4. El instalador requiere la contraseña del equipo por lo que tienes que ingresarla. También te solicita que revises el acuerdo de licencia, da "ENTER" para ver los términos de la licencia y desplázate hasta el final, aquí es donde confirmas que aceptas los términos, escribiendo *yes*.
5. Aquí la terminal te pregunta donde se va instalar "Anaconda", para lo cual mantendremos la ubicación predeterminada del equipo presionando ENTER y aceptamos escribiendo "yes".
6. Ahora ya tienes instalado Anaconda. Para poder visualizarlo, ingresa el comando:

   source ~/-bashrc
7. Para comprobar que la instalación fue exitosa, ingresa en la terminal:

   anaconda-navigator.
8. De ahora en adelante, el último comando mencionado será utilizado siempre que queramos ingresar a programar utilizando Jupyter Notebooks o Spyder.

Podemos corroborar que la instalación de Python en nuestro equipo sea correcta y actualizada de la siguiente manera:

1. Ir a la terminal, para esto puedes acceder con "ctrl" + "alt" + t, o desde las aplicaciones abrir "Terminal"
2. Escribir en la terminal lo siguientes comandos:

   sudo apt-get update

   sudo apt-get upgrade
3. Para comprobar cuál versión de Python tenemos instalada, ingresamos el comando:

   python –V

APÉNDICE G

ACCESO A JUPYTER NOTEBOOK

1. Ahora accederás a Jupyter desde Ubuntu, presiona las teclas “ctrl” + “alt” + t, y escribe en la terminal el comando:

   anaconda-navigator

Te aparecerá la siguiente ventana:

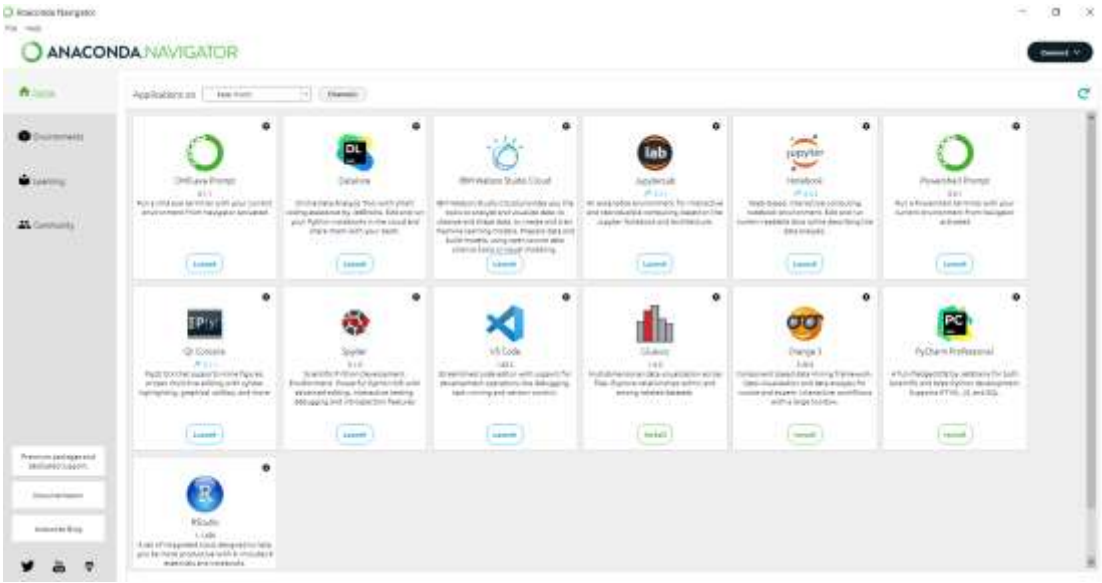


Figura F2. Anaconda navigator

2. Procede a seleccionar Jupyter Notebook y da clic en *launch*. Aparecerá la siguiente ventana:

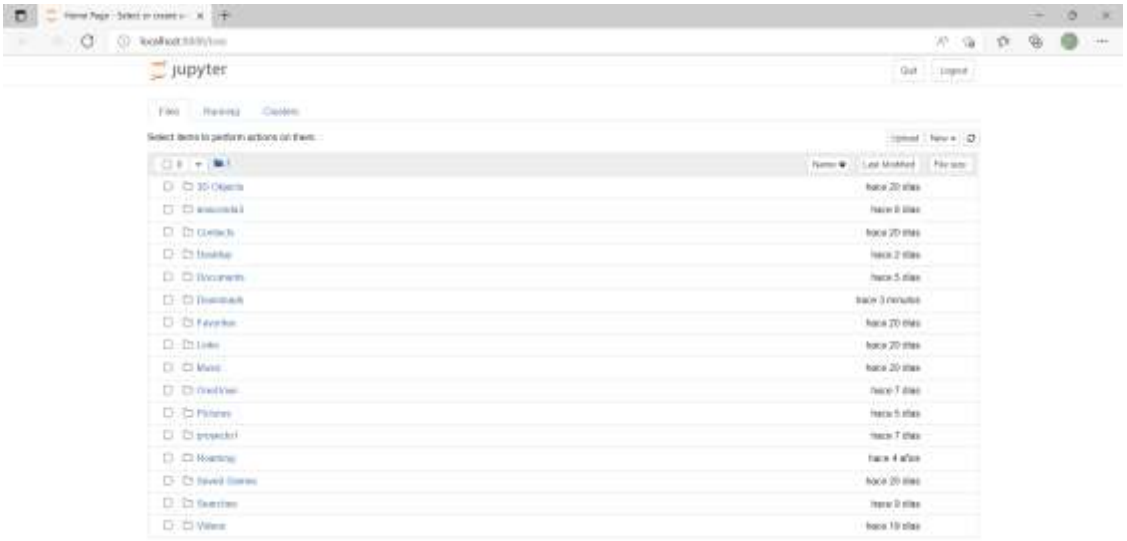


Figura F3. *Home page*

3. Da clic en *New* y selecciona Python 3 (ipykernel):

¡Listo! Ya has configurado tu equipo, ahora puedes ir a los ejercicios propuestos.

*********Este procedimiento fue realizado en mayo del 2020 y es válido al día 9 de mayo del 2020. Debido a actualizaciones de las versiones de Ubuntu, Python, etc., a futuro pueden existir pequeñas variaciones en el procedimiento. ********

APÉNDICE H

EJERCICIOS PROPUESTOS

EJERCICIOS PROPUESTOS

0. PRIMEROS PASOS EN JUPYTER NOTEBOOK

1. LECTURA DE DATOS DESDE UNA TABLA

2. CÁLCULO DE MEDIDAS DE TENDENCIA CENTRAL

3. COVARIANZA Y CORRELACIÓN DE DATOS

4. ESTANDARIZACIÓN DE DATOS Y SEMIVARIOGRAMA EN PYTHON

5. GRAFICANDO PERFIL GEOLÓGICO

6. RED DE SCHMIDT Y RED DE WULFF

7. MANIPULACIÓN Y VISUALIZACIÓN DE UNA NUBE DE DATOS EN PYTHON

8. SEGMENTACIÓN DE NUBES DE PUNTOS BASADA EN COLOR Y *K-MEANS* EN PYTHON

9. BÚSQUEDA DE PLANOS DE DISCONTINUIDAD A PARTIR DE UNA NUBE DE PUNTOS

EJERCICIO 0. PRIMEROS PASOS EN JUPYTER NOTEBOOK

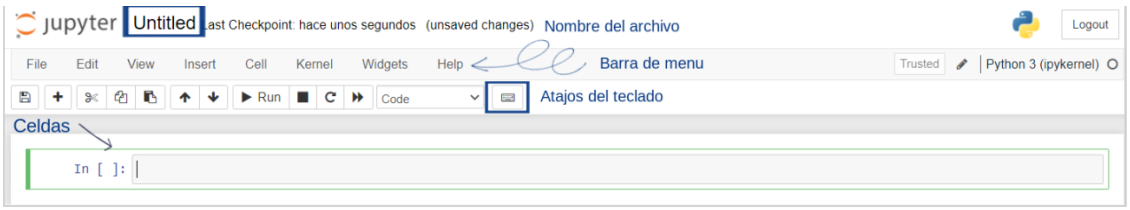
March 31, 2022

1 EJERCICIO 0. PRIMEROS PASOS EN JUPYTER NOTEBOOK

Autora: Brenda Ortiz Soto

En este ejercicio te familiarizarás con el entorno de trabajo de Jupyter e imprimirás tu primera línea de código.

Cuando abras un nuevo *notebook*, visualizarás el área de trabajo:



`Fig 1. Área de trabajo de Jupyter Notebook`

En la parte superior aparece el nombre del Jupyter Notebook, por default se llama *Untitled* y corresponde con el nombre del archivo .ipynb

Para cambiar el título sólo da clic y renómbralo.

En la barra de menu, encuentras diferentes opciones para gestionar el notebook (abrir, guardar como, hacer una copia, etc), las celdas (insertar, correr celda, etc) y el kernel (interrumpir, reiniciar, reconectar, etc).

1.1 Estructura de un notebook

El notebook consiste de una secuencia de celdas (*cells*) y utilizaremos dos tipos de celdas:

- Celdas de código (*Code cells*): te permiten escribir y editar el código.
- Celdas *Markdown* (*Markdown cells*): te permiten escribir texto enriquecido utilizando lenguaje de marcado (*Markdown language*). Este tipo de celdas mejorarán la presentación de tus notebooks y será más fácil para los usuarios poder entender los notebooks.

Para insertar una celda, da clic en el menu en donde aparece el icono de + o también puedes dar clic en *Insert* y luego *Insert cell below*.

Cada que agregamos una celda, comienza siendo una celda de código pero se puede cambiar usando el menú despegable de la barra de herramientas, dicho menu lo encuentras del lado izquierdo del icono de atajos de teclado.

Puedes ejecutar una celda usando las teclas *Shift + Enter*, usando las teclas *Ctrl + Enter* o seleccionando la celda a ejecutar y luego da clic en *Run*. Como puedes darte cuenta, puedes hacerlo de distintas maneras, escoge la que sea más cómoda para ti.

[19]:
```
print("Esto es una celda de código")
```

```
Esto es una celda de código
```

¿Cómo saber si la celda ya se ejecutó? Observa el extremo izquierdo de la celda anterior.

- Si solamente aparece In [ ], quiere decir que aún no se ha ejecutado
- Si lo que aparece es In [*], entonces está en proceso de ejecución
- Si aparece un número entre los corchetes significa que la celda ya se ejecutó

También puedes ir documentando tu notebook en las celdas de código, esto es una buena práctica cuando quieres explicarle a alguien más que está haciendo tu código. Para comentar el código sólo agrega un signo de "#".

[20]:
```
#descomenta la siguiente línea e imprime tu primera línea de código
#print("¡Hola mundo!")
```

Puedes acceder a los atajos del teclado desde el icono de teclado en la barra de menu.

Para guardar tu noteebook, puedes hacerlo con el comando *Crtl +S* o en da clic en *File* y luego en *Save & Checkpoint* y para descargar tu notebook das clic en *File* y luego en *Download as*, aquí encontrarás una lista de diferentes extensiones para descarga.

Para salir del notebook, asegúrate de guardarlo como se indicó antes y dirígete a *File* y luego da clic en *Close and Halt*.

¡Ahora es tu turno de explorar la barra de menu!

Referencias:

1. The Jupyter Notebook — Jupyter Notebook 7.0.0a2 documentation. (n.d.). Readthedocs.Io. Recuperado el 31 de marzo de 2022, de https://jupyter-notebook.readthedocs.io/en/latest/notebook.html

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 1. LECTURA DE DATOS DESDE UNA TABLA

March 31, 2022

1 EJERCICIO 1. LECTURA DE DATOS DESDE UNA TABLA

Adaptado por: Brenda Ortiz Soto

Versión original: Daría Chemkaeva

En este ejercicio se aborda como leer datos desde un archivo .csv y como manipular dichos datos. Para obtener el *dataset* puedes dar clic aquí. El *dataset* corresponde con una lista de más de 5000 datos correspondientes a un listado de todos los minerales registrados en el "IMA".

Primero, requieres descargar el archivo que contiene la información (en este caso, entra al enlace antes mencionado), esté será un archivo separado por comas, es decir, con extensión .csv; es importante que tu archivo se encuentre en la misma carpeta donde abriste este *notebook*. Si tus datos a importar se encuentran en una hoja de cálculo, bastará con que guardes tu archivo con extensión csv.

Una vez que tengas la información descargada, procedemos a importar las liberías necesarias para poder manipular los datos, realizar gráficas, etc. OJO: importar las librerías será de ahora en adelante lo primero que aparezca en el código.

1.1 Importación de librerías y lectura de datos

[2]:
```
import pandas as pd   #importamos la librería pandas y la "abreviamos" como "pd"
import seaborn as sns    # importamos la librería seaborn
sns.set()               #alias para set_theme(), aplica tema default
import matplotlib.pyplot as plt  # importamos la librería
%matplotlib inline
#función mágica, garantiza que Jupyther Notebook muestre los gráficos␣
 ↪realizados con matplotlib

#Las librerías se abrevian para que sea más fácil llamarlas en el código
```

Ahora leeremos un archivo separado con comas, utilizando la sintaxis: nombre=pd.read_csv('nombre del archivo.extensión')

[3]:
```
datos_e1 = pd.read_csv('e1_datos.csv') #los datos se almacenan en la variable␣
 ↪"datos_e1"
```

1.2 Conociendo el dataset

Los datos ya están importados y para poder previsualizar la tabla escribimos: datos_e1.head()

[4]:
```
datos_e1.head() #Por default apareceran 5 filas
```

[4]:
```
      Mineral Name                    RRUFF Chemistry (plain)  \
0        Abellaite                          NaPb2+2(CO3)2(OH)
1       Abelsonite                               Ni2+C31H32N4
2  Abenakiite-(Ce)  Na26Ce3+6(SiO3)6(P5+O4)6(C4+O3)6(S4+O2)O
3    Abernathyite                         K(U6+O2)As5+O4·3H2O
4         Abhurite                          Sn2+21O6(OH)14Cl16

               IMA Chemistry (plain) Chemistry Elements   IMA Number  \
0                   NaPb2(CO3)2(OH)        Na Pb C O H  IMA2014-111
1                         NiC31H32N4           Ni C H N  IMA1975-013
2  Na26Ce6(Si6O18)(PO4)6(CO3)6(SO2)O   Na Ce Si O P C S  IMA1991-054
3                 K(UO2)(AsO4)·3H2O         K U O As H          NaN
4                Sn2+21O6(OH)14Cl16          Sn O H Cl  IMA1983-061

  RRUFF IDs Country of Type Locality  Year First Published  \
0       NaN                   Spain                2014.0
1   R070007                     USA                1975.0
2       NaN                  Canada                1991.0
3       NaN                     USA                1956.0
4   R060227            Saudi Arabia                1983.0

               IMA Status Structural Groupname Fleischers Groupname  \
0                Approved
1                Approved
2                Approved
3  Grandfathered|Approved       Natroautunite             autunite
4                Approved

                                      Status Notes Crystal Systems  \
0  Ibáñez-Insa J, Elvira J J, Llovet X, Pérez-Can…       hexagonal
1  Milton C, Dwornik E J, Estep-Barnes P A, Finke…       triclinic
2  McDonald A M, Chao G Y, Grice J D (1994) Abena…       hexagonal
3  Thompson M E, Ingram B, Gross E B (1956) Abern…      tetragonal
4  Matzko J J, Evans H T, Mrose M E, Aruscavage P…       hexagonal

   Oldest Known Age (Ma)
0                  370.0
1                   56.0
2                  124.0
3                  358.9
4                    0.0
```

Pero si queremos mostrar más filas, basta con ingresar entre los parentésis el número de filas a previsualizar. Probemos con 8:

[5]:
```
datos_e1.head(8)
```

[5]:

	Mineral Name	RRUFF Chemistry (plain)
0	Abellaite	NaPb2+2(CO3)2(OH)
1	Abelsonite	Ni2+C31H32N4
2	Abenakiite-(Ce)	Na26Ce3+6(SiO3)6(P5+O4)6(C4+O3)6(S4+O2)O
3	Abernathyite	K(U6+O2)As5+O4·3H2O
4	Abhurite	Sn2+21O6(OH)14Cl16
5	Abramovite	Pb2SnInBiS7
6	Abswurmbachite	Cu2+Mn3+6O8(SiO4)
7	Abuite	CaAl2(PO4)2F2

	IMA Chemistry (plain)	Chemistry Elements	IMA Number
0	NaPb2(CO3)2(OH)	Na Pb C O H	IMA2014-111
1	NiC31H32N4	Ni C H N	IMA1975-013
2	Na26Ce6(Si6O18)(PO4)6(CO3)6(SO2)O	Na Ce Si O P C S	IMA1991-054
3	K(UO2)(AsO4)·3H2O	K U O As H	NaN
4	Sn2+21O6(OH)14Cl16	Sn O H Cl	IMA1983-061
5	Pb2SnInBiS7	Pb Sn In Bi S	IMA2006-016
6	Cu2+Mn3+6O8(SiO4)	Cu Mn O Si	IMA1990-007
7	CaAl2(PO4)2F2	Ca Al P O F	IMA2014-084

	RRUFF IDs	Country of Type Locality	Year First Published
0	NaN	Spain	2014.0
1	R070007	USA	1975.0
2	NaN	Canada	1991.0
3	NaN	USA	1956.0
4	R060227	Saudi Arabia	1983.0
5	R070037	Russia	2006.0
6	NaN	Greece	1990.0
7	NaN	Japan	2014.0

	IMA Status	Structural Groupname	Fleischers Groupname
0	Approved		
1	Approved		
2	Approved		
3	Grandfathered|Approved	Natroautunite	autunite
4	Approved		
5	Approved		
6	Approved	Braunite	braunite
7	Approved		

	Status Notes	Crystal Systems
0	Ibáñez-Insa J, Elvira J J, Llovet X, Pérez-Can…	hexagonal
1	Milton C, Dwornik E J, Estep-Barnes P A, Finke…	triclinic
2	McDonald A M, Chao G Y, Grice J D (1994) Abena…	hexagonal
3	Thompson M E, Ingram B, Gross E B (1956) Abern…	tetragonal
4	Matzko J J, Evans H T, Mrose M E, Aruscavage P…	hexagonal
5	Yudovakaya M A, Trybkin N V, Koporulina E V, B…	triclinic

```
6  Reinecke T, Tillmanns E, Bernhardt H J (1991) …      tetragonal
7  Enju S, Uehara S (2017) Abuite, CaAl_2_(PO_4_)…    orthorhombic

   Oldest Known Age (Ma)
0                 370.00
1                  56.00
2                 124.00
3                 358.90
4                   0.00
5                   0.04
6                 541.00
7                    NaN
```

Si queremos confirmar que nuestros datos (datos_e1) son un *DataFrame* y queremos conocer que tipo de valores son, basta escribir la siguiente sintexis:

[6]:
```
type(datos_e1)
```

[6]:
```
pandas.core.frame.DataFrame
```

[7]:
```
#confirmamos que es un DataFrame o un "pandas.core.frame.DataFrame" como Python lo usa internamente

datos_e1.dtypes
```

[7]:
```
Mineral Name                 object
RRUFF Chemistry (plain)      object
IMA Chemistry (plain)        object
Chemistry Elements           object
IMA Number                   object
RRUFF IDs                    object
Country of Type Locality     object
Year First Published        float64
IMA Status                   object
Structural Groupname         object
Fleischers Groupname         object
Status Notes                 object
Crystal Systems              object
Oldest Known Age (Ma)       float64
dtype: object
```

Observamos que la mayoría de los valores corresponden con tipo *object* y vemos que sólo dos valores corresponden con tipo flotante (float64). Es importante que tengamos certeza de los tipos de valores que están almacenados en el *DataFrame* para poder manipularlos posteriormente.

[8]:
```
datos_e1.head(3)      #Observa la primera columna que se observa en el DataFrame
```

[8]:
```
      Mineral Name                    RRUFF Chemistry (plain)  \
0        Abellaite                          NaPb2+2(CO3)2(OH)
1       Abelsonite                               Ni2+C31H32N4
2  Abenakiite-(Ce)  Na26Ce3+6(SiO3)6(P5+O4)6(C4+O3)6(S4+O2)O

               IMA Chemistry (plain) Chemistry Elements   IMA Number  \
0                   NaPb2(CO3)2(OH)        Na Pb C O H  IMA2014-111
1                         NiC31H32N4           Ni C H N  IMA1975-013
2  Na26Ce6(Si6O18)(PO4)6(CO3)6(SO2)O   Na Ce Si O P C S  IMA1991-054

  RRUFF IDs Country of Type Locality  Year First Published IMA Status  \
0       NaN                    Spain                2014.0   Approved
1   R070007                      USA                1975.0   Approved
2       NaN                   Canada                1991.0   Approved

  Structural Groupname Fleischers Groupname  \
0
1
2

                                      Status Notes Crystal Systems  \
0  Ibáñez-Insa J, Elvira J J, Llovet X, Pérez-Can…       hexagonal
1  Milton C, Dwornik E J, Estep-Barnes P A, Finke…       triclinic
2  McDonald A M, Chao G Y, Grice J D (1994) Abena…       hexagonal

   Oldest Known Age (Ma)
0                  370.0
1                   56.0
2                  124.0
```

Como puedes ver, es una numeración que inicia en 0 y corresponde con el índice del *DataFrame*. El índice se utiliza para identificar la posición de los datos (no es una columna del *DataFrame*). ¿Y por qué comienza en cero? Porque Python usa indexación base 0.

[9]:
```
datos_e1.columns    #si queremos obtener los encabezados de cada columna del␣
↪data frame
```

[9]:
```
Index(['Mineral Name', 'RRUFF Chemistry (plain)', 'IMA Chemistry (plain)',
       'Chemistry Elements', 'IMA Number', 'RRUFF IDs',
       'Country of Type Locality', 'Year First Published', 'IMA Status',
       'Structural Groupname', 'Fleischers Groupname', 'Status Notes',
       'Crystal Systems', 'Oldest Known Age (Ma)'],
      dtype='object')
```

Si queremos conocer cuantas filas y cuantas columnas contiene el *DataFrame*, llamamos al método *shape/

[10]:
```
datos_e1.shape      #nombre_archivo.shape
```

[10]: (5739, 14)

1.3 Limpieza de los datos

Supongamos que algunas filas no son de utilidad

[11]:
```
datos_e1.drop(3, axis=0, inplace=True) #borra la fila con índice 3, recuerda que Python usa indexación en base cero
#Para borrar filas o columnas se usa el método drop
#axis=0 para borrar fila y axis=1 para borrar columna
#inplace=true altera? el dataframe mientras que inplace=false, NO modifica el dataframe y devuelve uno filas o columnas eliminadas
```

[12]:
```
datos_e1.head()
```

[12]:

	Mineral Name	RRUFF Chemistry (plain)	IMA Chemistry (plain)	Chemistry Elements	IMA Number	RRUFF IDs	Country of Type Locality	Year First Published	IMA Status	Structural Groupname	Fleischers Groupname	Status Notes	Crystal Systems
0	Abellaite	NaPb2+2(CO3)2(OH)	NaPb2(CO3)2(OH)	Na Pb C O H	IMA2014-111	NaN	Spain	2014.0	Approved			Ibáñez-Insa J, Elvira J J, Llovet X, Pérez-Can…	hexagonal
1	Abelsonite	Ni2+C31H32N4	NiC31H32N4	Ni C H N	IMA1975-013	R070007	USA	1975.0	Approved			Milton C, Dwornik E J, Estep-Barnes P A, Finke…	triclinic
2	Abenakiite-(Ce)	Na26Ce3+6(SiO3)6(P5+O4)6(C4+O3)6(S4+O2)O	Na26Ce6(Si6O18)(PO4)6(CO3)6(SO2)O	Na Ce Si O P C S	IMA1991-054	NaN	Canada	1991.0	Approved			McDonald A M, Chao G Y, Grice J D (1994) Abena…	hexagonal
4	Abhurite	Sn2+21O6(OH)14Cl16	Sn2+21O6(OH)14Cl16	Sn O H Cl	IMA1983-061	R060227	Saudi Arabia	1983.0	Approved			Matzko J J, Evans H T, Mrose M E, Aruscavage P…	hexagonal
5	Abramovite	Pb2SnInBiS7	Pb2SnInBiS7	Pb Sn In Bi S	IMA2006-016	R070037	Russia	2006.0	Approved				

```
5  Yudovakaya M A, Trybkin N V, Koporulina E V, B…       triclinic

   Oldest Known Age (Ma)
0                 370.00
1                  56.00
2                 124.00
4                   0.00
5                   0.04
```

[13]:
```
list(datos_e1.columns)
```

[13]:
```
['Mineral Name',
 'RRUFF Chemistry (plain)',
 'IMA Chemistry (plain)',
 'Chemistry Elements',
 'IMA Number',
 'RRUFF IDs',
 'Country of Type Locality',
 'Year First Published',
 'IMA Status',
 'Structural Groupname',
 'Fleischers Groupname',
 'Status Notes',
 'Crystal Systems',
 'Oldest Known Age (Ma)']
```

Ahora cambiemos los nombres anteriores y los reemplazaremos, todo en un diccionario. Utilizamos la siguiente sintaxis: nombre_archivo.rename(columns={'nombre_anterior': 'nuevo_nombre'}, inplace=True)

[14]:
```
datos_e1.rename(columns={'Mineral Name':'Mineral','Chemistry Elements':
↪'Elements', 'Country of Type Locality' : 'Country', 'Crystal Systems':
↪'Systems', 'Oldest Known Age (Ma)' : 'Age', 'Structural Groupname' :␣
↪'Groupname' }, inplace=True)
```

[15]:
```
#datos_e1.sample(5)   #el método shape nos permite ver una muestra, en este␣
↪caso 5
                        #borra el primer gato para ver que sucede
```

[16]:
```
#Ahora observemos que y cuántos valores son nulos:
datos_e1.isnull().sum(axis = 0)  #nombre_archivo.isnull().sum(axis=0)
# axis=0 los analiza por fila
```

[16]:
```
Mineral                      0
RRUFF Chemistry (plain)      0
IMA Chemistry (plain)        4
Elements                     4
IMA Number                2103
```

```
RRUFF IDs                 3498
Country                    199
Year First Published       202
IMA Status                 122
Groupname                    0
Fleischers Groupname         0
Status Notes                68
Systems                    502
Age                        947
dtype: int64
```

[17]:
```
#Almacenamos los valores nulos de 'Elements' y de 'IMA Chemistry (plain)'en␣
↪valores_nulos
valores_nulos = datos_e1['Elements'].isnull() | datos_e1['IMA Chemistry␣
↪(plain)'].isnull()
datos_e1[valores_nulos] #visualizamos
```

[17]:
```
                Mineral                    RRUFF Chemistry (plain)  \
277   Arrojadite-(NaFe)  Na2Fe2+(CaNa2)Fe2+13Al(PO4)11(PO3OH)(OH)2
1752           Fluorine                                           F
4020         Perkovaite                                 CaMg2(SO4)3
4974     Tadzhikite-(Y)           Ca3(Y,Ce)2(Ti,Al,Fe3+)B4Si4O22

     IMA Chemistry (plain) Elements IMA Number RRUFF IDs Country  \
277                    NaN      NaN        NaN   R070298     NaN
1752                   NaN      NaN        NaN       NaN  France
4020                   NaN      NaN        NaN       NaN     NaN
4974                   NaN      NaN        NaN       NaN     NaN

      Year First Published IMA Status Groupname Fleischers Groupname  \
277                    NaN        NaN
1752                1886.0        NaN
4020                   NaN        NaN
4974                   NaN        NaN

     Status Notes     Systems  Age
277           NaN  monoclinic  NaN
1752          NaN         NaN  NaN
4020          NaN   hexagonal  NaN
4974          NaN         NaN  NaN
```

[18]:
```
datos_e1 = datos_e1[datos_e1.Elements.notna()] #notna: detecta valores válidos␣
↪(que no contienen NaNs)
datos_e1
```

[18]:
```
              Mineral                  RRUFF Chemistry (plain)  \
0           Abellaite                        NaPb2+2(CO3)2(OH)
```

1	Abelsonite	Ni2+C31H32N4
2	Abenakiite-(Ce)	Na26Ce3+6(SiO3)6(P5+O4)6(C4+O3)6(S4+O2)O
4	Abhurite	Sn2+21O6(OH)14Cl16
5	Abramovite	Pb2SnInBiS7
…	…	…
5734	Zussmanite	K(Fe2+,Mg,Mn2+,Fe3+)13(Si,Al)18O42(OH)14
5735	Zvyaginite	Na2Zn2+Ti4+Nb5+2(Si2O7)2O2(OH)2(H2O)4
5736	Zvyagintsevite	Pd3Pb
5737	Zwieselite	Fe2+2PO4F
5738	Zýkaite	Fe3+4(As5+O4)3S6+O4(OH)·15H2O

	IMA Chemistry (plain)	Elements	IMA Number
0	NaPb2(CO3)2(OH)	Na Pb C O H	IMA2014-111
1	NiC31H32N4	Ni C H N	IMA1975-013
2	Na26Ce6(Si6O18)(PO4)6(CO3)6(SO2)O	Na Ce Si O P C S	IMA1991-054
4	Sn2+21O6(OH)14Cl16	Sn O H Cl	IMA1983-061
5	Pb2SnInBiS7	Pb Sn In Bi S	IMA2006-016
…	…	…	…
5734	K(Fe,Mg,Mn)13(Si,Al)18O42(OH)14	K Fe Mg Mn Si Al O H	IMA1964-018
5735	Na2ZnTiNb2(Si2O7)2O2(OH)2(H2O)4	Na Zn Ti Nb Si O H	IMA2013-071
5736	Pd3Pb	Pd Pb	IMA1966-006
5737	Fe2+2(PO4)F	Fe P O F	NaN
5738	Fe3+4(AsO4)3(SO4)(OH)·15H2O	Fe As O S H	IMA1976-039

	RRUFF IDs	Country	Year First Published	IMA Status
0	NaN	Spain	2014.0	Approved
1	R070007	USA	1975.0	Approved
2	NaN	Canada	1991.0	Approved
4	R060227	Saudi Arabia	1983.0	Approved
5	R070037	Russia	2006.0	Approved
…	…	…	…	…
5734	R050448	USA	1964.0	Approved
5735	NaN	Russia	2013.0	Redefined|Approved
5736	NaN	Russia	1966.0	Approved
5737	R050279	Germany	1841.0	Redefined|Approved
5738	R070477	Czech Republic	1976.0	Approved

	Groupname	Fleischers Groupname
0		
1		
2		
4		
5		
…	…	…
5734	Clay	
5735	Seidozerite-Lamprophyllite	
5736	Perovskite	zvyagintsevite

```
5737                    Triplite             triplite
5738
```

		Status Notes	Systems	\
0	Ibáñez-Insa J, Elvira J J, Llovet X, Pérez-Can…		hexagonal	
1	Milton C, Dwornik E J, Estep-Barnes P A, Finke…		triclinic	
2	McDonald A M, Chao G Y, Grice J D (1994) Abena…		hexagonal	
4	Matzko J J, Evans H T, Mrose M E, Aruscavage P…		hexagonal	
5	Yudovakaya M A, Trybkin N V, Koporulina E V, B…		triclinic	
…	…	…	…	
5734	Agrell S O, Bown M G, McKie D (1965) Deerite, …		hexagonal	
5735	Pekov I V, Lykova I S, Chukanov N V, Yapaskurt…		triclinic	
5736	Genkin A D, Murav´eva I V, Troneva N V (1966) …		cubic	
5737	Breithaupt J F A (1841) Phyletites ferrosus od…		monoclinic	
5738	Čech F, Jansa J, Novák F (1978) Zýkaite, Fe^3+…	orthorhombic|unknown		

	Age
0	370.00
1	56.00
2	124.00
4	0.00
5	0.04
…	…
5734	NaN
5735	370.00
5736	3100.00
5737	1820.00
5738	0.00

```
[5734 rows x 14 columns]
```

[19]:
```
#si queremos corroborar que dichos elementos fueron borrados, (sólo quita los #␣
↪de las siguientes dos líneas):
valores_nulos = datos_e1['Elements'].isnull() | datos_e1['IMA Chemistry␣
↪(plain)'].isnull()
datos_e1[valores_nulos]
```

[19]:
```
Empty DataFrame
Columns: [Mineral, RRUFF Chemistry (plain), IMA Chemistry (plain), Elements, IMA
Number, RRUFF IDs, Country, Year First Published, IMA Status, Groupname,
Fleischers Groupname, Status Notes, Systems, Age]
Index: []
```

[20]:
```
#Pudimos observar que 199 minerales no tienen un valor asignado en la columna␣
↪'country',
#procederemos a reemplazarlo por la palabra 'desconocido'
datos_e1['Country'] = datos_e1.Country.fillna(value = 'desconocido')
```

```
#podemos corroborar que ya no tenemos valores nulos en la columna de país:
#datos_e1.isnull().sum(axis = 0)
```

[21]:
```
#En la columna 'Elements' de este DataFrame aparecen los elementos que␣
↪constituyen al mineral en forma horizontal, los guardaremos de manera␣
↪vertical
#para poder manipularlos
elem = datos_e1.set_index('Mineral').Elements.str.split(' ', expand=True).
↪stack().reset_index('Mineral').reset_index(drop=True)
elem.columns = ['Mineral', 'Element']
elem   #ahora se repite el mineral y cada fila muestra un elemento diferente␣
↪para el mismo mineral
```

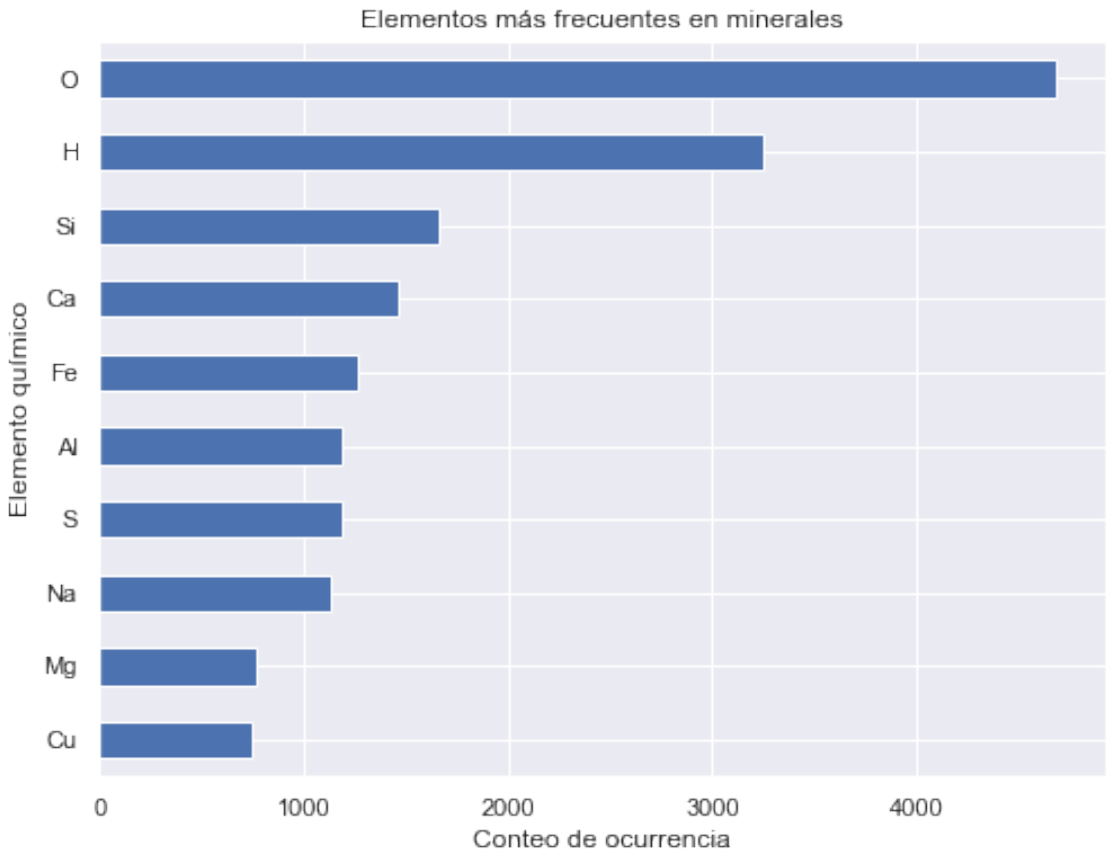
[21]:

	Mineral	Element
0	Abellaite	Na
1	Abellaite	Pb
2	Abellaite	C
3	Abellaite	O
4	Abellaite	H
…	…	…
27222	Zýkaite	Fe
27223	Zýkaite	As
27224	Zýkaite	O
27225	Zýkaite	S
27226	Zýkaite	H

[27227 rows x 2 columns]

1.4 Elaboración de gráficos

[22]:
```
#Si queremos ver el "top 10" de elementos que constituyen el mineral:
elem['Element'].value_counts()[0:10].sort_values().plot(kind='barh',␣
↪figsize=(8, 6))
plt.xlabel("Conteo de ocurrencia")
plt.ylabel("Elemento químico")
plt.title("Elementos más frecuentes en minerales");
```



Analizando la gráfica anterior, podemos observar que de los más de 5000 minerales, ¡más de 4000 minerales contienen oxígeno en su estructura cristalina! ¿Te hace sentido? ¿Por qué le siguen el Hidrógeno y Silicio? Te lo dejo de tarea. También te invito a modificar el valor de 10, para que analices más elementos y su ocurrencia.

[23]:
```
##Ahora analicemos en que país se tienen registrados la mayor cantidad de␣
↪minerales
datos_e1['Country'].value_counts()
```

[23]:
```
USA
811
Russia
790
Italy
355
Germany
346
Canada
228
…
```

```
India ?
1
Georgia / Israel
1
Italy (meteorite) / India (meteorite) / Poland (meteorite) / USA (meteorite)
1
USA/Germany
1
Oman
1
Name: Country, Length: 219, dtype: int64
```

[24]:
```
#Algunos minerales en la columna country incluyen la palabra 'meteorite'
countr = datos_e1.set_index('Mineral').Country.str.split(' / ', expand=True).
↪stack().reset_index('Mineral').reset_index(drop=True)
countr.columns = ['Mineral', 'Country']
```

[25]:
```
print(countr[countr['Country'].str.contains('meteorite', regex=False)]) #str.
↪contains busca si la cadena(string) contiene la palabra 'meteorite'
```

```
              Mineral               Country
18    Addibischoffite    Algeria (meteorite)
24          Adrianite     Mexico (meteorite)
49          Ahrensite    Morocco (meteorite)
59         Akimotoite  Australia (meteorite)
87      Allabogdanite     Russia (meteorite)
…                 …                  …
5554          Warkite      Italy (meteorite)
5664     Xenophyllite    Ukraine (meteorite)
5668           Xieite      China (meteorite)
5669        Xifengite      China (meteorite)
5728        Zagamiite    Morocco (meteorite)

[99 rows x 2 columns]
```

[26]:
```
print(countr[countr['Country'].str.contains('?', regex=False)])
#str.contains busca si la cadena(string) contiene el caracter '?'
#Antes modificamos los valores nulos en "country", aún así, existen datos que␣
↪contienen un signo de interrogación o el nombre del país seeguido de '?'
```

```
             Mineral        Country
137          Alum-(K)        Italy ?
138         Alum-(Na)              ?
148          Alunogen              ?
304      Arsenopyrite              ?
324          Asbolane              ?
350            Augite              ?
439            Baryte              ?
```

```
569      Bismuthinite              ?
629           Bornite              ?
927         Chalcocite             ?
933       Chalcopyrite             ?
1143       Cordierite      Germany ?
1154         Corundum        India ?
1496        Erythrite      Germany ?
1641  Ferroalluaudite          USA ?
2457          Iridium       Russia ?
3169        Magnetite              ?
3388         Mesolite      Iceland ?
4140     Phylloretine      Denmark ?
5043           Taenite  New Zealand ?
```

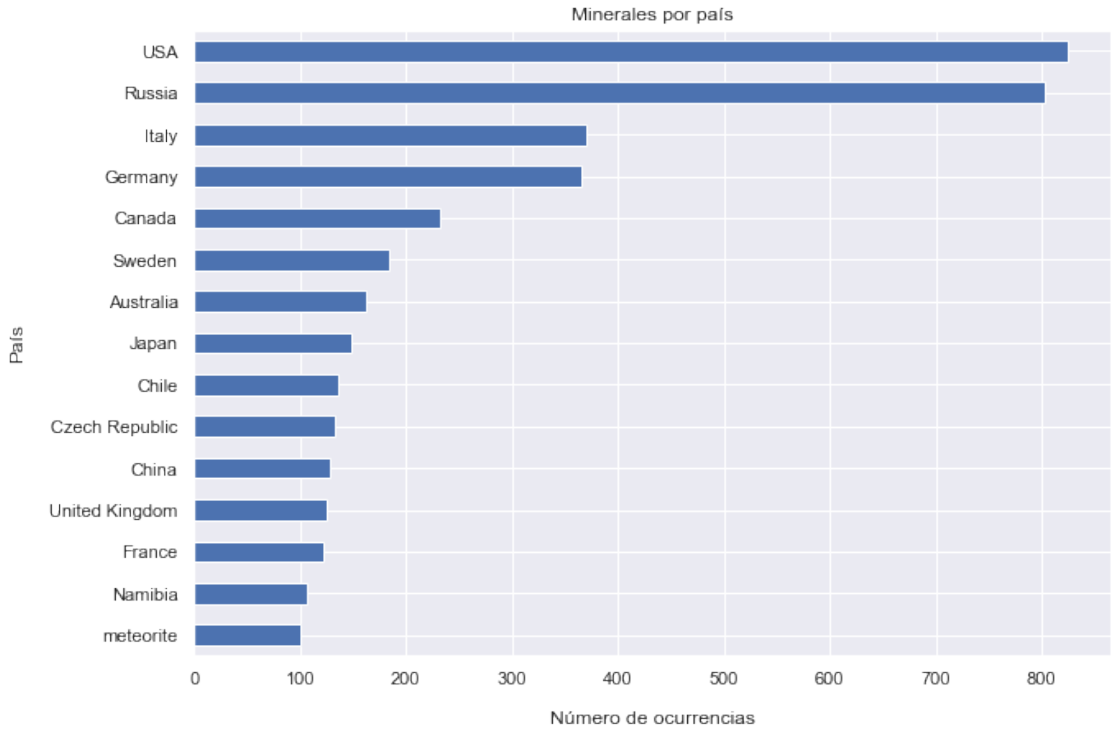
[27]:
```
countr['Country'] = countr['Country'].replace({' \?':''}, regex=True)␣
↪#reemplaza '?' por un string vacío
countr.loc[countr['Country'].str.contains('meteorite', case=False), 'Country']␣
↪= 'meteorite' #si la celda tiene algún país seguido de 'meteorite'
countr['Country'] = countr['Country'].replace('?', 'desconocido') #reemplaza '?
↪' por 'desconocido'
countr.loc[countr['Country'].str.contains('IDP', case=False), 'Country'] =␣
↪'IDP' #reemplaza IDP over USA por 'IDP' (esta instrucción es para el mineral␣
↪ Brownleeita)
```

[28]:
```
countr['Country'].str.strip() #str.string remueve los espacios al inicio al␣
↪final de la cadena (string)
countr.drop(countr[countr.Country == 'desconocido'].index, inplace=True) #quita␣
↪los minerales que no tienen país ('desconocido')
countr['Country'].value_counts()[0:20] #value_counts devuelve una serie que␣
↪contiene recuentos de valores únicos
#[0:20] muestra los primeros 20 valores
```

[28]:
```
USA                      824
Russia                   803
Italy                    370
Germany                  366
Canada                   233
Sweden                   185
Australia                162
Japan                    148
Chile                    137
Czech Republic           133
China                    129
United Kingdom           126
France                   122
Namibia                  107
meteorite                101
```

```
Democratic Republic of the Congo     99
Switzerland                          91
Norway                               85
Denmark (Greenland)                  81
Kazakhstan                           78
Name: Country, dtype: int64
```

[29]:
```
countr['Country'].value_counts()[0:15].sort_values().plot(kind='barh',
 ↪figsize=(10, 7))
#value_counts()[0:15] solo grafica los primeros 15
#sort.values ordena los valores, empezando por el valor más alto
#kind='barh' muestra las barras de manera horizontal
plt.xlabel("Número de ocurrencias", labelpad=14) #labelpad es la distancia
 ↪entre los ticklabels? y la etiqueta del eje
plt.ylabel("País", labelpad=14)
plt.title("Minerales por país");
```



Los países con mayor cantidad de minerales registrados en la IMA (International Mineralogical Asocciation) son Estados Unidos y Rusia. ¿Tiene que ver con la superficie de cada uno? ¿Con la investigación que cada país dedica?

[30]:
```
#Por último hagamos una comparación de minerales por país:
countr = countr[countr['Country'].map(countr['Country'].value_counts()) >= 140]
```

```
#map ejecuta una función específica para cada elemento en una iterable NO␣
 ↪ENTENDÍ
elem = elem[elem['Element'].map(elem['Element'].value_counts()) >= 600]
#la función merge permite unir dataramess, puede ser realizado sobre columnas o␣
 ↪sobre filas
result = pd.merge(countr, elem, on='Mineral') #on='Mineral' especifica que la␣
 ↪unión de las columnas 'countr' y 'elem' las hace mediante la columna común a␣
 ↪ambos 'Mineral'
result
```

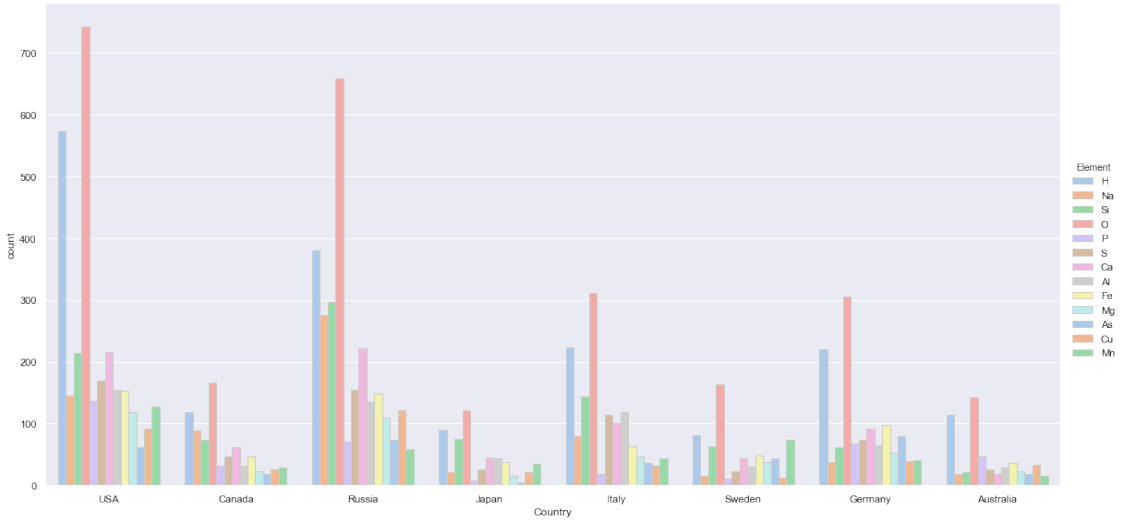
[30]:

	Mineral	Country	Element
0	Abelsonite	USA	H
1	Abenakiite-(Ce)	Canada	Na
2	Abenakiite-(Ce)	Canada	Si
3	Abenakiite-(Ce)	Canada	O
4	Abenakiite-(Ce)	Canada	P
…	…	…	…
10644	Zvyaginite	Russia	O
10645	Zvyaginite	Russia	H
10646	Zwieselite	Germany	Fe
10647	Zwieselite	Germany	P
10648	Zwieselite	Germany	O

[10649 rows x 3 columns]

[31]:
```
sns.catplot(x="Country", hue="Element", kind="count", palette="pastel",␣
 ↪edgecolor=".8", data=result, height=8, aspect = 2)
#catplot permite realizar gráficos con datos categóricos
#hue=parámetro
#kind="count" indicas que la longitud de la barra sea proporcional al número de␣
 ↪elementos
#data se refiere al dataset a plotear, en este caso "result"
```

[31]: <seaborn.axisgrid.FacetGrid at 0x243fa395308>



¿Cuáles son los países más ricos en elementos químicos y minerales?¿Es viable explotar la mayoría de los minerales? A este punto de la carrera sabemos que no, ¿por qué?

Este ejercicio es la introducción al análisis de datos en Python para profesionales de Ciencias de la Tierra. Los datos de minerales por país con los que trabajamos son de tipo categórico. ¿Qué tal si ahora utilizamos set de datos que incluyan valores cuantitativos? Imagina cuántos cálculos y gráficas puedes realizar en cuestión de minutos, y que además se puede compartir con otros profesionales para seguir expandiendo el análisis.

Referencias:

1. Chemkaeva, D. (2019). Overview of minerals dataset. Kaggle. https://www.kaggle.com/lsind18/overview-of-minerals-dataset

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 2. CÁLCULO DE MEDIDAS DE TENDENCIA CENTRAL

April 3, 2022

1 EJERCICIO 2. CÁLCULO DE MEDIDAS DE TENDENCIA CENTRAL

Autora: Brenda Ortiz Soto

En este ejercicio, se obtendrán la moda, la media y la mediana de un set de datos. Igualmente, se graficarán histogramas de los sets de datos. Esta vez vamos a usar las librerías: Pandas, Matplotlib, NumPy, SciPy, y Seaborn.

Para obtener el *dataset* puedes dar clic aquí.

Dicho *dataset* corresponde con información de sismos de magnitud 5.5 o mayor, que han ocurrido en todo el mundo desde 1965 hasta el 2016, e incluye la fecha y hora de ocurrencia, localidad del sismo, magnitud y profundidad. La información fue recolectada por el "Eartquake Information Center (NEIC)", encuentra más información del NEIC aquí.

1.1 Importación de librerías y lectura de datos

[1]:
```
#Importar librerías y asignarles un nombre corto
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.mlab as mlab
#puedes darte cuenta que importamos más librerías que en el ejercio anterior
#la razón: cada librería cuenta con funciones específicas y diferentes
```

[2]:
```
datos = pd.read_csv('e2_datos.csv') #los datos deben estar guardados en la␣
↪misma carpeta que el Jupyter Notebook
```

[3]:
```
datos.head()
```

[3]:

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	\
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	

```
3  01/08/1965  18:49:43   -59.076    -23.557  Earthquake   15.0          NaN
4  01/09/1965  13:32:50    11.938    126.427  Earthquake   15.0          NaN

   Depth Seismic Stations  Magnitude Magnitude Type  …  \
0                     NaN        6.0             MW  …
1                     NaN        5.8             MW  …
2                     NaN        6.2             MW  …
3                     NaN        5.8             MW  …
4                     NaN        5.8             MW  …

   Magnitude Seismic Stations  Azimuthal Gap  Horizontal Distance  \
0                         NaN            NaN                  NaN
1                         NaN            NaN                  NaN
2                         NaN            NaN                  NaN
3                         NaN            NaN                  NaN
4                         NaN            NaN                  NaN

   Horizontal Error  Root Mean Square            ID  Source Location Source  \
0               NaN               NaN  ISCGEM860706  ISCGEM          ISCGEM
1               NaN               NaN  ISCGEM860737  ISCGEM          ISCGEM
2               NaN               NaN  ISCGEM860762  ISCGEM          ISCGEM
3               NaN               NaN  ISCGEM860856  ISCGEM          ISCGEM
4               NaN               NaN  ISCGEM860890  ISCGEM          ISCGEM

  Magnitude Source     Status
0           ISCGEM  Automatic
1           ISCGEM  Automatic
2           ISCGEM  Automatic
3           ISCGEM  Automatic
4           ISCGEM  Automatic

[5 rows x 21 columns]
```

[4]:
```
datos.shape  #para saber cuantas filas y columnas contiene el dataframe
```

[4]:
```
(23412, 21)
```

[5]:
```
#De nuestro datos originales, haremos un dataframe llamado datos1 que sólo␣
↪incluya las columnas correspondientes a fecha, hora, latitud, longitud,␣
↪profundidad y magnitud
datos1 = datos[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
datos1.head() #para mostrar las primeras 5 líneas del dataframe
```

[5]:
```
         Date      Time  Latitude  Longitude  Depth  Magnitude
0  01/02/1965  13:44:18    19.246    145.616  131.6        6.0
1  01/04/1965  11:29:49     1.863    127.352   80.0        5.8
2  01/05/1965  18:05:58   -20.579   -173.972   20.0        6.2
```

```
3  01/08/1965  18:49:43   -59.076    -23.557   15.0        5.8
4  01/09/1965  13:32:50    11.938    126.427   15.0        5.8
```

[6]:
```
#comprobamos que el dataset no contenga datos nulos
valores_nulos = datos1.isnull().sum()
valores_nulos
#podemos observar que el dataframe está completo y no requiere modificar o␣
 ↪borrar datos como en el ejercicio 1
```

[6]:
```
Date         0
Time         0
Latitude     0
Longitude    0
Depth        0
Magnitude    0
dtype: int64
```

1.2 Resumen estadístico

Python nos permite obtener un resumen estadístico de cada variable o de la variable que le especifiquemos, mediante el cual obtenemos: el conteo total, el promedio, desviación estándar, valor mínimo, cuartiles y valor máximo. Obteniendo el resumen estadístico de todas las variables:

[7]:
```
datos1.describe().transpose()   #dataframe.describe().transpose
#en este caso, no nos es de utilidad obtener los datos estadísticos de la␣
 ↪latitud y longitud pero se incluyen como ejemplo
```

[7]:
```
              count       mean         std      min       25%       50%  \
Latitude    23412.0   1.679033   30.113183  -77.080 -18.65300   -3.5685
Longitude   23412.0  39.639961  125.511959 -179.997 -76.34975  103.9820
Depth       23412.0  70.767911  122.651898   -1.100  14.52250   33.0000
Magnitude   23412.0   5.882531    0.423066    5.500   5.60000    5.7000

                 75%      max
Latitude    26.19075   86.005
Longitude  145.02625  179.998
Depth       54.00000  700.000
Magnitude    6.00000    9.100
```

Y si queremos visualizar el resumen estadístico para una variable, en este caso para la magnitud:

[8]:
```
datos1['Magnitude'].describe() # la sintaxis es: nombre_archivo['variable'].
 ↪describe()
```

[8]:
```
count    23412.000000
mean         5.882531
std          0.423066
min          5.500000
```

```
25%          5.600000
50%          5.700000
75%          6.000000
max          9.100000
Name: Magnitude, dtype: float64
```

Obteniendo las medidas de tendencia central. De la lista, promedio es el mean y la mediana es el estadístico que dice 50%

[9]:
```
#la sintaxis para obtener la moda de la variable 'Magnitude' es:␣
 ↪nombre_archivo['variable'].mode
#notar que sólo imprime los resultados, si deseamos guardar los datos, debemos␣
 ↪crear variables para guardarlos
#Para este ejercicio sólo guardaremos la media y la desviación estándar
datos1['Magnitude'].mode()
```

[9]:
```
0    5.5
dtype: float64
```

[10]:
```
#la sintaxis para obtener la mediana de la variable 'Magnitude' es:␣
 ↪nombre_archivo['variable'].median
datos1['Magnitude'].median()
```

[10]:
```
5.7
```

[11]:
```
#la sintexis para obtener la media de la variable 'Magnitude' es:␣
 ↪nombre_archivo['variable'].mean
media = datos1['Magnitude'].mean()
media
```

[11]:
```
5.882530753460003
```

[12]:
```
#la sintexis para obtener la media geométrica de la variable 'Magnitude' es:␣
 ↪stats.gmean(nombre_archivo['variable'])
stats.gmean(datos1['Magnitude'])
```

[12]:
```
5.868444764632867
```

Puedes notar que mediante el método describe obtuvimos la media (mean) y la mediana (la cual corresponde con el segundo cuartil o 50%). También obtuvimos dichos valores con las funciones mean y median que pertenecen al módulo stats de scipy (fíjate en la primera celda de este ejercicio, ahí es donde importamos stats)

Ahora obtengamos las medidas de dispersión:

[13]:
```
#la sintaxis para obtener la varianza de la variable 'Magnitude' es:␣
 ↪nombre_archivo['variable'].var()
datos1['Magnitude'].var()
```

[13]:
```
0.17898453516966728
```

[14]:
```
#la sintexis para obtener la desviación estándar de la variable 'Magnitude' es:␣
↪nombre_archivo['variable'].std
std = datos1['Magnitude'].std()
std
```

[14]:
```
0.4230656393157772
```

[15]:
```
#la sintexis para obtener los cuartiles de la variable 'Magnitude' es:␣
↪nombre_archivo['variable'].quntile([.25,.5,.75])
datos1['Magnitude'].quantile([.25,.5,.75])
```

[15]:
```
0.25    5.6
0.50    5.7
0.75    6.0
Name: Magnitude, dtype: float64
```

[16]:
```
#Imaginemos que quieres calcular percentiles de la variable 'Magnitude' es:␣
↪nombre_archivo['variable'].quntile([percentil])
#Por ejemplo el percentil 80 y el 90
datos1['Magnitude'].quantile([.8,.9])
```

[16]:
```
0.8    6.1
0.9    6.4
Name: Magnitude, dtype: float64
```

1.3 Agrupación de datos

[17]:
```
#Imaginemos que también queremos agrupar todos los datos por "Magnitud" y␣
↪definir sus parámetros estadísticos a partir de datos agrupados
grouped_data = datos1.groupby('Magnitude') #datos agrupados por magnitud
grouped_data.describe()
```

[17]:

	Latitude						\
	count	mean	std	min	25%	50%	
Magnitude							
5.50	4685.0	0.559046	30.477473	-77.0800	-20.39100	-4.133000	
5.51	1.0	34.144000	NaN	34.1440	34.14400	34.144000	
5.52	4.0	37.476125	0.441895	37.2315	37.24975	37.267333	
5.53	1.0	34.162000	NaN	34.1620	34.16200	34.162000	
5.54	1.0	37.300500	NaN	37.3005	37.30050	37.300500	
…	…	…	…	…	…	…	
8.40	2.0	-10.351500	8.362952	-16.2650	-13.30825	-10.351500	
8.60	2.0	2.206000	0.171120	2.0850	2.14550	2.206000	
8.70	1.0	51.251000	NaN	51.2510	51.25100	51.251000	
8.80	1.0	-36.122000	NaN	-36.1220	-36.12200	-36.122000	

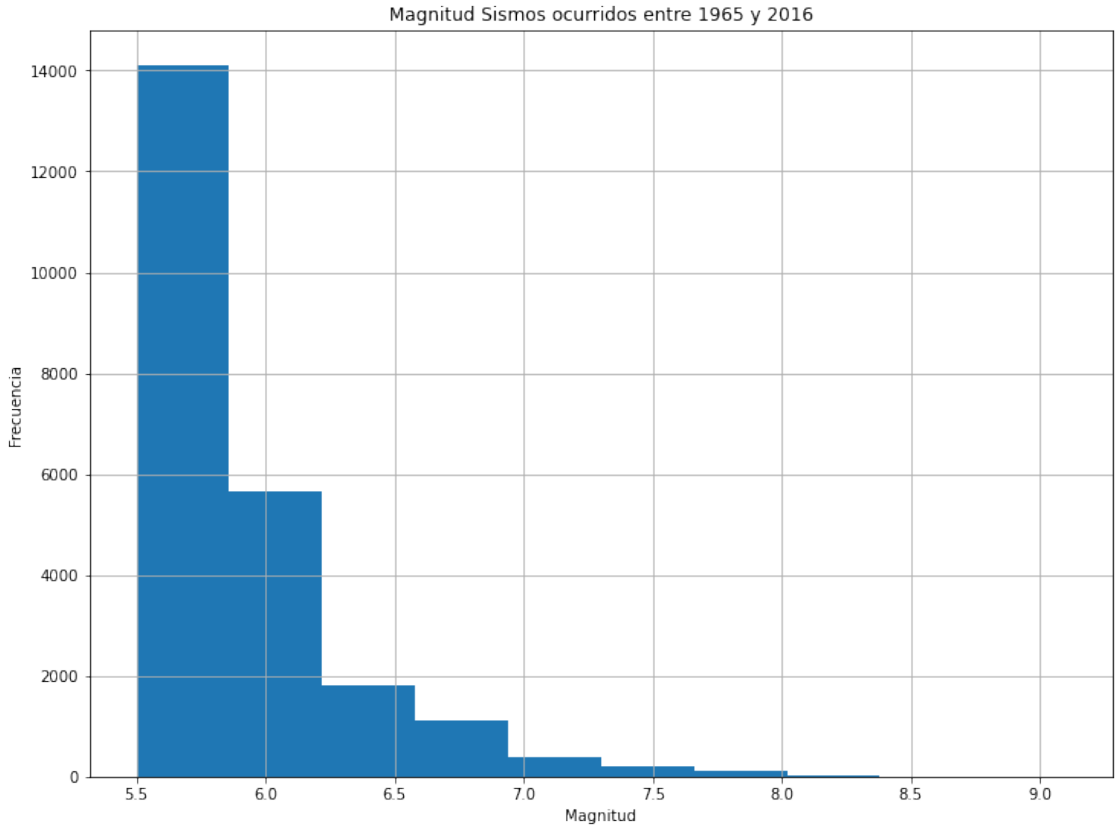
```
9.10           2.0  20.796000  24.750152   3.2950  12.04550  20.796000

                                Longitude              …              \
                  75%        max     count        mean  …         75%
Magnitude                                                 …
5.50        23.532000  85.735000    4685.0   34.785978  …  143.473000
5.51        34.144000  34.144000       1.0 -117.697000  … -117.697000
5.52        37.493708  38.138333       4.0 -116.886375  … -116.354458
5.53        34.162000  34.162000       1.0 -116.852000  … -116.852000
5.54        37.300500  37.300500       1.0 -116.534167  … -116.534167
…                 …          …       …          …  …          …
8.40        -7.394750  -4.438000       2.0   13.863000  …   57.615000
8.60         2.266500   2.327000       2.0   95.085500  …   96.096750
8.70        51.251000  51.251000       1.0  178.715000  …  178.715000
8.80      -36.122000 -36.122000       1.0  -72.898000  …  -72.898000
9.10        29.546500  38.297000       2.0  119.177500  …  130.775250

                     Depth                                                \
                   max    count        mean         std     min     25%     50%
Magnitude
5.50        179.984000   4685.0   70.856438  120.984309   0.000  15.000  33.000
5.51       -117.697000      1.0    3.292000         NaN   3.292   3.292   3.292
5.52       -116.311833      4.0    3.575000    2.807579   0.900   1.275   3.700
5.53       -116.852000      1.0    9.615000         NaN   9.615   9.615   9.615
5.54       -116.534167      1.0    1.200000         NaN   1.200   1.200   1.200
…                  …      …         …          …      …      …      …
8.40        101.367000      2.0   33.500000    0.707107  33.000  33.250  33.500
8.60         97.108000      2.0   25.000000    7.071068  20.000  22.500  25.000
8.70        178.715000      1.0   30.300000         NaN  30.300  30.300  30.300
8.80        -72.898000      1.0   22.900000         NaN  22.900  22.900  22.900
9.10        142.373000      2.0   29.500000    0.707107  29.000  29.250  29.500


              75%      max
Magnitude
5.50        56.900  700.000
5.51         3.292    3.292
5.52         6.000    6.000
5.53         9.615    9.615
5.54         1.200    1.200
…              …      …
8.40        33.750   34.000
8.60        27.500   30.000
8.70        30.300   30.300
8.80        22.900   22.900
9.10        29.750   30.000
```

```
[64 rows x 24 columns]
```

1.4 Elaboración de histograma

Interpretar datos puede ser más fácil si utilizamos gráficas para su representación. Visualizemos los datos en un histograma y en un diagrama de caja o *boxplot*.

[18]:
```
datos1['Magnitude'].hist(figsize=(12,9),bins=10)       # la sintaxis es:␣
↪nombre_archivo['variable'].hist(figsize=(ancho, alto))
#bins=10 indica que queremos que el histograma tenga 10 clases
plt.title('Magnitud Sismos ocurridos entre 1965 y 2016')  #etiqueta el título␣
↪del histograma
plt.xlabel("Magnitud")                    #etiqueta eje "y"
plt.ylabel("Frecuencia")                  #etiqueta eje "y"
plt.show()                                #para visualizar el histograma
```



De la gráfica anterior podemos observar que la mayoría de los sismos ocurridos en los últimos 51 años han sido de magnitud 5.5 y conforme aumenta la magnitud son menos frecuentes, ¿te lo imaginabas?

También te puedes dar cuenta que estos datos presentan una distribución lognormal (la mayor "carga de datos" se encuentra hacia la izquierda o sesgada la izquierda o sesgo positivo)

Del coeficiente de simetría sabemos que si este es mayor que 1, la distribución es asimétrica positiva mientras que si es menor que 1 es asimétrica negativa. Podemos observarlo en la gráfica y calcularlo de la siguiente manera:

[19]:
```
asimetria = stats.skew(datos1['Magnitude'])
asimetria
```

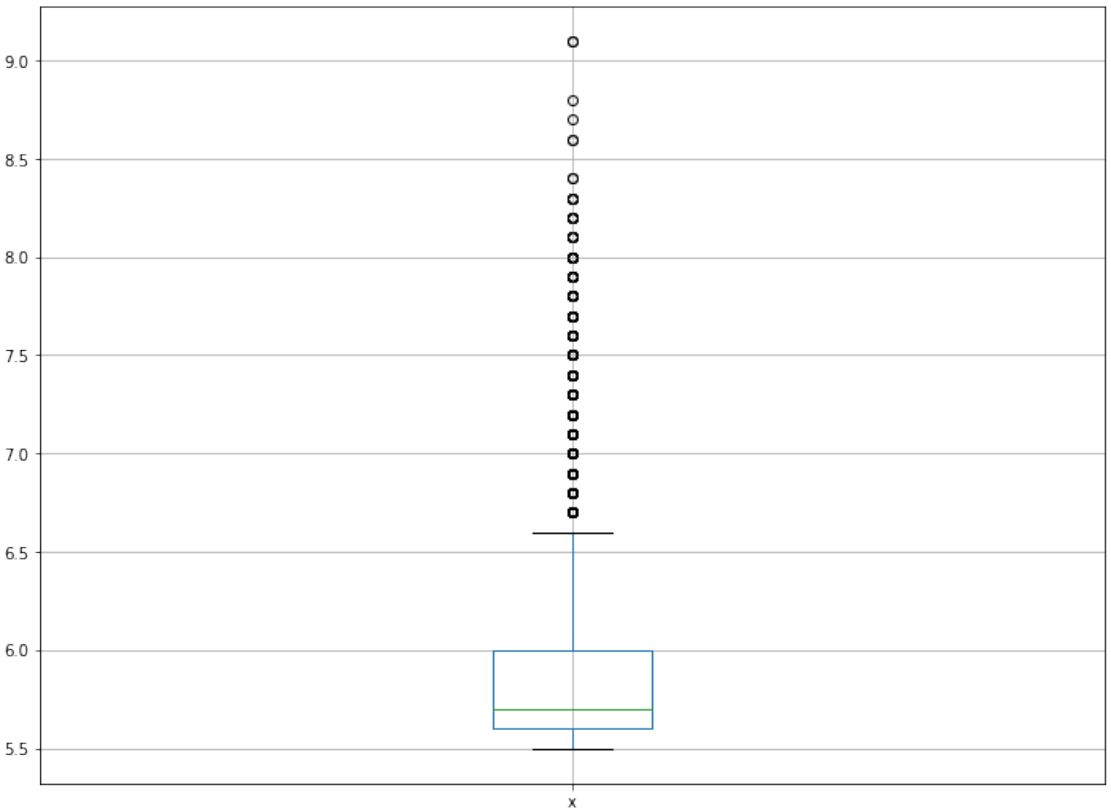
[19]:
```
1.8482272812598326
```

1.5 Graficando *boxplot*

Aunque los histogramas son de utilidad, también podemos hacer uso de un gráfico que nos brinde mayor información: el diagrama de caja, diagrama de bigotes o *boxplot*.

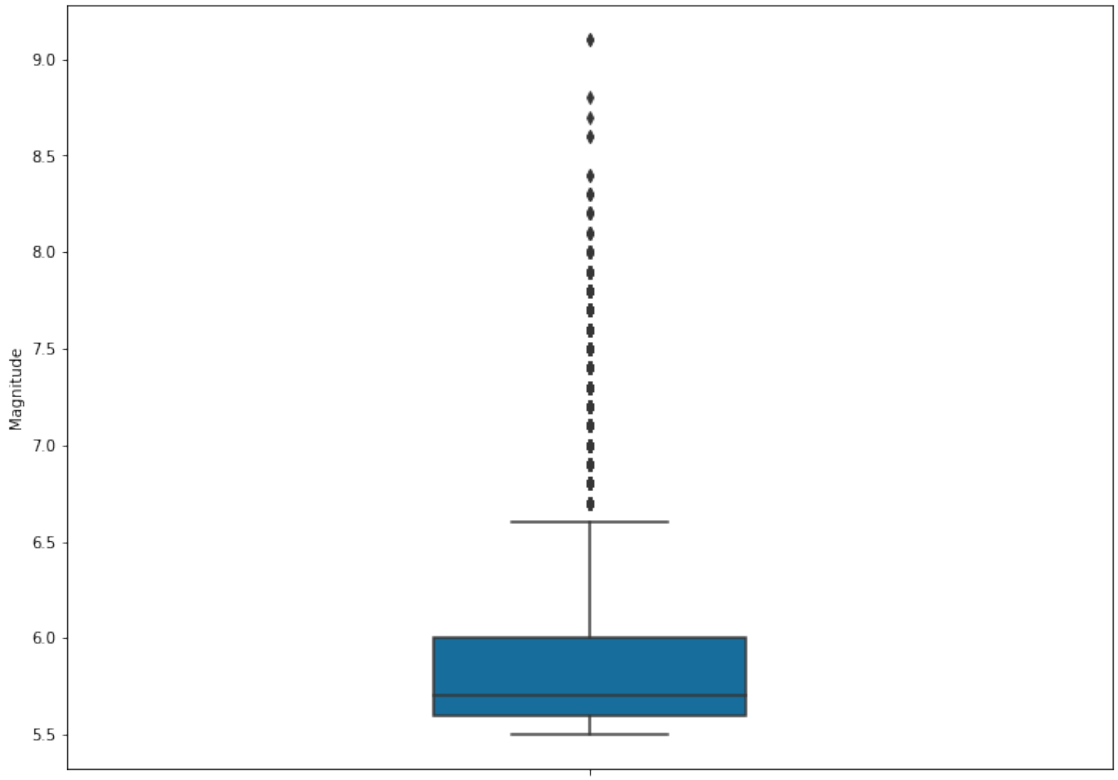
Ventajas: - Proporciona una visión general de la simetría de los datos - Nos permite ver de manera muy rápida los valores atípicos (*outliers*) - Permite observar la dispersión de los puntos respecto a la mediana, los percentiles 25 y 75 y los valores mínimo y máximo

[20]:
```
#Ahora dibujemos un diagrama de caja o de bigote (boxplot) con todos los datos␣
↪de magnitud de los sismos registrados desde 1965 hasta 2016
#Diagrama de caja utilizando Pandas
fig = plt.figure(5, figsize=(12,9))  #definimos el tamaño del gráfico
boxplot1 = pd.DataFrame.boxplot(datos1['Magnitude'])
```



La mediana corresponde con la línea de color verde en la caja. Los límites de las cajas definen los percentiles 25 y 75%. Las líneas azules verticales son los "bigotes" y los puntos que se encuentran por encima del extremo superior corresponden con los valores atípicos o en inglés "outliers".

[21]:
```
#Otra ventaja es que podemos realizar los gráficos mediante diferentes␣
 ↪librerías:
#utilizando seaborn
fig2 = plt.figure(5, figsize=(12,9))
boxplot2 = sns.boxplot(y='Magnitude',
                 data=datos1,
                 width=0.3,
                 palette="colorblind")
```



[22]:
```
#Si quieres utilizar tus gráficos realizados en Jupyter, puedes exportarlos␣
 ↪como jpg de la siguiente manera:
plot_file_name="boxplot.jpg"
#guardamos como formato jpeg
boxplot2.figure.savefig(plot_file_name,
                    format='jpeg',
                    dpi=100)
```

```
#Puedes corroborar que ya está guardado tu gráfico, accediendo a la carpeta en␣
↪la que estás trabajando
```

[23]:
```
# También puedes hacer una tabla de frecuencias
tabla_frec=(datos1
  .groupby("Magnitude")
  .agg(Frequency=("Magnitude", "count")))
tabla_frec.head(15)
```

[23]:

Magnitude	Frequency
5.50	4685
5.51	1
5.52	4
5.53	1
5.54	1
5.55	1
5.58	1
5.60	3967
5.62	1
5.63	1
5.64	2
5.66	2
5.67	2
5.69	1
5.70	3079

¿Qué observas con las frecuencias de los datos? Notaste que la mayoría de los sismos tienen magnitud de 5.50, 5.60, 5.70, etc. ¿Por qué piensas que ocurre esto?

Como ejercicio, puedes analizar la magnitud de los sismos por año y ver si los datos tienen cambios importantes en su distribución año con año.

Referencias:

1. National Earthquake Information Center (NEIC). (n.d.). Usgs.Gov. Recuperado el 25 de septiembre de 2022, de https://www.usgs.gov/natural-hazards/earthquake-hazards/national-earthquake-information-center-neic?qt-science_support_page_related_con=3

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 3. COVARIANZA Y CORRELACIÓN DE DATOS

April 3, 2022

1 EJERCICIO 3. COVARIANZA Y CORRELACIÓN DE DATOS

Autores: Brenda Ortiz Soto, Darío E. Solano Rojas

En este ejercicio calcularemos la covarianza y correlación de datos de tiempo de erupción del geiser Old Faithful localizado en el Parque Nacional Yellowstone en Wayoming, Estados Unidos. Para este ejercicio vamos a usar Pandas, NumPy y Matplotlib. Encuentra el *data set* aquí.

El set de datos está constituido por 2 columnas: *eruptions*, se refiere al tiempo de erupción en minutos y *waiting* es el tiempo de espera hasta la siguiente erupción.

[1]:
```
#Importando las librerías
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
datos = pd.read_csv("e3_datos.csv")
```

[3]:
```
datos.head()
```

[3]:

	eruptions	waiting
0	3.600	79
1	1.800	54
2	3.333	74
3	2.283	62
4	4.533	85

[4]:
```
datos.shape
```

[4]:
```
(272, 2)
```

1.1 Obtención de covarianza y correlación

Como podemos observar el dataset está constituido por 272 lecturas. Ahora procedamos a cuantificar la covarianza y la correlación entre eruptions y waiting. Recurda: El coeficiente de correlación es la versión normalizada de la covarianza.

[5]:
```
#Podemos obtener rápidamente la matriz de covarianza del set de datos
datos.cov()  #dataset.cov()
```

[5]:

	eruptions	waiting
eruptions	1.302728	13.977808
waiting	13.977808	184.823312

[6]:
```
#Y ahora la matriz de correlación:
datos.corr()
```

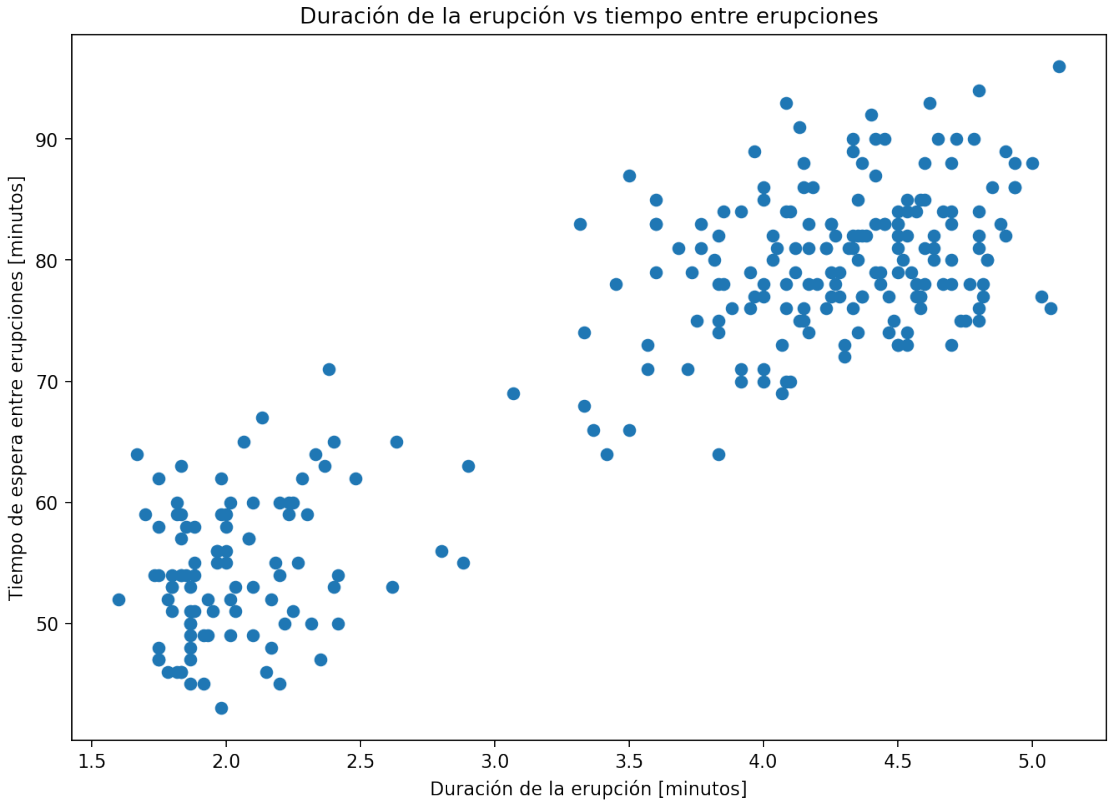
[6]:

	eruptions	waiting
eruptions	1.000000	0.900811
waiting	0.900811	1.000000

1.2 Gráfico de dispersión (*scatter*) y ajuste de modelo lineal

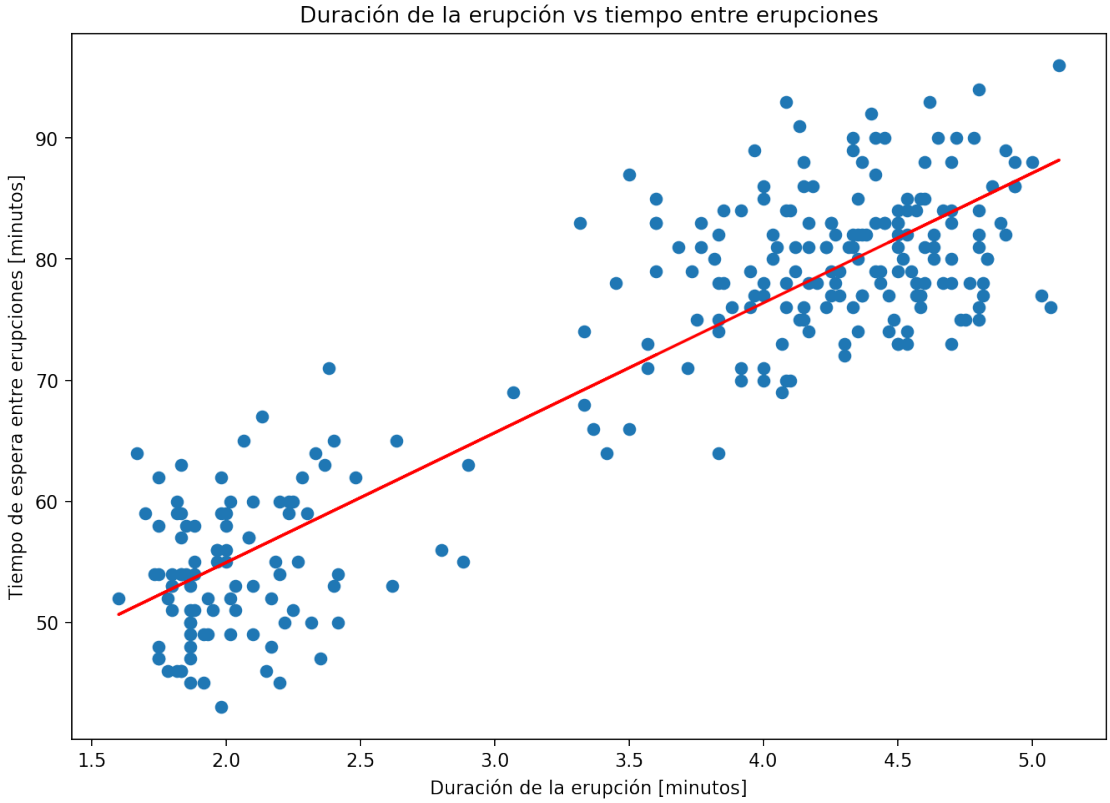
¿Cómo es la correlación entre *eruptions* y *waiting*? Sabemos que el coeficiente de correlación nos indica el grado de dependencia lineal entre dos variables, un valor cercano a 1 nos indica que los datos tienen una correlación lineal mientras que un valor cercano a cero nos indica que los datos no son correlacionables. La correlación entre las dos variables es de 0.90, ¿te imaginas cómo se ven distribuidos los datos en un *scatter*? Espero que sí y si no, ya lo podrás observar en la siguiente gráfica.

[7]:
```
#Ahora grafiquemos un scatterplot o diagrama de dispersión usando pyplot de␣
 ↪matplotlib
plt.figure(figsize=(10, 7), dpi=160) #dpi indica la resolución de la figura
plt.scatter(datos.eruptions,datos.waiting)
plt.title("Duración de la erupción vs tiempo entre erupciones")
plt.xlabel("Duración de la erupción [minutos]")
plt.ylabel("Tiempo de espera entre erupciones [minutos]")

#mostramos el gráfico
plt.show()

#Si quieres guardar la figura, descomenta la siguiente línea
#plt.savefig('erupciones_vs_tiempo_espera.png')
```



[8]:
```
#Ahora calculamos un modelo lineal simple de los datos, repetimos el plot, y le agregamos el modelo lineal

#Calculamos la pendiente(m) y la ordenada en el origen (b)
m, b = np.polyfit(datos.eruptions,datos.waiting, 1)

#Repetimos el gráfico anterior
plt.figure(figsize=(10, 7), dpi=160) #dpi indica la resolución de la figura
plt.scatter(datos.eruptions,datos.waiting)
plt.title("Duración de la erupción vs tiempo entre erupciones")
plt.xlabel("Duración de la erupción [minutos]")
plt.ylabel("Tiempo de espera entre erupciones [minutos]")


#Le agregamos el modelo lineal encima en color rojo
plt.plot(datos.eruptions, m*datos.eruptions + b,'r')

#mostramos el gráfico
plt.show()

#Si quieres guardar la figura, descomenta la siguiente línea
```

```
#plt.savefig('erupciones_vs_tiempo_espera_modelo.png')
```



¿Observas alguna tendencia en los puntos? Recuerdas que el coeficiente de correlación de la dos variables es de 0.90. ¿Te hace sentido el coeficiente de correlación y la tendencia lineal de los datos? ¿Qué interpretación le puedes dar a los datos?

Referencias:

1. (N.d.). Cmu.Edu. Recuperado el 28 de noviembre de 2020 de https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 4. ESTANDARIZACIÓN DE DATOS Y SEMIVARIOGRAMA EN PYTHON

April 3, 2022

1 EJERCICIO 4. ESTANDARIZACIÓN DE DATOS Y SEMIVARIOGRAMA EN PYTHON

Adaptado al español por: Brenda Ortiz Soto

Versión original: Michael Pyrcz

En este ejercicio estandarizaremos datos para posteriormente graficar un semivariograma. Puedes encontrar el *data set* aquí. El *data set* contiene datos de porosidad y permeabilidad de dos facies distintas. Los datos contienen la siguiente información: coordenadas en 'X' (metros), coordenadas en 'Y'(metros), facies (0 si corresponde a lutita o 1 si corresponde a areniscas), así como datos de porosidad (fracción) y permeabilidad (mDarcy).

Este ejercicio es adaptado al español, la versión original es del profesor Michael Pyrcz, quién se dedica a la enseñanza de Geoestadística, Machine Learning y Análisis de datos espaciales, te dejo el enlace a su página personal y te invito a visitar todo el contenido que genera y comparte en Github así como en Youtube. OJO: Michael Pyrcz es el autor, este ejercicio sólo es una adaptación al español de su trabajo.

Para este ejercicio, requerimos importar una libería que aún no hemos utilizado y requiere una previa instalación. Dicha librería es "geostatspy". Encuentra la documentación aquí.

Para su instalación nos dirigimos a "Anaconda", damos click en "Environments", damos click en el triángulo verde (junto a "base(root)"), click en "Open Terminal" y escribimos "pip install geostatspy".

1.1 Importación de librerías

[20]:
```
import geostatspy.GSLIB as GSLIB
import geostatspy.geostats as geostats
```

[21]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1.2 Lectura del set de datos

[24]:
```
df = pd.read_csv("data_var.csv")
```

[25]:
```
df.head()
```

[25]:

	X	Y	Facies	Porosity	Perm	AI
0	100.0	900.0	1.0	0.100187	1.363890	5110.699751
1	100.0	800.0	0.0	0.107947	12.576845	4671.458560
2	100.0	700.0	0.0	0.085357	5.984520	6127.548006
3	100.0	600.0	0.0	0.108460	2.446678	5201.637996
4	100.0	500.0	0.0	0.102468	1.952264	3835.270322

[26]:
```
#Del DataFrame original, haremos dos copias, uno incluirá únicamente las facies␣
↪de arenas y otro incluirá las facies de lutita

df_sand = pd.DataFrame.copy(df[df['Facies'] == 1]).reset_index() #[df['Facies']␣
↪== 1] de la columna Facies copia únicamente los valores iguales a 1
df_shale = pd.DataFrame.copy(df[df['Facies'] == 0]).reset_index()␣
↪#[df['Facies'] == 0] de la columna Facies copia únicamente los valores␣
↪iguales a 0
df_sand.head()                  #previsualiza únicamente los datos con facies =␣
↪1, los cuales están guardados en el nuevo dataframe "df_sand"
```

[26]:

	index	X	Y	Facies	Porosity	Perm	AI
0	0	100.0	900.0	1.0	0.100187	1.363890	5110.699751
1	8	100.0	100.0	1.0	0.137453	5.727603	5823.241783
2	9	200.0	900.0	1.0	0.137062	14.771314	5621.146994
3	10	200.0	800.0	1.0	0.125984	10.675436	4292.700500
4	18	300.0	900.0	1.0	0.111516	27.999817	4183.466773

[27]:
```
df_shale.head() #previsualiza únicamente los datos con facies = 0, los cuales␣
↪están guardados en el nuevo dataframe "df_shale"
```

[27]:

	index	X	Y	Facies	Porosity	Perm	AI
0	1	100.0	800.0	0.0	0.107947	12.576845	4671.458560
1	2	100.0	700.0	0.0	0.085357	5.984520	6127.548006
2	3	100.0	600.0	0.0	0.108460	2.446678	5201.637996
3	4	100.0	500.0	0.0	0.102468	1.952264	3835.270322
4	5	100.0	400.0	0.0	0.110579	3.691908	5295.267191

1.3 Resumen estadístico

[28]:
```
df_sand.describe().transpose()    #resumen estadístico de las facies de arenas
```

[28]:

	count	mean	std	min	25%	\
index	162.0	117.981481	59.058816	0.000000	75.250000	
X	162.0	831.111111	238.857269	50.000000	800.000000	

```
Y          162.0   526.987654  142.707797   100.000000   469.000000
Facies     162.0     1.000000    0.000000     1.000000     1.000000
Porosity   162.0     0.181001    0.037196     0.083842     0.155735
Perm       162.0   293.798999  399.989890     0.381032    31.996547
AI         162.0  3441.701689  969.924695  1844.166880  2748.296631

                50%          75%          max
index     117.500000   157.750000   260.000000
X         945.000000   975.000000  1005.000000
Y         509.000000   549.000000   939.000000
Facies      1.000000     1.000000     1.000000
Porosity    0.194823     0.207754     0.242298
Perm      155.888123   385.177730  2642.999829
AI       3179.596509  3998.567914  6197.834381
```

[29]:
```
df_shale.describe().transpose()          #resumen estadístico de las facies de lutita
```

[29]:
```
           count         mean         std          min          25%  \
index       99.0   149.666667   93.588330     1.000000    41.500000
X           99.0   300.444444  196.365820    40.000000   201.000000
Y           99.0   425.111111  183.665784    29.000000   376.000000
Facies      99.0     0.000000    0.000000     0.000000     0.000000
Porosity    99.0     0.100212    0.014483     0.058871     0.091902
Perm        99.0     3.568463    6.782476     0.033611     0.723788
AI          99.0  5450.493543  728.871517  3595.586977  4897.828718

                50%          75%          max
index     201.000000   225.500000   259.000000
X         231.000000   300.000000   970.000000
Y         416.000000   456.000000   989.000000
Facies      0.000000     0.000000     0.000000
Porosity    0.101987     0.109846     0.141657
Perm        1.530878     3.743835    52.500870
AI       5570.604405  5951.816775  7881.898531
```

Analizando el resumen estadístico de las facies de arenas y las facies de lutita, ¿qué interpretaciones puedes hacer respecto a la porosidad y la permeabilidad?

Se procede a la estandarización de los datos *(normal score)* en conjunto y separados por facies. Esto es requisito para la simulación Gaussiana secuencial, además la transformación Gaussiana nos ayuda con los valores atípicos *(outliers)* y provee variogramas interpretables. Esto lo lograremos con la librería "geostats".

[30]:
```
geostats.nscore          #para observar los parámetros requeridos para la función nscore
```

[30]:
```
<function geostatspy.geostats.nscore(df, vcol, wcol=None, ismooth=False,
dfsmooth=None, smcol=0, smwcol=0)>
```

1.4 Estandarización de datos

[31]:
```
#Transformación gaussiana para todos los datos

#df['Npor'] es la nueva columna que almacenará los datos estandarizados para la␣
 ↪propiedad de porosidad
df['NPor'], tvPor, tnsPor = geostats.nscore(df, 'Porosity') #estandarización␣
 ↪para los valores de la columna de Porosidad de todos los datos

#Transformación gaussiana por facies (arenas)
df_sand['NPor'], tvPorSand, tnsPorSand = geostats.nscore(df_sand, 'Porosity')␣
 ↪#estandarización de los valores de porosidad para las facies de arenas
#Transformación gaussiana por facies (lutitas)
df_shale['NPor'], tvPorShale, tnsPorShale = geostats.nscore(df_shale,␣
 ↪'Porosity') #estandarización de los valores de porosidad para las facies de␣
 ↪lutitas

#Transformación gaussiana para todos los datos
df['NPerm'], tvPermSand, tnsPermSand = geostats.nscore(df, 'Perm') ␣
 ↪#estandarización para los valores de la columna de Permeabilidad de todos␣
 ↪los datos

#Transformación gaussiana por facies (arenas)
df_sand['NPerm'], tvPermSand, tnsPermSand = geostats.nscore(df_sand, 'Perm')␣
 ↪#estandarización de los valores de permeabilidad para las facies de arenas
#Transformación gaussiana por facies (lutitas)
df_shale['NPerm'], tvPermShale, tnsPermShale = geostats.nscore(df_shale,␣
 ↪'Perm')  #estandarización de los valores de permeabilidad para las facies de␣
 ↪lutitas
```

[32]:
```
#Previsualizando los dataframes con los valores estandarizados, como puedes␣
 ↪observar se agregaron dos columnas "Npor" y "Nperm"
df.head()
```

[32]:

	X	Y	Facies	Porosity	Perm	AI	NPor	NPerm
0	100.0	900.0	1.0	0.100187	1.363890	5110.699751	-0.907799	-0.893392
1	100.0	800.0	0.0	0.107947	12.576845	4671.458560	-0.566754	-0.144561
2	100.0	700.0	0.0	0.085357	5.984520	6127.548006	-1.498129	-0.322431
3	100.0	600.0	0.0	0.108460	2.446678	5201.637996	-0.522198	-0.624092
4	100.0	500.0	0.0	0.102468	1.952264	3835.270322	-0.810549	-0.758293

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

[33]:
```
   index      X      Y  Facies  Porosity       Perm           AI      NPor  \
0      0  100.0  900.0     1.0  0.100187   1.363890  5110.699751 -2.158819
1      8  100.0  100.0     1.0  0.137453   5.727603  5823.241783 -0.817609
2      9  200.0  900.0     1.0  0.137062  14.771314  5621.146994 -0.839418
3     10  200.0  800.0     1.0  0.125984  10.675436  4292.700500 -1.031153
4     18  300.0  900.0     1.0  0.111516  27.999817  4183.466773 -1.621370

      NPerm
0 -2.158819
1 -1.468475
2 -0.955141
3 -1.237102
4 -0.694044
```

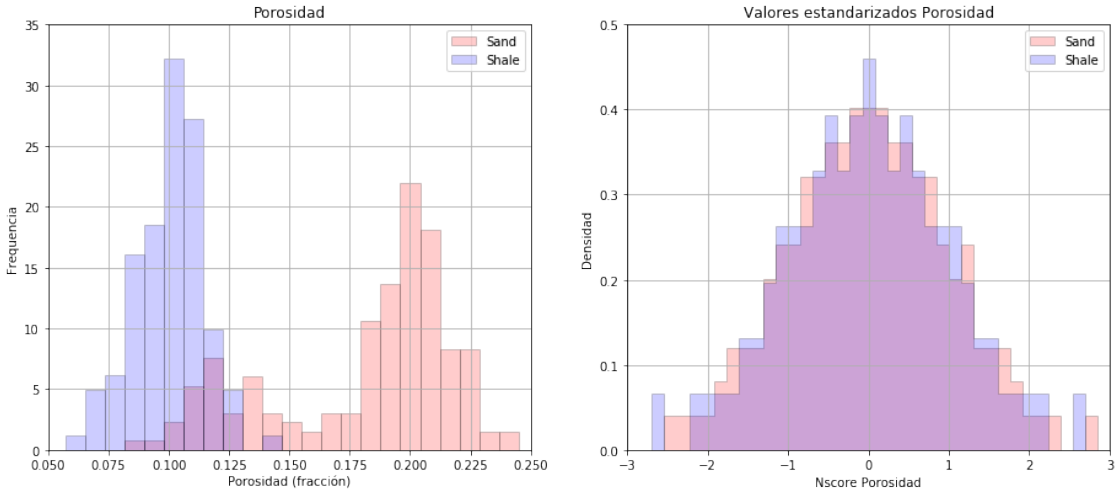
[34]:
```
df_shale.head()
```

[34]:
```
   index      X      Y  Facies  Porosity       Perm           AI      NPor  \
0      1  100.0  800.0     0.0  0.107947  12.576845  4671.458560  0.501298
1      2  100.0  700.0     0.0  0.085357   5.984520  6127.548006 -0.967422
2      3  100.0  600.0     0.0  0.108460   2.446678  5201.637996  0.589456
3      4  100.0  500.0     0.0  0.102468   1.952264  3835.270322  0.050661
4      5  100.0  400.0     0.0  0.110579   3.691908  5295.267191  0.781781

      NPerm
0  1.807354
1  1.051717
2  0.389414
3  0.152506
4  0.650837
```

1.5 Comparación de valores y valores estandarizados

[35]:
```
#Ahora comparemos en dos gráficos los valores de porosidad de ambas facies␣
 ↪contra los valores estandarizados de porosidad de ambas facies
#Si queremos dibujar varios gráficos en un mismo espacio hacemos uso del método␣
 ↪"subplot", de ahora en adelante será de gran utilidad para comparar gráficos

#Fíjate en que el método subplot tiene 3 números en paréntesis, los cuales son␣
 ↪121 y 122
#Imaginemos que estamos dibujando una tabla, donde el primer número es el␣
 ↪número de filas y el segundo número es el número de columnas
#Por lo tanto nuestra "tabla" es de 1x2 (una fila y dos columnas) y el tercer␣
 ↪número nos indicaría la celda o espacio donde se dibuja el subplot

plt.subplot(121)                    #histograma que muestra la porosidad de␣
 ↪ambas facies
#Graficando la porosidad de las facies de arenas en rojo
```

```
plt.hist(df_sand['Porosity'], facecolor='red',bins=np.linspace(0.0,0.
 ↪4,50),alpha=0.2,density=True,edgecolor='black',label='Sand')
#Graficando la porosidad de las facies de lutitas en azul
plt.hist(df_shale['Porosity'], facecolor='blue',bins=np.linspace(0.0,0.
 ↪4,50),alpha=0.2,density=True,edgecolor='black',label = 'Shale')
#delimita los ejes, eje "x" de 0.05 a 0.25 y eje  "y" de 0 a 35.0
plt.xlim([0.05,0.25]); plt.ylim([0,35.0])
#Etiquetando ejes y título del gráfico
plt.xlabel('Porosidad (fracción)'); plt.ylabel('Frequencia'); plt.
 ↪title('Porosidad')
plt.legend(loc='upper right')  #ubicación de la leyenda
plt.grid(True)

plt.subplot(122)                    #histograma que muestra los valores␣
 ↪estandarizados de porosidad de ambas facies
#Graficando los valores estandarizados de porosidad de las facies de arenas en␣
 ↪rojo
plt.hist(df_sand['NPor'], facecolor='red',bins=np.linspace(-3.0,3.
 ↪0,40),histtype="stepfilled",alpha=0.
 ↪2,density=True,cumulative=False,edgecolor='black',label='Sand')
#Graficando los valores estandarizados de porosidad de las facies de lutitas en␣
 ↪azul
plt.hist(df_shale['NPor'], facecolor='blue',bins=np.linspace(-3.0,3.
 ↪0,40),histtype="stepfilled",alpha=0.
 ↪2,density=True,cumulative=False,edgecolor='black',label='Shale')
#delimita los ejes, eje "x" de -3.0 a 3.0 y eje  "y" de 0 a 0.50
plt.xlim([-3.0,3.0]); plt.ylim([0,0.50])
plt.xlabel('Nscore Porosidad'); plt.ylabel('Densidad'); plt.title('Valores␣
 ↪estandarizados Porosidad')
plt.legend(loc='upper right')
plt.grid(True)

plt.subplots_adjust(left=0.0, bottom=0.0, right=2.0, top=1.2, wspace=0.2,␣
 ↪hspace=0.3)
plt.show()
```



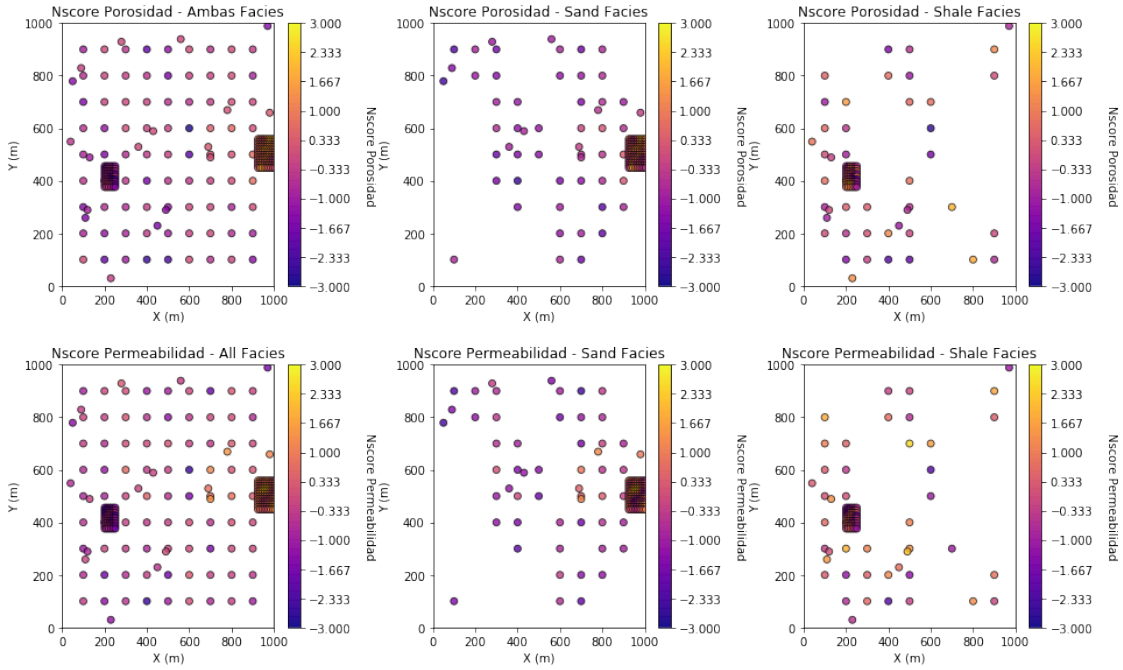
¿Qué diferencias observas entre ambos gráficos? ¿Cuál es la finalidad de estandarizar los datos?

Ahora ubiquemos los datos espacialmente:

1.6 Ubicando los datos espacialmente

[36]:
```
#Grafiquemos subplots, ahora será de 2 filas y 3 columnas.

cmap = plt.cm.plasma                #establecemos los colores del mapa, en este␣
↪caso "plasma". Existen otros como 'viridis', 'inferno', 'magma'.

#Graficando los valores estandarizados de porosidad ambas facies
plt.subplot(231)
GSLIB.locmap_st(df,'X','Y','NPor',0,1000,0,1000,-3,3,'Nscore Porosidad - Ambas␣
↪Facies','X (m)','Y (m)','Nscore Porosidad',cmap)

#Graficando los valores estandarizados de porosidad de las facies de arenas
plt.subplot(232)
GSLIB.locmap_st(df_sand,'X','Y','NPor',0,1000,0,1000,-3,3,'Nscore Porosidad -␣
↪Sand Facies','X (m)','Y (m)','Nscore Porosidad',cmap)

#Graficando los valores estandarizados de porosidad de las facies de lutitas
plt.subplot(233)
GSLIB.locmap_st(df_shale,'X','Y','NPor',0,1000,0,1000,-3,3,'Nscore Porosidad -␣
↪Shale Facies','X (m)','Y (m)','Nscore Porosidad',cmap)

#Graficando los valores estandarizados de permeabilidad de ambas facies
plt.subplot(234)
GSLIB.locmap_st(df,'X','Y','NPerm',0,1000,0,1000,-3,3,'Nscore Permeabilidad -␣
↪All Facies','X (m)','Y (m)','Nscore Permeabilidad',cmap)
```

```
#Graficando los valores estandarizados de permeabilidad de las facies de arenas
plt.subplot(235)
GSLIB.locmap_st(df_sand,'X','Y','NPerm',0,1000,0,1000,-3,3,'Nscore␣
↪Permeabilidad - Sand Facies','X (m)','Y (m)','Nscore Permeabilidad',cmap)

#Graficando los valores estandarizados de permeabilidad de las facies de lutitas
plt.subplot(236)
GSLIB.locmap_st(df_shale,'X','Y','NPerm',0,1000,0,1000,-3,3,'Nscore␣
↪Permeabilidad - Shale Facies','X (m)','Y (m)','Nscore Permeabilidad',cmap)

plt.subplots_adjust(left=0.0, bottom=0.0, right=2.0, top=1.8, wspace=0.4,␣
↪hspace=0.3)
plt.show()
```



Ahora que ya puedes visualizar espacialmente los datos, ¿qué interpretaciones puedes hacer? ¿en qué zonas son más porosas las rocas? ¿en qué zona son más permeables? ¿a qué facies corresponden? ¿siguen alguna distribución las facies de arenas y las de lutitas?

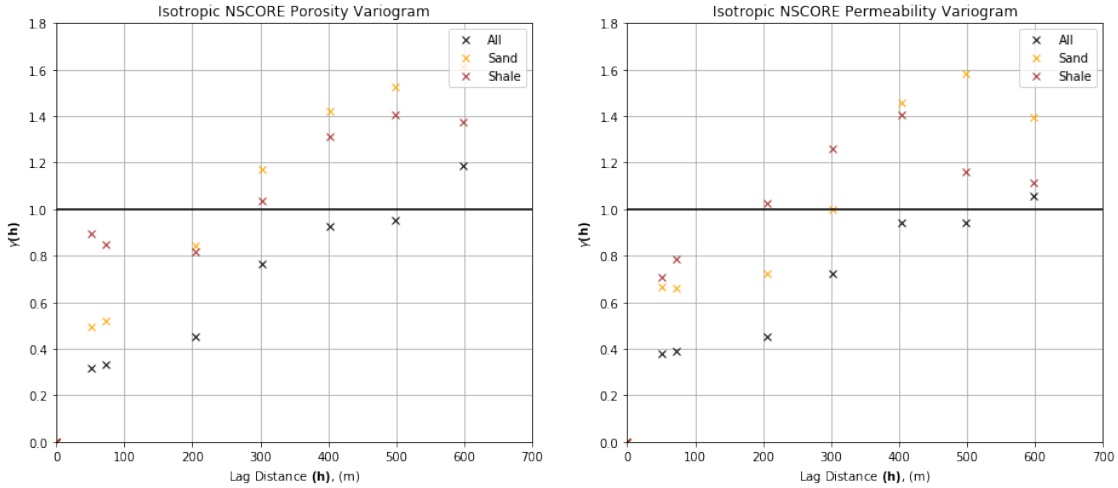
1.7 Semivariograma

[37]:
```
geostats.gamv                    #para observar los parámetros requeridos para␣
↪la función gamv
```

[37]:
```
<function geostatspy.geostats.gamv(df, xcol, ycol, vcol, tmin, tmax, xlag,
xltol, nlag, azm, atol, bandwh, isill)>
```

[38]:
```
#Se definen los valores de los parámetros
tmin = -9999.; tmax = 9999.;
lag_dist = 100.0; lag_tol = 100.0; nlag = 7; bandh = 9999.9; azi = 0; atol = 90.
 ↪0; isill = 1

lag, por_sand_gamma, por_sand_npair = geostats.
 ↪gamv(df_sand,"X","Y","NPor",tmin,tmax,lag_dist,lag_tol,nlag,azi,atol,bandh,isill)
lag, por_shale_gamma, por_shale_npair = geostats.
 ↪gamv(df_shale,"X","Y","NPor",tmin,tmax,lag_dist,lag_tol,nlag,azi,atol,bandh,isill)
lag, por_gamma, por_npair = geostats.
 ↪gamv(df,"X","Y","NPor",tmin,tmax,lag_dist,lag_tol,nlag,azi,atol,bandh,isill)

lag, perm_sand_gamma, perm_sand_npair = geostats.
 ↪gamv(df_sand,"X","Y","NPerm",tmin,tmax,lag_dist,lag_tol,nlag,azi,atol,bandh,isill)
lag, perm_shale_gamma, perm_shale_npair = geostats.
 ↪gamv(df_shale,"X","Y","NPerm",tmin,tmax,lag_dist,lag_tol,nlag,azi,atol,bandh,isill)
lag, perm_gamma, perm_npair = geostats.
 ↪gamv(df,"X","Y","NPerm",tmin,tmax,lag_dist,lag_tol,nlag,azi,atol,bandh,isill)

plt.subplot(121)
plt.plot(lag,por_gamma,'x',color = 'black',label = 'All')
plt.plot(lag,por_sand_gamma,'x',color = 'orange',label = 'Sand')
plt.plot(lag,por_shale_gamma,'x',color = 'brown',label = 'Shale')
plt.plot([0,2000],[1.0,1.0],color = 'black')
plt.xlabel(r'Lag Distance $\bf(h)$, (m)')
plt.ylabel(r'$\gamma \bf(h)$')
plt.title('Isotropic NSCORE Porosity Variogram')
plt.xlim([0,700])
plt.ylim([0,1.8])
plt.legend(loc='upper right')
plt.grid(True)

plt.subplot(122)
plt.plot(lag,perm_gamma,'x',color = 'black',label = 'All')
plt.plot(lag,perm_sand_gamma,'x',color = 'orange',label = 'Sand')
plt.plot(lag,perm_shale_gamma,'x',color = 'brown',label = 'Shale')
plt.plot([0,2000],[1.0,1.0],color = 'black')
plt.xlabel(r'Lag Distance $\bf(h)$, (m)')
plt.ylabel(r'$\gamma \bf(h)$')
plt.title('Isotropic NSCORE Permeability Variogram')
plt.xlim([0,700])
plt.ylim([0,1.8])
plt.legend(loc='upper right')
plt.grid(True)
```

```
plt.subplots_adjust(left=0.0, bottom=0.0, right=2.0, top=1.2, wspace=0.2,␣
↪hspace=0.3)
plt.show()
```



¿Qué interpretación puedes hacer de los variogramas?

Referencias:

1. Pyrcz, M.J., Jo. H., Kupenko, A., Liu, W., Gigliotti, A.E., Salomaki, T., and Santos, J., 2021, GeostatsPy Python Package, PyPI, Python Package Index, https://pypi.org/project/geostatspy/.

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 5. GRAFICANDO PERFIL GEOLÓGICO

April 4, 2022

1 EJERCICIO 5. GRAFICANDO PERFIL GEOLÓGICO

Autora: Brenda Ortiz Soto

En este ejercicio importaremos una nueva librería: plotly. Dicha librería requiere realizar una instalación de la misma; para esto, seguimos los siguientes pasos: Nos dirigimos a "Anaconda", damos click en "Environments", damos click en el triángulo verde (junto a "base(root)"), click en "Open Terminal" y escribimos "pip install plotly==4.12.0" También instalaremos la dependencia "Kaleido" siguiendo los pasos anteriores y escribiendo "pip install -U kaleido".

El *data set* utilizado en este ejercicio corresponde con datos de campo compartidos por el profesor Sergio Salinas Sánchez.

[1]:
```
import pandas as pd          #pandas, nuestra librería amiga para manipular datasets
import plotly.express as px  #importamos la librería plotly, previamente se indica como instalarla
```

[2]:
```
datos=pd.read_csv('e_coord.csv', sep = ';')
#previsualizamos los datos
```

De la previsualización podemos notar 2 aspectos importantes, el nombre de las últimas 4 columnas y los valores nulos. Podríamos continuar con "x mE", "y mN", etc pero para hacer un mejor trabajo y evitar errores futuros en el código, reemplazaremos los nombres de las últimas 4 columnas utilizando la siguiente sintaxis. Si eres muy cuidadoso, podrías omitir este paso, pero recuerda que no somos tan amigos de los espacios, ni de las tildes, o de nombres largos y confusos.

[3]:
```
datos.rename(columns={'x mE':'long','y mN':'lat', 'z' : 'elev', 'join_Lito': 'litologia'}, inplace=True)
datos
```

[3]:

	Punto	Distancia	long	lat	elev	litologia
0	1.0	0.000000	355956.7714	2834366.287	780.0	Lutita-Arenisca
1	2.0	24.000849	355976.7579	2834379.575	780.0	Lutita-Arenisca
2	3.0	48.001697	355996.7444	2834392.864	780.0	Lutita-Arenisca
3	4.0	72.002546	356016.7309	2834406.152	780.0	Lutita-Arenisca
4	5.0	96.003395	356036.7174	2834419.440	780.0	Lutita-Arenisca
..	…	…	…	…	…	…
994	NaN	NaN	NaN	NaN	NaN	NaN

```
995    NaN        NaN          NaN          NaN    NaN              NaN
996    NaN        NaN          NaN          NaN    NaN              NaN
997    NaN        NaN          NaN          NaN    NaN              NaN
998    NaN        NaN          NaN          NaN    NaN              NaN

[999 rows x 6 columns]
```

[4]:
```
#Listo, el nombre de las columnas ya quedó ahora sólo faltan los valores nulos
#Como podemos observar el dataframe tiene valores nulos, para una buen manejo␣
 ↪de los datos es necesario hacer la limpieza de los mismos
#hacemos uso de la siguiente sintaxis para saber cuantos valores son nulos:
datos.isnull().sum(axis = 0)
```

[4]:
```
Punto        421
Distancia    421
long         421
lat          421
elev         421
litologia    421
dtype: int64
```

[5]:
```
#Como puedes darte cuenta, hay 421 filas que no tienen ningún valor
#nulos = datos['Punto'].isnull()
#datos[nulos]
#quita los "#" de las líneas anteriores para visualizar los datos nulos
```

[6]:
```
#Haciendo uso de la siguiente sintexis, excluimos los valores nulos
datoslimpios= datos[datos.Punto.notna()]
datoslimpios

#Como se menciona en los capítulos del material, debemos tener cuidado con el␣
 ↪nombre de las variables, puedes fijarte que en el dataframe
#las coordenadas de "x" y "y" están indicadas como "x mE" y "y mN"␣
 ↪respectivamente, indicándonos la longitud y latitud
#Ese espacio podría provocar algún error en el código si nos olvidamos de él
#Por lo tanto, evitaremos el uso de espacio y en este caso cambiaremos los␣
 ↪nombres, así como tambien el de "join_Lito"

##me marca una advertencia cuando hago esto
```

[6]:
```
     Punto     Distancia         long          lat   elev        litologia
0      1.0      0.000000  355956.7714  2834366.287  780.0  Lutita-Arenisca
1      2.0     24.000849  355976.7579  2834379.575  780.0  Lutita-Arenisca
2      3.0     48.001697  355996.7444  2834392.864  780.0  Lutita-Arenisca
3      4.0     72.002546  356016.7309  2834406.152  780.0  Lutita-Arenisca
4      5.0     96.003395  356036.7174  2834419.440  780.0  Lutita-Arenisca
..     …           …          …            …     …              …
```

```
573  574.0  13752.486290  367409.0464  2841980.502  540.0          aluvion
574  575.0  13776.487140  367429.0329  2841993.791  540.0          aluvion
575  576.0  13800.487990  367449.0195  2842007.079  540.0          aluvion
576  577.0  13824.488840  367469.0060  2842020.367  540.0          aluvion
577  578.0  13824.488840  367469.0060  2842020.367  540.0          aluvion

[578 rows x 6 columns]
```

[7]:
```
datoslimpios
```

[7]:
```
     Punto     Distancia         long          lat   elev        litologia
0      1.0      0.000000  355956.7714  2834366.287  780.0  Lutita-Arenisca
1      2.0     24.000849  355976.7579  2834379.575  780.0  Lutita-Arenisca
2      3.0     48.001697  355996.7444  2834392.864  780.0  Lutita-Arenisca
3      4.0     72.002546  356016.7309  2834406.152  780.0  Lutita-Arenisca
4      5.0     96.003395  356036.7174  2834419.440  780.0  Lutita-Arenisca
..     …           …          …           …    …              …
573  574.0  13752.486290  367409.0464  2841980.502  540.0          aluvion
574  575.0  13776.487140  367429.0329  2841993.791  540.0          aluvion
575  576.0  13800.487990  367449.0195  2842007.079  540.0          aluvion
576  577.0  13824.488840  367469.0060  2842020.367  540.0          aluvion
577  578.0  13824.488840  367469.0060  2842020.367  540.0          aluvion

[578 rows x 6 columns]
```

Plotly nos permite realizar gráficas en 3D, dichos gráficos son de gran utilidad en las Ciencias de la Tierra, ya que podemos ubicarlos espacialmente y podemos asignar diferentes colores, dependiendo litología, valores de permeabilidad, concentraciones de algún mineral, etc.

[8]:
```
fig = px.scatter_3d(datoslimpios, x='long', y='lat', z='elev',␣
 ↪color='litologia', color_continuous_scale=px.colors.sequential.Jet)
fig.update_traces(marker=dict(size=3.0))
fig.show()
```

En la esquina superior derecha del gráfico, puedes observar que existen varios iconos que te permiten hacer zoom al gráfico, descargar el gráfico como png, panear el gráfico, entre otros. Para hacer zoom también puedes hacer "scroll" con el ratón. Si colocas el puntero en el perfil, puedes observar un recuadro con los datos asociados al punto que estas indicando. Como puedes darte cuenta, ¡Plotly es una librería bastante interactiva!

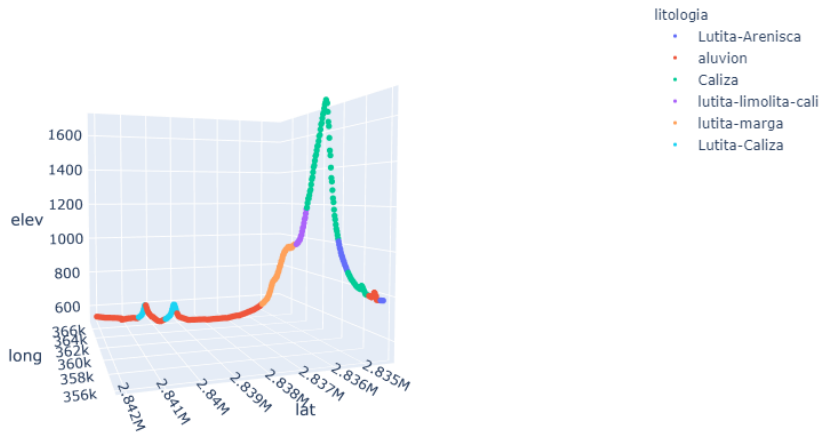


Fig 1. Perfil geológico en plotly

De esta manera, podemos obtener una visualización en 3D de un perfil geológico, claro, sólo visualizamos el perfil topográfico y los distintos contactos entre litologías, pero imagina tener datos de distintos barrenos localizados en algún área de interés y sus valores de concentración mineral, podría servir para hacer interpretaciones sobre las "tendencias de mineralización" o también podrías identificar fácilmente donde existen valores altos y donde los valores "no pagan". ¿Qué otros datos geológicos te imaginas que puedes visualizar en plotly?

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 6. RED SCHMIDT Y RED DE WULFF

April 4, 2022

1 EJERCICIO 6. RED SCHMIDT Y RED DE WULFF

Adaptado por: Brenda Ortiz Soto

Encuentra la versión original aquí.

En este ejercicio, se grafican datos estructurales en la Red de Wulff y Schmidt. Además utilizaremos una nueva librería, llamada mplstereonet. Para la instalación de la misma, nos dirigimos a "Anaconda", damos click en "Environments", damos click en el triángulo verde (junto a "base(root)"), click en "Open Terminal" y escribimos "pip install mplstereonet"

[1]:
```
import matplotlib.pyplot as plt        #importamos ambas librerías
import mplstereonet                    #ojo, previa instalación
```

Con mplstereonet, podemos hacer uso de la red de Wulff y/o de la red de Schmitt, dependiendo de los objetivos de nuestro trabajo. Sólo bastará especificar el parámetro "projection"

projection='equal_angle_stereonet' EN CASO DE utilizar la red de Wulff

projection='equal_area_stereonet' EN CASO DE utilizar la red de Schmitt

Si escribimos: projection='stereonet', por default ploteará los datos sobre una red de Schmitt

A continuación haremos una comparación entre ambas redes:

[2]:
```
fig = plt.figure() #crea nueva figura

#Graficamos Red de Schmidt
ax1 = fig.add_subplot(1,2,1, projection='equal_area_stereonet')

#Graficamos Red de Wulff
ax2 = fig.add_subplot(1,2,2, projection='equal_angle_stereonet')

# Ploteamos los mismos datos en ambas redes con un ciclo "for"
for ax in [ax1, ax2]:
    ax.grid(True) #True para que muestre todos los círculos mayores y menores␣
 ↪en la red
    ax.set_azimuth_ticklabels([])
    ax.plane(315, 20)                  #graficamos el plano 315/20
    ax.line([20, 30, 40, 50], [110, 265, 170,30]) #graficamos las líneas 20/
 ↪110, 30/265, 40/170, 50/030
```

```
ax1.set_title('Áreas iguales (Red de Schmidt)')
ax2.set_title('Ángulos iguales (Red de Wulff)')

# Ajuste para que los gráficos acomoden mejor
fig.subplots_adjust(hspace=0, wspace=0.2, left=0.01, bottom=0.1, right=0.99)

fig.suptitle('Fig 1. Comparación entre red equiareal y red equiangular\n'
             'Mismos datos ploteados', y=0.1)
plt.show()

fig.savefig("comparacion_redes.png") #de esta manera, puedes guardar el gráfico
```

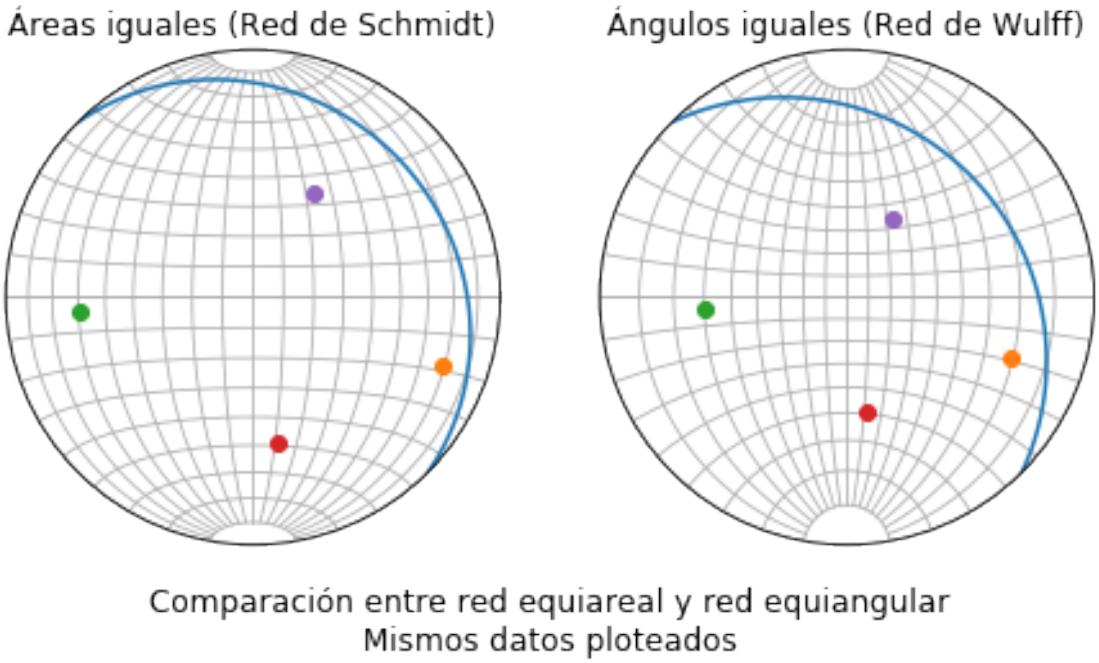


Fig 1. Comparación de redes estereográficas

Referencias:

1. Mplstereonet — mplstereonet 0.6-dev documentation. (n.d.). Readthedocs.Io. Recuperado el 30 de marzo de 2021, de https://mplstereonet.readthedocs.io/en/latest/index.html

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 7. MANIPULACIÓN Y VISUALIZACIÓN DE UNA NUBE DE PUNTOS EN PYTHON

April 4, 2022

1 EJERCICIO 7. MANIPULACIÓN Y VISUALIZACIÓN DE UNA NUBE DE PUNTOS EN PYTHON

Autores: Darío Solano, Brenda Ortiz y Josué García. División de Ingeniería en Ciencias de la Tierra, Facultad de ingeniería, UNAM, 2021. Contacto: dsolano@unam.mx, dario.e.solano@gmail.com

- Notebook disponible para descarga
- Visualización en línea disponible aquí

1.1 Introducción

Las nubes de puntos generadas a partir de técnicas fotogramétricas o de barridos de LiDAR tienen la característica de ser de un volumen considerable. Contrario a la intuición inicial que uno pudiera tener, las nubes de puntos no son un elemento gráfico, sino que son texto, el cual debe ser interpretado para poder generar una visualización. La mayoría de los softwares especializados en nubes de puntos son capaces de realizar una visualización básica de las nubes de puntos. Sin embargo, manipular las nubes para enfocarse en las áreas y atributos de interés de las mismas no siempre es una tarea fácil.

En este ejercicio se aborda el cómo manipular una nube de puntos utilizando herramientas de python, al tiempo que se abordan diferentes consideraciones y técnicas de visualización de las mismas. La nube de puntos que vamos a explorar corresponde a una de las paredes de basalto de la Casa Club del Académico. Dicha pared tiene basaltos de diferentes texturas, sombras, vegetación y cortes hechos para la explotación de la roca como material de construcción. La nube en cuestión se hizo a partir de un levantamiento fotogramétrico con drones y tiene un tamaño aproximado de 1 Gb.

Para la realización de este ejercicio vamos a utilizar numpy, matplotlib y open 3d.

1.2 Importación de librerías y lectura de datos

[1]:
```
#En esta parte importamos las librerías necesarias y definimos los parámetros␣
↪para la visualización de las figuras
#%matplotlib notebook   #descomentar para visualización interactiva
import numpy as np
import matplotlib
```

[2]:
```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
matplotlib.rcParams['figure.figsize'] = [14, 9]
matplotlib.rcParams['figure.dpi'] = 400
```

[3]:
```
#Aquí leemos la nube de puntos. Tú necesitas cambiar esta ruta para leerla en␣
 ↪tu compu ;)
file_data_path='/Users/SOPORTE TI/Downloads/CCA_Oblicuo_15m_Highest.txt'
point_cloud= np.loadtxt(file_data_path,skiprows=1) #Le vamos a dejar que se␣
 ↪salte la primera fila solo por precauci'on ;)
```

1.3 Conociendo la nube de puntos

[4]:
```
type(point_cloud)
```

[4]:
```
numpy.ndarray
```

[5]:
```
print("ndim: ", point_cloud.ndim) #número de dimensiones
print("shape:", point_cloud.shape) #la dimensión en filas x columnas
print("size: ", point_cloud.size) #el número total de elementos
```

```
ndim:  2
shape: (15724620, 9)
size:  141521580
```

[6]:
```
print("\nHay ", point_cloud.shape[0], "puntos en esta nube de puntos de ejemplo!
 ↪\n"  ) #el número total de elementos
```

```
Hay  15724620 puntos en esta nube de puntos de ejemplo!
```

[7]:
```
print(point_cloud[0, :])  # first row of x2
```

```
[ 4.79328470e+05  2.13789256e+06  2.28447830e+03  3.10000000e+01
  3.10000000e+01  2.90000000e+01 -9.26565000e-01  2.80720000e-01
 -2.50347000e-01]
```

Las primeros tres elementos son las coordenadas X, Y, Z Los siguientes tres elementos son los colores RGB que van de 0 a 255 Los siguientes tres elementos son el vector normal unitario de cada punto de la nube

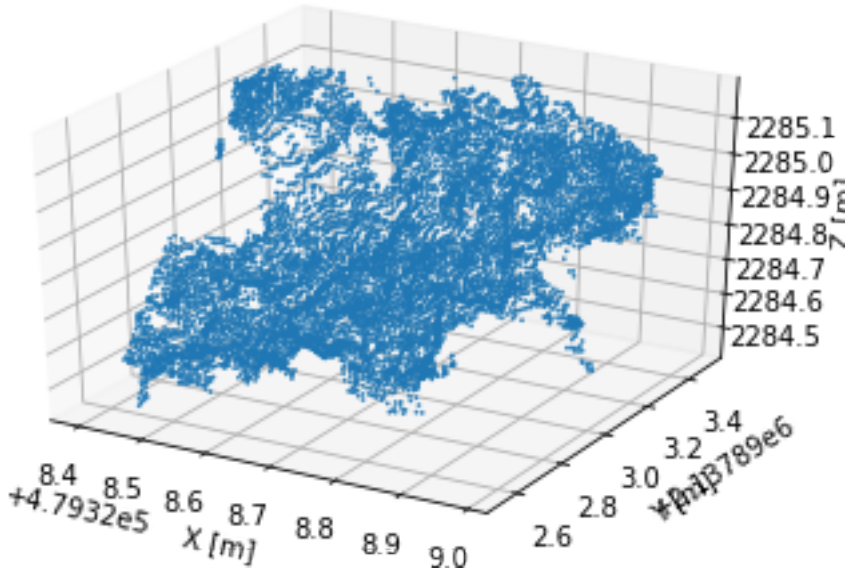
1.4 Visualizaciones con scatter en 3D

1.4.1 10 mil puntos sin color

[8]:
```
xyz=point_cloud[0:10000,:3] #Primero visualizamos 10000 de los ~14 millones de␣
 ↪puntos
rgb=point_cloud[0:10000,3:6]
```

```
ax = plt.axes(projection='3d')#Después creamos unos ejes en 3D para poder␣
 ↪colocar los puntos

#Aquí visualizamos los puntos sin color
ax.scatter(xyz[:,0], xyz[:,1], xyz[:,2],'.', s=0.5, alpha=0.9)

#Aquí ponemos las etiquetas
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
```

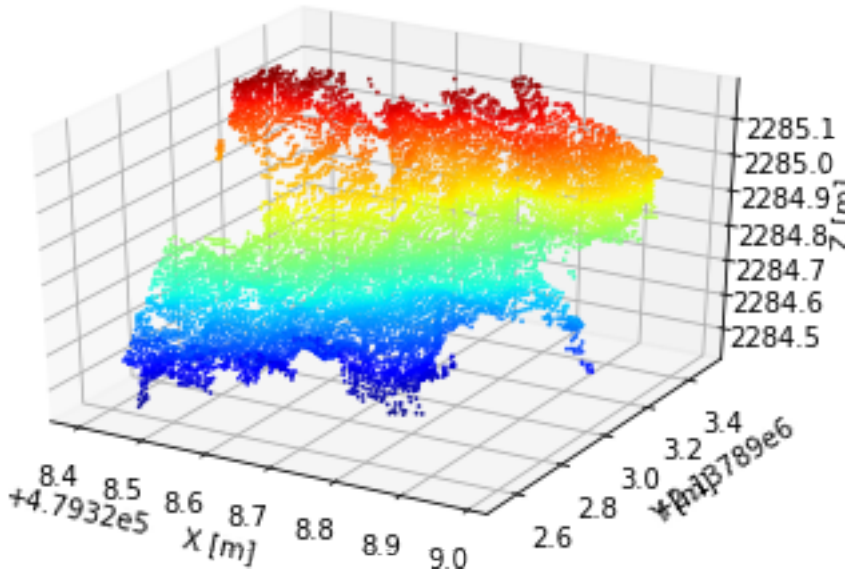
[8]: `Text(0.5, 0, 'Z [m]')`



1.4.2 10 mil puntos con color de acuerdo a la elevación

[9]:
```
ax = plt.axes(projection='3d')

#Aquí visualizamos los puntos usando la elevación como el atributo para darle␣
 ↪color
ax.scatter(xyz[:,0], xyz[:,1], xyz[:,2],'.',c=xyz[:,2],cmap=plt.cm.jet, s=0.5,␣
 ↪alpha=0.9)

ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
```

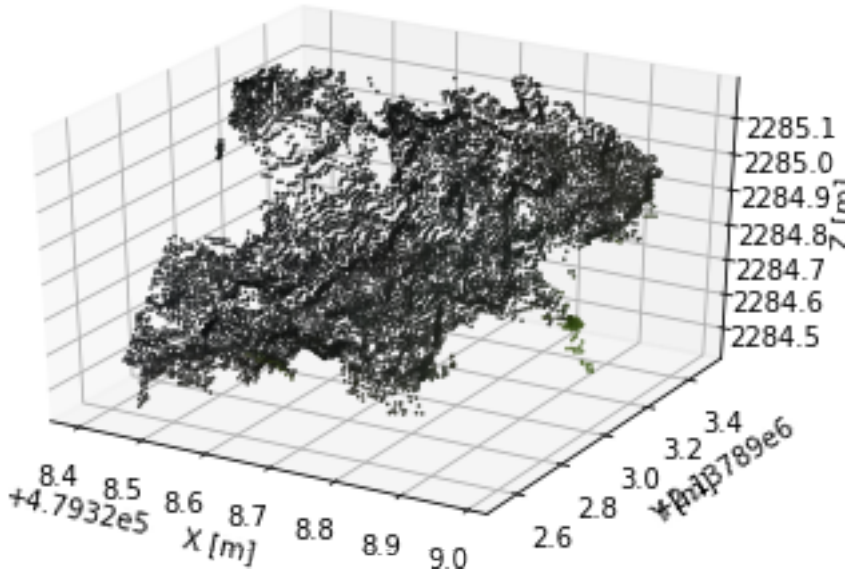
[9]: Text(0.5, 0, 'Z [m]')



1.4.3 10 mil puntos con color real

[10]:
```
ax = plt.axes(projection='3d')

#Aquí visualizamos los puntos utilizando el color a partir del vector RGB
ax.scatter(xyz[:,0], xyz[:,1], xyz[:,2],'.',c=rgb/255, s=0.3, alpha=0.9)

ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
```

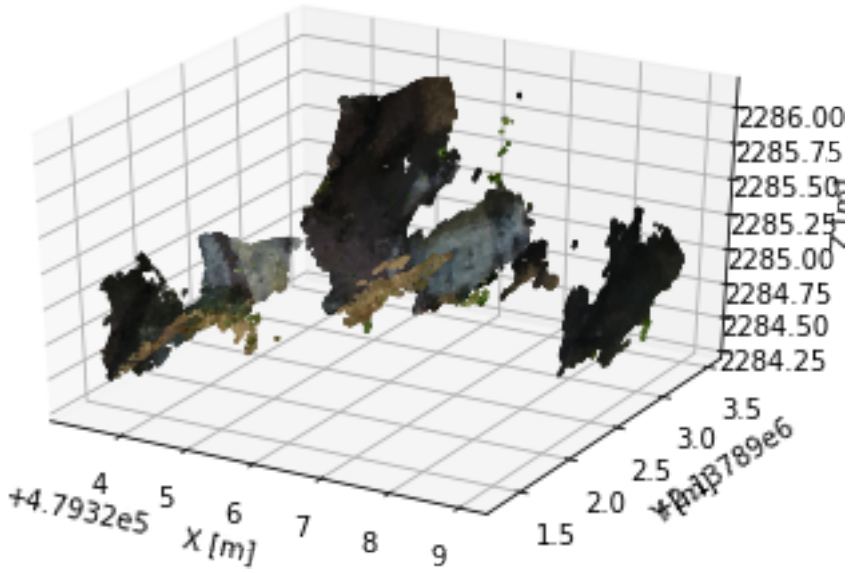
[10]: Text(0.5, 0, 'Z [m]')



1.4.4 100 mil puntos con color real

[11]:
```
#Ahora vamos a usar 100 mil de los ~14 millones de puntos
xyz=point_cloud[0:100000,:3]
rgb=point_cloud[0:100000,3:6]

ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0], xyz[:,1], xyz[:,2],'.',c=rgb/255, s=0.3, alpha=0.9)
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
```

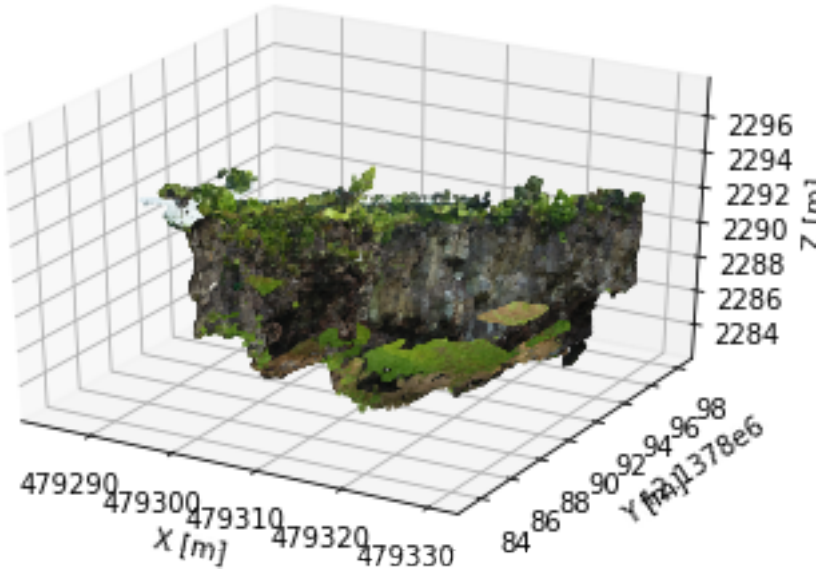
[11]:
```
Text(0.5, 0, 'Z [m]')
```



1.4.5 1 cada 50 puntos de los ~14 millones de puntos

[12]:
```
#Ahora vamos a usar 1 cada 50 de los ~14 millones de puntos
xyz=point_cloud[0::50,:3] #[start:end:step]
rgb=point_cloud[0::50,3:6]

ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0], xyz[:,1], xyz[:,2],'.',c=rgb/255, s=0.3, alpha=0.9)
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
```

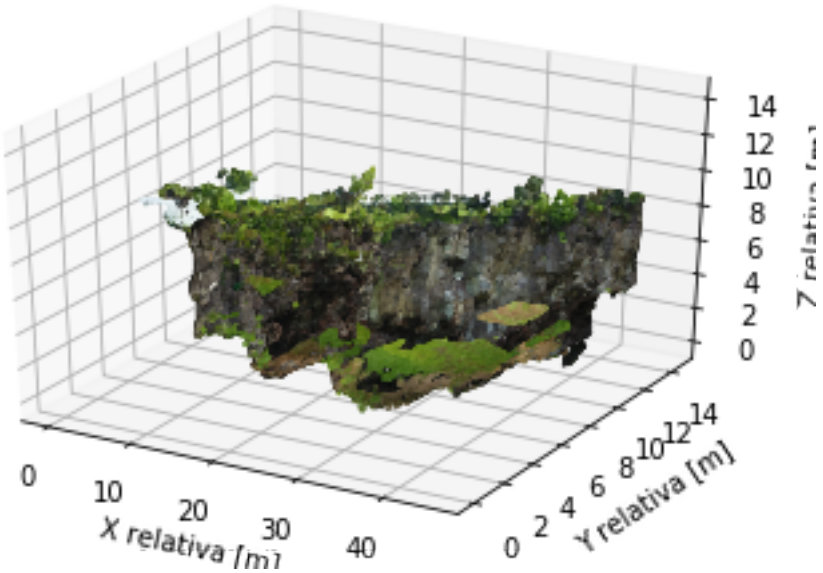
[12]:
```
Text(0.5, 0, 'Z [m]')
```



1.4.6 Obteniendo los valores relativos de X, Y, Z

[13]:
```
ax = plt.axes(projection='3d')

#En esta expresión le restamos el valor medio a los puntos para cada dirección
ax.scatter(xyz[:,0]-np.min(xyz[:,0]),xyz[:,1]-np.min(xyz[:,1]), xyz[:,2]-np.
 ↪min(xyz[:,2]),'.',c=rgb/255, s=0.3, alpha=0.9)

ax.set_xlabel('X relativa [m]')
ax.set_ylabel('Y relativa [m]')
ax.set_zlabel('Z relativa [m]')
```

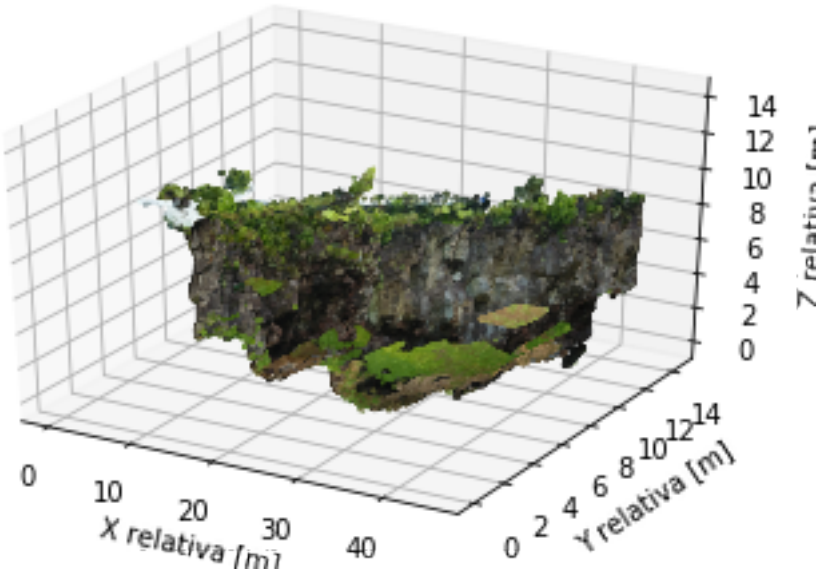
[13]:
```
Text(0.5, 0, 'Z relativa [m]')
```



1.5 Selección del 2% de los puntos usando probabilidad

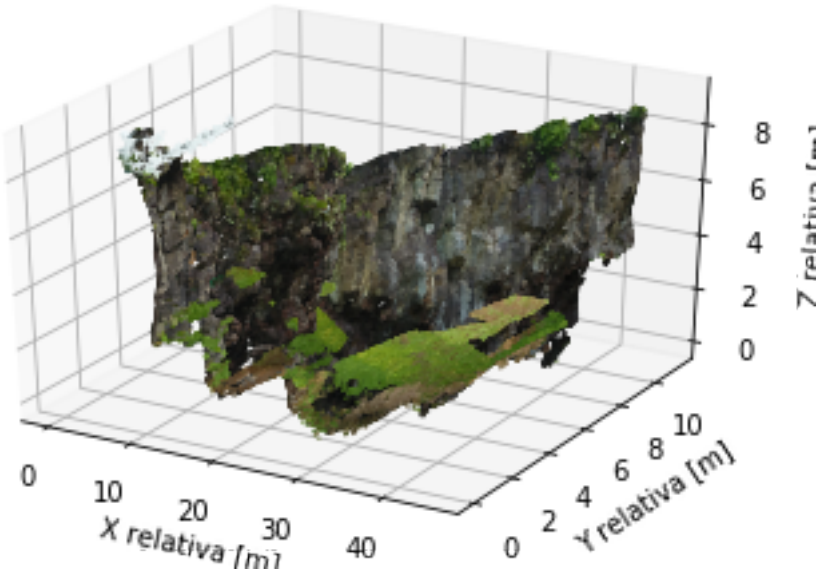
[14]:
```
#Generamos una máscara de 0s y 1s usando probabilidades
#Vamos a usar X% de False y Y% de True
X=0.98
Y=1-X
np.random.seed(22) #Aqui uso el método seed para reproducibilidad del random␣
 ↪generado abajo
mask = np.random.choice([False, True], len(point_cloud), p=[X, Y])
point_cloud1=point_cloud[mask]
```

[15]:
```
xyz=point_cloud1[:,:3]
rgb=point_cloud1[:,3:6]
ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0]-np.min(xyz[:,0]),xyz[:,1]-np.min(xyz[:,1]), xyz[:,2]-np.
 ↪min(xyz[:,2]),'.',c=rgb/255, s=0.3, alpha=0.9)
ax.set_xlabel('X relativa [m]');
ax.set_ylabel('Y relativa [m]');
ax.set_zlabel('Z relativa [m]');
```



1.6 Selección espacial de un subset de datos

[16]:
```
min_Z=np.min(point_cloud1,axis=0)[2]
spatial_query=point_cloud1[abs(point_cloud1[:,2]-min_Z)<9]
xyz=spatial_query[:,:3]
rgb=spatial_query[:,3:6]
ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0]-np.min(xyz[:,0]),xyz[:,1]-np.min(xyz[:,1]), xyz[:,2]-np.
 ↪min(xyz[:,2]),'.',c=rgb/255, s=0.2, alpha=0.9)
ax.set_xlabel('X relativa [m]')
ax.set_ylabel('Y relativa [m]')
ax.set_zlabel('Z relativa [m]')
plt.savefig('CCA_wall.png', bbox_inches='tight')
```



1.7 Para guardar la nube de puntos recortada

[17]:
```
np.save('xyz_subset', xyz)
np.save('rgb_subset', rgb)
```

1.8 Visualización acelerada en 3D usando Open3D

Si nunca habías usado open3d, necesitas instalarlo en tu computadora usando la siguiente celda

[18]:
```
#Descomenta la línea de abajo y ejecuta el !pip para instalar open3d. Si la␣
↪computadora no detecta que ya se instaló, reinicia el kernel o cierra los␣
↪notebooks y Anaconda y vuelve a abrir todo.
#!pip install open3d
```

[19]:
```
import open3d as o3d #Aquí importamos open 3d después de que lo instales.
```

[20]:
```
import numpy as np
from open3d.j_visualizer import JVisualizer
output_path='/Users/SOPORTE TI/Downloads/'
```

[21]:
```
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(point_cloud1[:,:3])
pcd.normals = o3d.utility.Vector3dVector(point_cloud1[:,6:9])
```

[22]:
```
o3d.visualization.draw_geometries([pcd],point_show_normal=False)
```

```
[Open3D WARNING] [ViewControl] SetViewPoint() failed because window height and
width are not set.
```

[23]:
```
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(point_cloud1[:,:3])
pcd.colors = o3d.utility.Vector3dVector(point_cloud1[:,3:6]/255)
pcd.normals = o3d.utility.Vector3dVector(point_cloud1[:,6:9])
```

[24]:
```
o3d.visualization.draw_geometries([pcd],point_show_normal=False)
```

[25]:
```
poisson_mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(pcd,␣
↪depth=8, width=0, scale=1.1, linear_fit=False)[0]
bbox = pcd.get_axis_aligned_bounding_box()
p_mesh_crop = poisson_mesh.crop(bbox)
o3d.io.write_triangle_mesh(output_path+"p_mesh_c.ply", p_mesh_crop)
```

[25]:
```
True
```

[26]:
```
#Esta función fue creada por Florent Poux (ver Ref. 2)
def lod_mesh_export(mesh, lods, extension, path):
    mesh_lods={}
    for i in lods:
        mesh_lod = mesh.simplify_quadric_decimation(i)
        o3d.io.write_triangle_mesh(path+"lod_"+str(i)+extension, mesh_lod)
        mesh_lods[i]=mesh_lod
    print("generation of "+str(i)+" LoD successful")
    return mesh_lods
```

[27]:
```
my_lods = lod_mesh_export(p_mesh_crop, [10000000,5000,1000,100], ".ply",␣
↪output_path)
#Se descargan 4 archivos
```

```
generation of 100 LoD successful
```

[28]:
```
o3d.visualization.draw_geometries([my_lods[10000000]], mesh_show_back_face=True)
```

Se abrirán los modelos en 3D en una ventana externa. Es importante tener paciencia ya que por el tamaño de los datos con los que trabajamos (aprox 1 Gb) pueden tardar un poco más en visualizarse los modelos.

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 8. SEGMENTACIÓN DE NUBES DE PUNTOS BASADA EN COLOR Y K-MEANS EN PYTHON

April 4, 2022

1 EJERCICIO 8. SEGMENTACIÓN DE NUBES DE PUNTOS BASADA EN COLOR y K-MEANS EN PYTHON

Autores: Darío Solano, Brenda Ortiz y Josué García. División de Ingeniería en Ciencias de la Tierra, Facultad de ingeniería, UNAM, 2021. Contacto: dsolano@unam.mx, dario.e.solano@gmail.com

- Notebook disponible para descarga
- Visualización en línea disponible aquí

1.1 Introducción

Las nubes de puntos se generan a partir de fotografías de un objeto observado desde diferentes posiciones conocidas de observación. Como resultado, las nubes de puntos contienen información de posición de un objeto, ya sean relativas o absolutas. Además, las nubes de puntos contienen información de otros atributos como el color.

Al analizar una nube de puntos en el espacio del color, es posible observar agrupamientos que corresponden a condiciones similares de iluminación y composición de las rocas. Por lo tanto, es posible hacer una separación o agrupamiento de puntos de una nube con características de color similares. Del mismo modo, se pueden llegar a separar grupos de puntos que reflejan comportamientos distintivos de las rocas. Sin embargo, es importante siempre tener en mente las limitaciones de las nubes de puntos para no llegar a interpretaciones equivocadas.

El color de cada uno de los puntos de una nube estará determinado por diferentes factores, entre los cuales destacan los materiales de los objetos, el equipo utilizado para adquirir las fotografías y la iluminación. El material de los objetos generalmente no se puede controlar. Las características del equipo tampoco se pueden controlar fácilmente sin invertir económicamente. La iluminación depende de la fuente de luz, pero en general es el sol, y por lo tanto es cambiante a lo largo del día.

En este ejericio vamos a utilizar el atributo de color que cada uno de los puntos de una nubes de tiene asociado. ello vamos a utilizar una técnica de segmentación (o clasificación) conocida como K-means, la cual es una de las herramientas más populares para clasificaciones no supervisadas en machine learning.

Para la realización de este ejercicio vamos a utilizar numpy, matplotlib y sklearn.

1.2 Importación de librerías y lectura de datos

[1]:
```
#%matplotlib notebook
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
matplotlib.rcParams['figure.figsize'] = [14, 9]
matplotlib.rcParams['figure.dpi'] = 400
```

1.3 Conociendo la nube de puntos

[2]:
```
file_data_path1='/Users/dario/Downloads/xyz_subset.npy'
file_data_path2='/Users/dario/Downloads/rgb_subset.npy'
```

[3]:
```
xyz= np.load(file_data_path1)
rgb= np.load(file_data_path2)
```

[4]:
```
print("shape xyz: ", xyz.shape) #la dimensión en filas x columnas de xyz
print("shape rgb:", rgb.shape) #la dimensión en filas x columnas de rgb
```

```
shape xyz:  (260478, 3)
shape rgb: (260478, 3)
```

[5]:
```
print("\nHay ", xyz.shape[0], "puntos en esta nube de puntos de ejemplo!\n"  )␣
↪#el número total de elementos
```

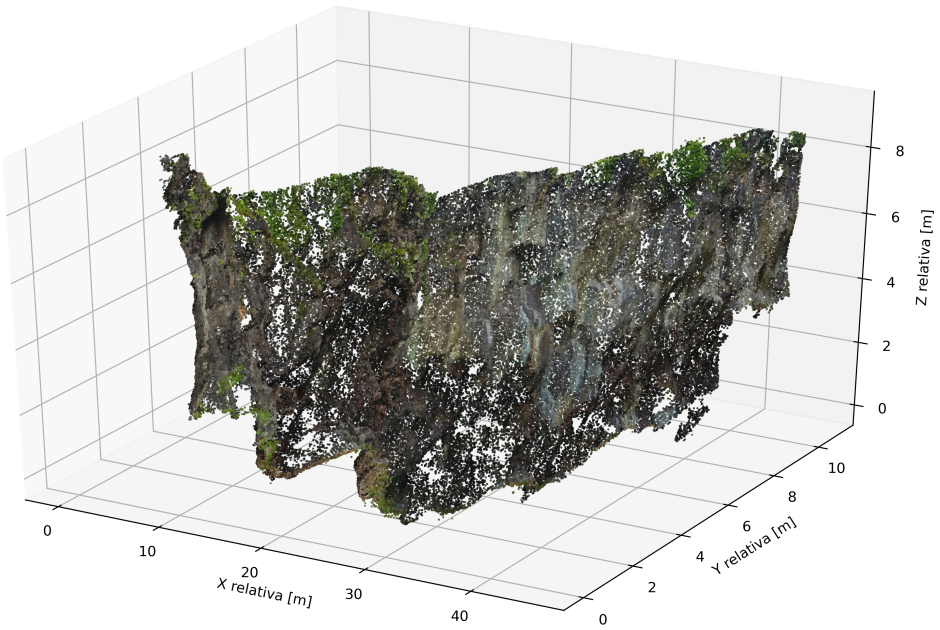
```
Hay  260478 puntos en esta nube de puntos de ejemplo!
```

[6]:
```
print(xyz[0, :])  # first row of x2
```

```
[ 479328.4409 2137892.5978    2284.5623]
```

1.4 Visualizaciones con scatter en 3D

[7]:
```
ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0]-np.min(xyz[:,0]),xyz[:,1]-np.min(xyz[:,1]), xyz[:,2]-np.
↪min(xyz[:,2]),'.',c=rgb/255, s=0.2, alpha=0.9)
ax.set_xlabel('X relativa [m]')
ax.set_ylabel('Y relativa [m]')
ax.set_zlabel('Z relativa [m]')
plt.savefig('CCA_wall.png', bbox_inches='tight')
```

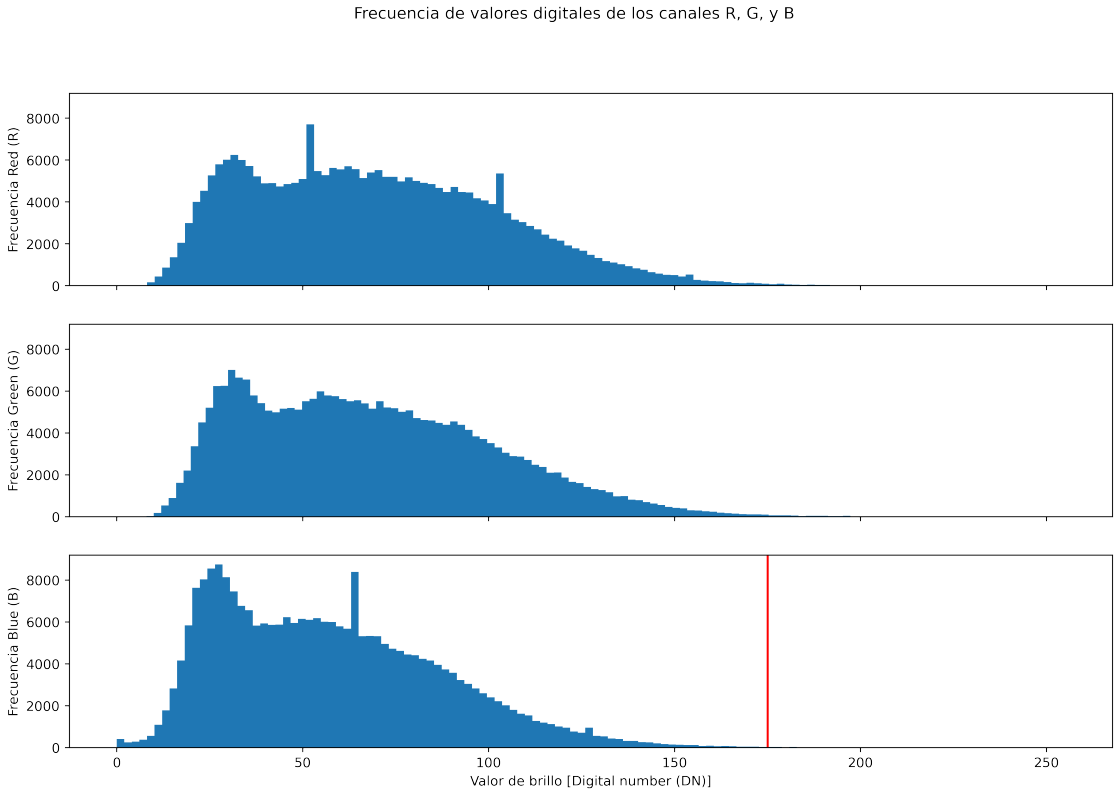


1.5 Análisis básico de los valores en los canales RGB

1.5.1 Histogramas

[8]:
```
fig, axs = plt.subplots(3, sharex=True, sharey=True)
fig.suptitle('Frecuencia de valores digitales de los canales R, G, y B')
axs[0].hist(rgb[:,0], bins = 125);
axs[0].set_ylabel('Frecuencia Red (R)')
axs[1].hist(rgb[:,1], bins = 125);
axs[1].set_ylabel('Frecuencia Green (G)')
axs[2].hist(rgb[:,2], bins = 125);
axs[2].set_ylabel('Frecuencia Blue (B)')
plt.xlabel('Valor de brillo [Digital number (DN)]')
plt.axvline(x=175,color='red')
```

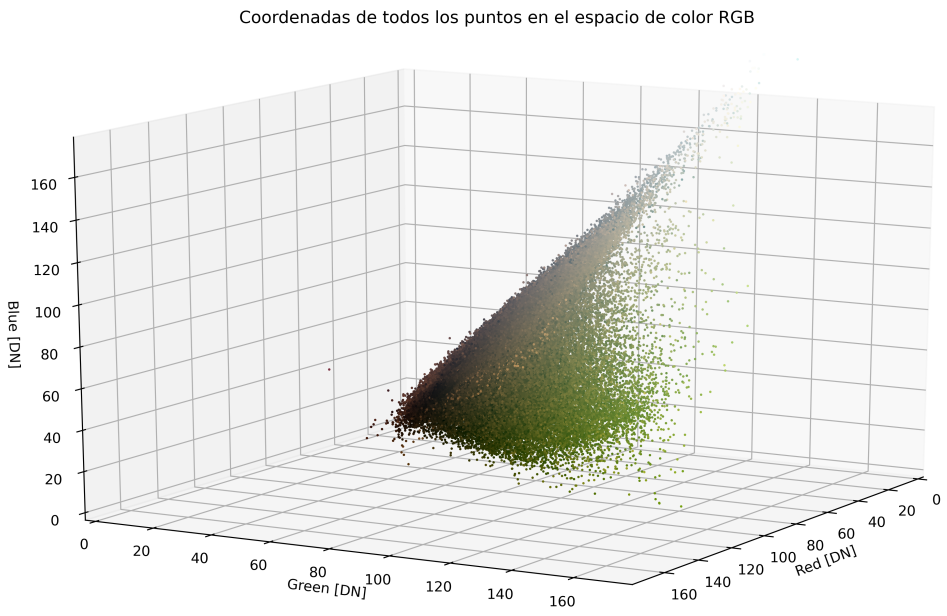
[8]:
```
<matplotlib.lines.Line2D at 0x7f8ec00d9bb0>
```



1.5.2 Espacio de color RGB

[9]:
```
ax = plt.axes(projection='3d')
plt.title('Coordenadas de todos los puntos en el espacio de color RGB')
ax.scatter(rgb[:,0],rgb[:,1], rgb[:,2],'o',c=rgb/255, s=0.5, alpha=1)
ax.view_init(elev=15., azim=30)
ax.set_xlabel('Red [DN]')
ax.set_ylabel('Green [DN]')
ax.set_zlabel('Blue [DN]')
ax.set_xlim([0, 175])
ax.set_ylim([0, 175])
ax.set_zlim([0, 175])
```

[9]:
```
(0.0, 175.0)
```



1.6 Clasificación de la nube de puntos usando el método de K-means

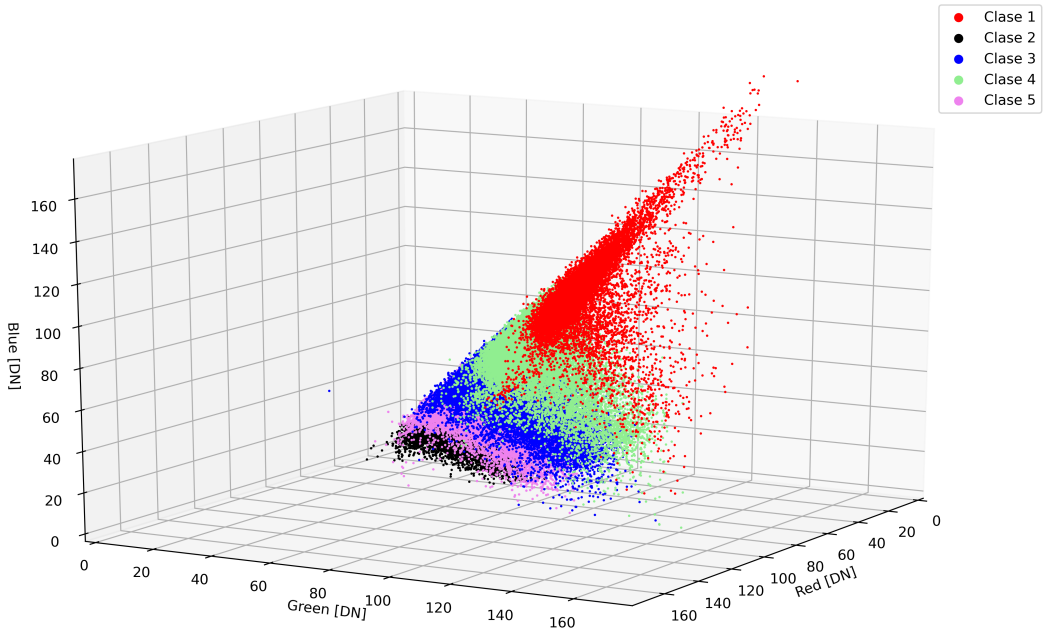
1.6.1 Implementación del método de K-means

[10]:
```
from sklearn.cluster import KMeans

km = KMeans(
    n_clusters=5, init='random',
    n_init=10, max_iter=3000,
    tol=1e-04, random_state=0
)
y_km = km.fit_predict(rgb)
centroids =km.cluster_centers_

print (y_km)
print(y_km.shape)
print(xyz.shape)
print(rgb.shape)
```

```
[1 1 4 … 2 4 2]
(260478,)
(260478, 3)
(260478, 3)
```

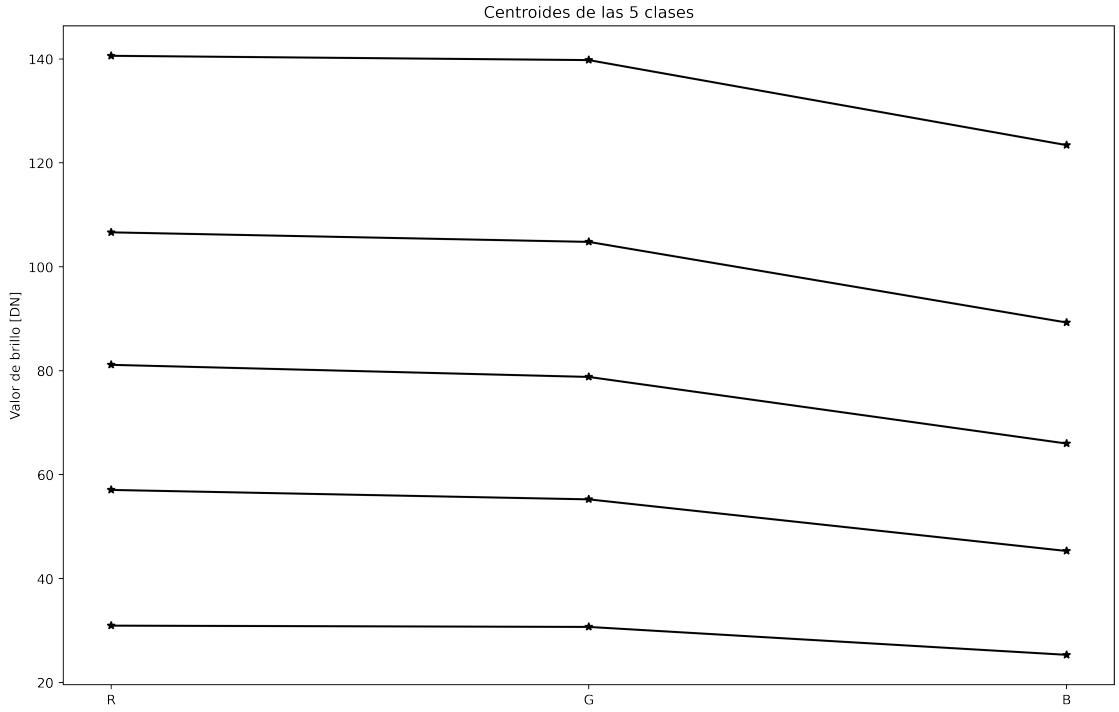
1.6.2 Etiquetado de datos en el espacio de color RGB

[11]:
```
ax = plt.axes(projection='3d')

ax.scatter(rgb[y_km == 0,0],rgb[y_km == 0,1], rgb[y_km == 0,2],'o',c='red', s=0.
 ↪5, alpha=1, label='Clase 1')
ax.scatter(rgb[y_km == 1,0],rgb[y_km == 1,1], rgb[y_km == 1,2],'o',c='black',␣
 ↪s=0.5, alpha=1, label='Clase 2')
ax.scatter(rgb[y_km == 2,0],rgb[y_km == 2,1], rgb[y_km == 2,2],'o',c='blue',␣
 ↪s=0.5, alpha=1, label='Clase 3')
ax.scatter(rgb[y_km == 3,0],rgb[y_km == 3,1], rgb[y_km ==␣
 ↪3,2],'o',c='lightgreen', s=0.5, alpha=1, label='Clase 4')
ax.scatter(rgb[y_km == 4,0],rgb[y_km == 4,1], rgb[y_km == 4,2],'o',c='violet',␣
 ↪s=0.5, alpha=1, label='Clase 5')

ax.view_init(elev=15., azim=30)
ax.set_xlabel('Red [DN]')
ax.set_ylabel('Green [DN]')
ax.set_zlabel('Blue [DN]')
ax.set_xlim([0, 175])
ax.set_ylim([0, 175])
ax.set_zlim([0, 175])
lgnd=ax.legend()
for handle in lgnd.legendHandles:
    handle.set_sizes([30.0])
```



1.6.3 Visualización de los centroides

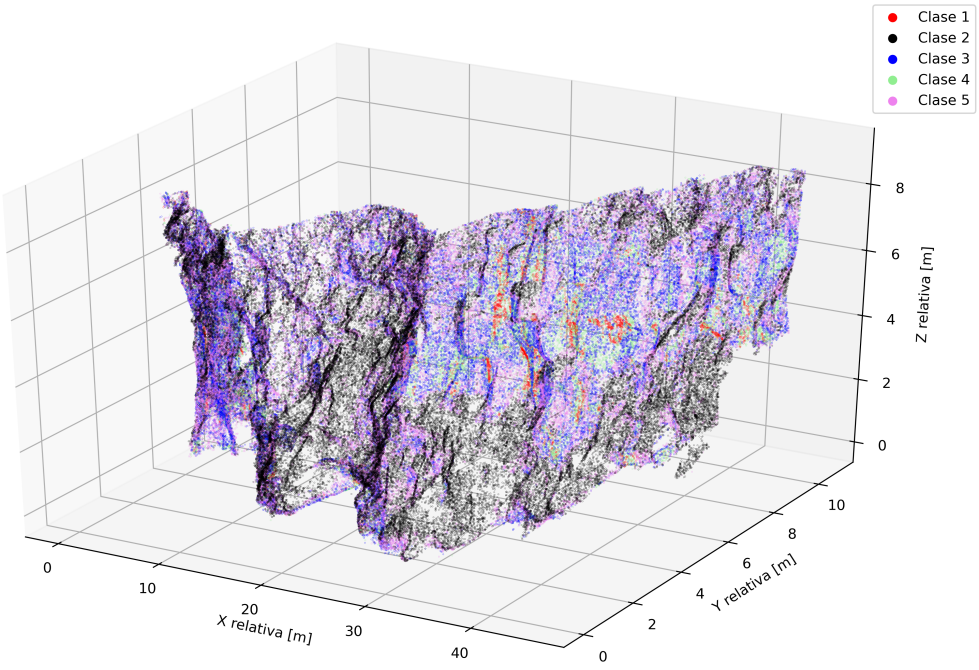
[12]:
```
ax = plt.axes()
ax.plot(centroids.transpose(), c='black', marker='*')
centroids.shape
ax.set_xticks([0, 1, 2]);
ax.set_xticklabels(['R', 'G','B']);
ax.set_ylabel('Valor de brillo [DN]')
plt.title('Centroides de las 5 clases');
```



1.6.4 Etiquetado de datos en el espacio XYZ

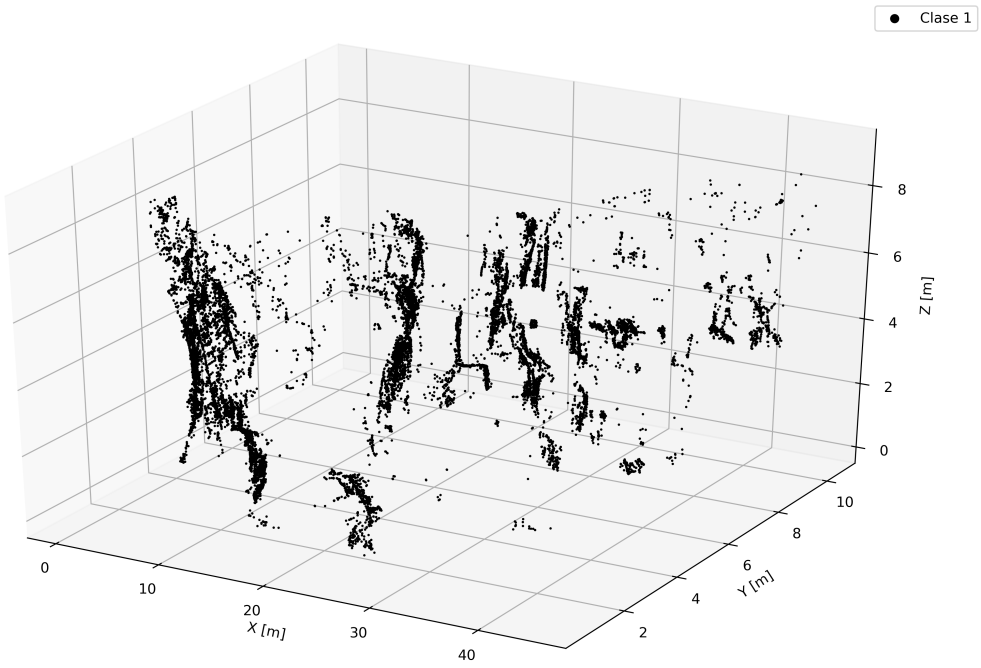
[13]:
```
ax = plt.axes(projection='3d')
ax.scatter(xyz[y_km == 0,0]-np.min(xyz[:,0]),xyz[y_km == 0,1]-np.min(xyz[:,1]),␣
 ↪xyz[y_km == 0,2]-np.min(xyz[:,2]),'.',c='red', s=0.01, alpha=1)
ax.scatter(xyz[y_km == 1,0]-np.min(xyz[:,0]),xyz[y_km == 1,1]-np.min(xyz[:,1]),␣
 ↪xyz[y_km == 1,2]-np.min(xyz[:,2]),'.',c='black', s=0.01, alpha=1)
ax.scatter(xyz[y_km == 2,0]-np.min(xyz[:,0]),xyz[y_km == 2,1]-np.min(xyz[:,1]),␣
 ↪xyz[y_km == 2,2]-np.min(xyz[:,2]),'.',c='blue', s=0.01, alpha=1)
ax.scatter(xyz[y_km == 3,0]-np.min(xyz[:,0]),xyz[y_km == 3,1]-np.min(xyz[:,1]),␣
 ↪xyz[y_km == 3,2]-np.min(xyz[:,2]),'.',c='lightgreen', s=0.01, alpha=1)
```

```
ax.scatter(xyz[y_km == 4,0]-np.min(xyz[:,0]),xyz[y_km == 4,1]-np.min(xyz[:,1]),␣
 ↪xyz[y_km == 4,2]-np.min(xyz[:,2]),'.',c='violet', s=0.01, alpha=1)
ax.set_xlabel('X relativa [m]')
ax.set_ylabel('Y relativa [m]')
ax.set_zlabel('Z relativa [m]')
lgnd=ax.legend(['Clase 1','Clase 2','Clase 3', 'Clase 4','Clase 5'])
for handle in lgnd.legendHandles:
    handle.set_sizes([30.0])
plt.savefig('classified_all.png', bbox_inches='tight')
```



1.6.5 Extracción de una clase de interés

[14]:
```
ax = plt.axes(projection='3d')
cofi=0; #class of interest
ax.scatter(xyz[y_km == cofi,0]-np.min(xyz[:,0]),xyz[y_km == cofi,1]-np.min(xyz[:
 ↪,1]), xyz[y_km == cofi,2]-np.min(xyz[:,2]),'o',c='black', s=0.5, alpha=1)
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
lgnd=ax.legend(['Clase 1'])
for handle in lgnd.legendHandles:
    handle.set_sizes([30.0])
plt.savefig('classified_class2.png', bbox_inches='tight')
```

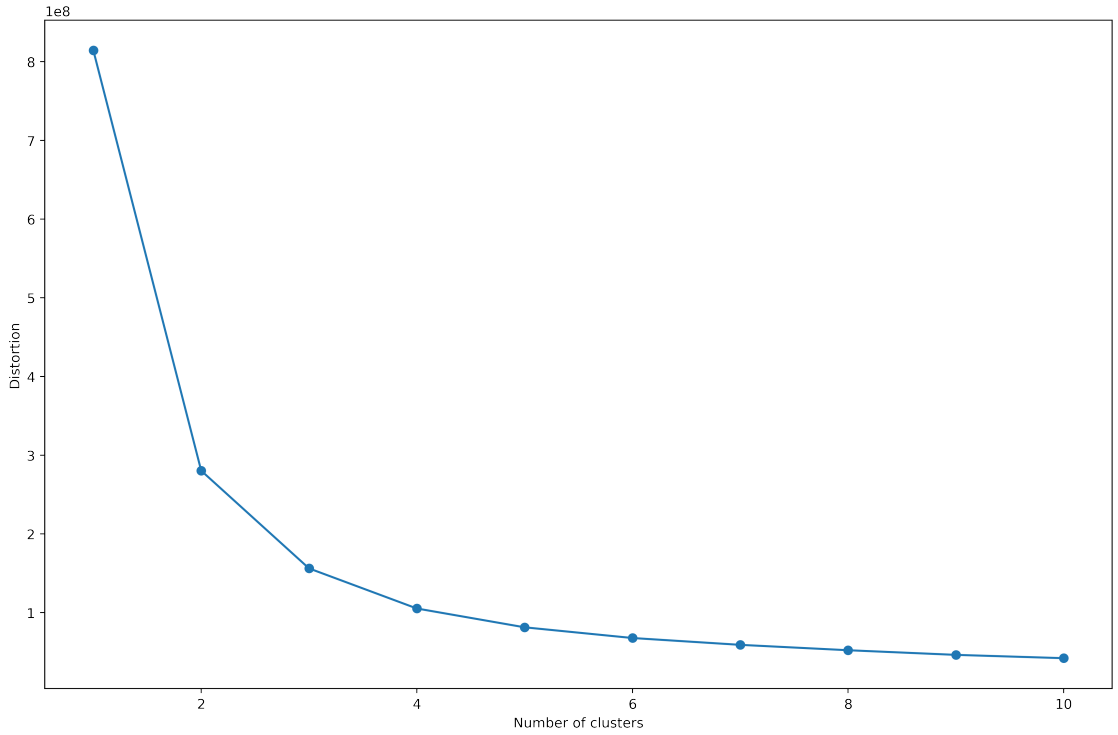


1.7 Para guardar los puntos que corresponden a la Clase 1

[15]:
```
np.save('xyz_subset_class1', xyz[y_km == cofi,:])
```

1.7.1 Determinación de el número adecuado de clases

[16]:
```
# calculate distortion for a range of number of cluster
distortions = []
for i in range(1, 11):
    km = KMeans(
        n_clusters=i, init='random',
        n_init=10, max_iter=300,
        tol=1e-04, random_state=0
    )
    km.fit(rgb)
    distortions.append(km.inertia_)

# plot
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters');
plt.ylabel('Distortion');
```



Referencias:

Canal de Youtube de Darío: https://www.youtube.com/channel/UCmeMXDwmAyZfmgi3ZiXffYw

1. scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. (2021). Retrieved 3 March 2021, from https://scikit-learn.org/
2. Poux, F. (2021). Fundamentals to clustering high-dimensional data (3D point clouds). Retrieved 3 March 2021, from https://towardsdatascience.com/fundamentals-to-clustering-high-dimensional-data-3d-point-clouds-3196ee56f5da
3. Poux, F., Neuville, R., Nys, G., & Billen, R. (2018). 3D Point Cloud Semantic Modelling: Integrated Framework for Indoor Spaces and Furniture. Remote Sensing, 10(9), 1412. doi: 10.3390/rs10091412

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".

EJERCICIO 9. BÚSQUEDA DE PLANOS DE DISCONTINUIDAD A PARTIR DE UNA NUBE DE PUNTOS

April 4, 2022

1 EJERCICIO 9. BÚSQUEDA DE PLANOS DE DISCONTINUIDAD A PARTIR DE UNA NUBE DE PUNTOS

Autores: Darío Solano, Brenda Ortíz y Josué García. División de Ingeniería en Ciencias de la Tierra, Facultad de ingeniería, UNAM, 2021. Contacto: dsolano@unam.mx, dario.e.solano@gmail.com - Notebook disponible para descarga

- Visualización en línea disponible aquí

1.1 Introducción

La aproximación de planos para representar fracturas, fallas, planos de estratificacion, etc., es una práctica común en el ejercicio del profesional de las carreras de Ciencias de la Tierra. Por ejemplo, muchas veces se requiere representar la orientación promedio de una superficie de fracturamiento, lo cual se puede conseguir con la toma sistemática de mediciones usando una brújula de mano y el posterior cálculo de una orientación promedio que represente dicha discontinuidad.

Las nubes de puntos, a su vez, contienen información de posición relativa o absoluta de un objeto en el espacio tridimensional. En el caso de paredes de roca, las nubes de puntos pueden capturan la geometría tridimendional de superficies como superficies de fracturamiento. Evidentemente, sería de utilidad el poder calcular una superficie planar a partir de las posiciones XYZ de los puntos de una nube. Sin embargo, las nubes de puntos continen muchísimos datos (en el orden de millones de puntos), algo que casi puede llegar a ser una desventaja.

Tres puntos en el espacio tridimensional definen un plano. En la geometría euclidiana, los planos definidos por tres puntos pueden expresarse en términos de una ecuación general Ax+By+Cz=D. Tres puntos también pueden utilizarse para definir la ecuación paramétrica en términos de dos vectores directores **u** y **v** y un punto contenido en el plano. Dichos ejemplos representan planos que pueden o no atravesar el origen cartesiano. Un plano también se puede representar en términos de su proyección estereográfica, lo cual sacrifica algo de información, ya que se asume que el plano atraviesa por el origen, pero se requiere el uso de solo dos parámetros para representar un plano: el rumbo y el echado.

En el campo de la geología estructural, existen diferentes métodos para encontrar planos representativos de una manera sistemática utilizando la estadística. La estadística que considera las particulariedades de líneas y planos en una proyección estereográfica se conoce como estadística esférica. Al igual que en la estadística tradicional, los cálculos de medidas de tendencia central -tal como el promedio- se encuentran sujetas a sesgo por distribuciones no normales de los datos

de orientación. Afortunadamente, existe más de un método para calcular orientaciones promedio, tanto en la geología estructural como en otras disciplinas no tan conocidas para la geología, como lo es el campo de la visión de computadora.

En este ejemplo vamos a analizar utilizar las coordenadas cartesianas (XYZ) de nubes de puntos para calcular planos representativos de una discontinuidad en términos de rumbo y echado. Es decir, vamos a llevar la información de las nubes de puntos obtenidas con drones al campo de la geología estructural. Posteriormente, vamos a analiar la prevalencia estadística de alguna familia de planos en particular por medio de la (quasi) transformada de Hough (un método clásico de la visión de computadora). Finalmente, vamos a comparar el cálculo de el plano dominante obtenido con la la (quasi) transofrmada de Hough contra los métodos estadísticos de la geología estructural.

1.2 Importación de librerías y lectura de datos

[1]:
```
#%matplotlib notebook
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
matplotlib.rcParams['figure.figsize'] = [14, 9]
matplotlib.rcParams['figure.dpi'] = 400
```

[2]:
```
file_data_path1='/Users/dario/Downloads/xyz_subset_class1.npy'
xyz=np.load(file_data_path1);
```

1.3 Conociendo la nube de puntos

[3]:
```
print("shape xyz: ", xyz.shape) #la dimensión en filas x columnas de xyz
```

```
shape xyz:  (15903, 3)
```

[4]:
```
print("\nHay ", xyz.shape[0], "puntos en esta nube de puntos de ejemplo!\n"  )␣
 ↪#el número total de elementos
```

```
Hay  15903 puntos en esta nube de puntos de ejemplo!
```

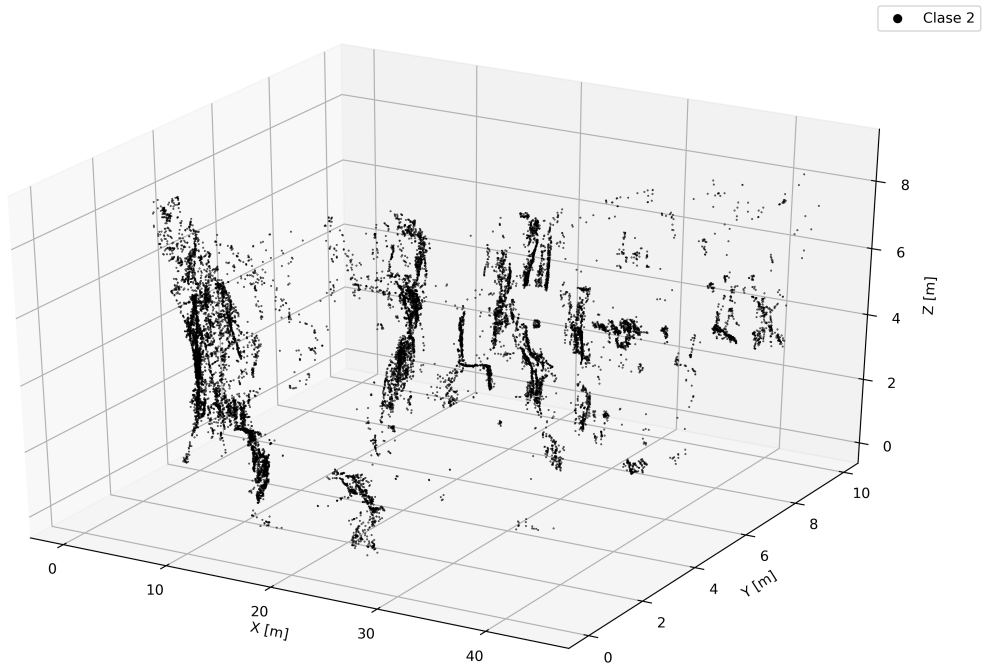
[5]:
```
print(xyz[0, :])  # first row of xyz
```

```
[ 479326.7884 2137893.3171    2285.0361]
```

1.4 Visualizaciones con scatter en 3D de la clase de interés

[6]:
```
ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0]-np.min(xyz[:,0]),xyz[:,1]-np.min(xyz[:,1]), xyz[:,2]-np.
 ↪min(xyz[:,2]),'o',c='black', s=0.1, alpha=1)
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

```
ax.set_zlabel('Z [m]')
lgnd=ax.legend(['Clase 2'])
for handle in lgnd.legendHandles:
    handle.set_sizes([30.0])
```



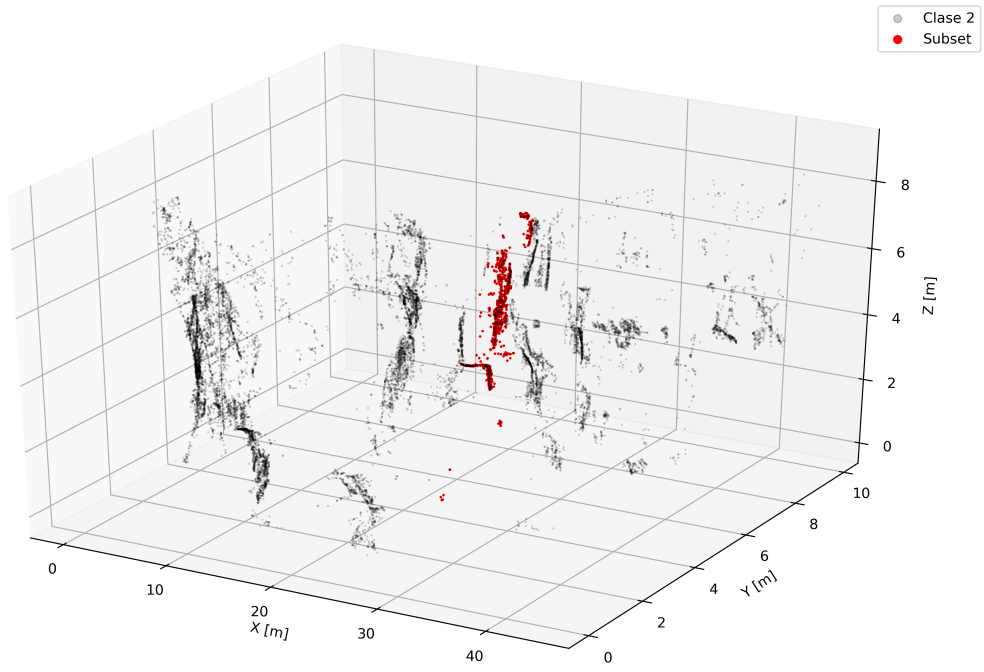
1.5 Selección espacial de un subset de datos de la nube

1.5.1 Recortando una selección de datos usando el eje X

[7]:
```
X1=30;
X2=31.5;
min_X=np.min(xyz,axis=0)[0]
spatial_query=xyz[(abs(xyz[:,0]-min_X)>X1) & (abs(xyz[:,0]-min_X)<X2)]
xyz1=spatial_query;
```

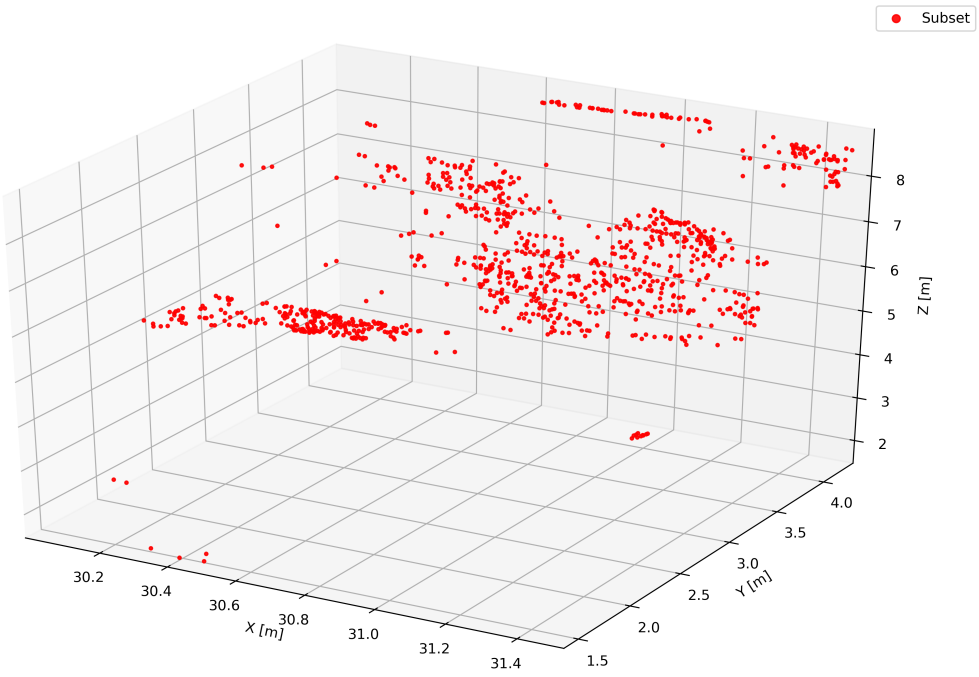
[8]:
```
ax = plt.axes(projection='3d')
ax.scatter(xyz[:,0]-np.min(xyz[:,0]),xyz[:,1]-np.min(xyz[:,1]), xyz[:,2]-np.
 ↪min(xyz[:,2]),'o',c='black', s=0.2, alpha=0.2)
ax.scatter(xyz1[:,0]-np.min(xyz[:,0]),xyz1[:,1]-np.min(xyz[:,1]), xyz1[:,2]-np.
 ↪min(xyz[:,2]),'.',c='red', s=0.7, alpha=1)

ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

```
ax.set_zlabel('Z [m]')
lgnd=ax.legend(['Clase 2', 'Subset'])
for handle in lgnd.legendHandles:
    handle.set_sizes([30.0])
```



1.5.2 Acercamiento a los datos en la dirección X

[9]:
```
ax = plt.axes(projection='3d')
ax.scatter(xyz1[:,0]-np.min(xyz[:,0]),xyz1[:,1]-np.min(xyz[:,1]), xyz1[:,2]-np.
 ↪min(xyz[:,2]),'.',c='red', s=5, alpha=0.9)

ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_zlabel('Z [m]')
ax.set_xlim([np.min(xyz1[:,0]-np.min(xyz[:,0])) , np.max(xyz1[:,0]-np.min(xyz[:
 ↪,0]))])
ax.set_ylim([np.min(xyz1[:,1]-np.min(xyz[:,1])) , np.max(xyz1[:,1]-np.min(xyz[:
 ↪,1]))])
ax.set_zlim([np.min(xyz1[:,2]-np.min(xyz[:,2])) , np.max(xyz1[:,2]-np.min(xyz[:
 ↪,2]))])
lgnd=ax.legend(['Subset'])
for handle in lgnd.legendHandles:
    handle.set_sizes([30.0])
plt.savefig('class2_subset.png', bbox_inches='tight')
```



1.6 Calcular el plano que contiene a los puntos de la nube

1.6.1 Quiasi-Transformada de Hough

La transformada de Hough es un método usado en el campo de la visión de computadora para la detección de curvas. El método se basa en la parametrización de curvas, la formación de curvas de manera aleatoria y el conteo de las que son más populares. En este caso vamos a implementar un método para la detección del plano más común entre tripletas de puntos. Para ello haremos:

1. Selección de un tanto % de los puntos de la clase a estudiar
2. Formación de tríadas (o tripletas) de puntos con combinatoria
3. Cálculo de rumbo y echado del plano formado por cada tripleta usando la regla de la mano derecha
4. Conteo de los planos (en términos de rumbo y echado) que más se repiten

Generando una lista de los índices de xyz1

[10]:
```
idx=np.array(list(range(len(xyz1))));
print(len(idx))
```

```
1021
```

Seleccionando una muestra del Y% de los índices

[11]:
```
#Generamos una máscara de 0s y 1s usando probabilidades
#Vamos a usar X% de False y Y% de True
X=0.98
```

```
Y=1-X
np.random.seed(10)
maskp = np.random.choice([False, True], len(idx), p=[X, Y])
idx_mskd= idx[maskp]
```

Generando tríadas de puntos usando combinatoria

[12]:
```
import itertools
c = list(itertools.combinations(idx_mskd, 3))
```

[13]:
```
print(len(c))
```

969

[14]:
```
print(c[:][0:3])
```

[(56, 248, 327), (56, 248, 395), (56, 248, 435)]

Función que calcula rumbo y echado con regla de la mano derecha a partir de tres puntos en el espacio XYZ

[15]:
```
#Funcion para calcular rumbo y echado de un planbo a partir de tres puntos en␣
 ↪coordenadas cartesianas.
#La autoría de esta función es de Tyler Pubben
#Disponible para descarga en: https://github.com/tpubben/StructureCalculations/
 ↪blob/master/strike_dip.py

import math

# The appropriate input for this function is a list of tuples in the format
# [(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)]
def calc_strikedip(pts):
    ptA, ptB, ptC = pts[0], pts[1], pts[2]
    x1, y1, z1 = float(ptA[0]), float(ptA[1]), float(ptA[2])
    x2, y2, z2 = float(ptB[0]), float(ptB[1]), float(ptB[2])
    x3, y3, z3 = float(ptC[0]), float(ptC[1]), float(ptC[2])


    u1 = float(((y1 - y2) * (z3 - z2) - (y3 - y2) * (z1 - z2)))
    u2 = float((-((x1 - x2) * (z3 - z2) - (x3 - x2) * (z1 - z2))))
    u3 = float(((x1 - x2) * (y3 - y2) - (x3 - x2) * (y1 - y2)))

    '''
    Calculate pseudo eastings and northings from origin
    these are actually coordinates of a new point that represents
    the normal from the plane's origin defined as (0,0,0).

    If the z value (u3) is above the plane we first reverse the easting
    then we check if the z value (u3) is below the plane, if so
```

```
    we reverse the northing.

    This is to satisfy the right hand rule in geology where dip is always
    to the right if looking down strike.
    '''
    if u3 < 0:
        easting = u2
    else:
        easting = -u2

    if u3 > 0:
        northing = u1
    else:
        northing = -u1

    if easting >= 0:
        partA_strike = math.pow(easting, 2) + math.pow(northing, 2)
        strike = math.degrees(math.acos(northing / math.sqrt(partA_strike)))
    else:
        partA_strike = northing / math.sqrt(math.pow(easting, 2) + math.
 ↪pow(northing, 2))
        strike = math.degrees(2 * math.pi - math.acos(partA_strike))

    # determine dip
#    print(strike, 'strike')
    part1_dip = math.sqrt(math.pow(u2, 2) + math.pow(u1, 2))
    part2_dip = math.sqrt(math.pow(u1,2) + math.pow(u2,2) + math.pow(u3,2))
    dip = math.degrees(math.asin(part1_dip / part2_dip))

    return strike, dip
```

Calculando los rumbos y echados de los planos generados por las tríadas de puntos

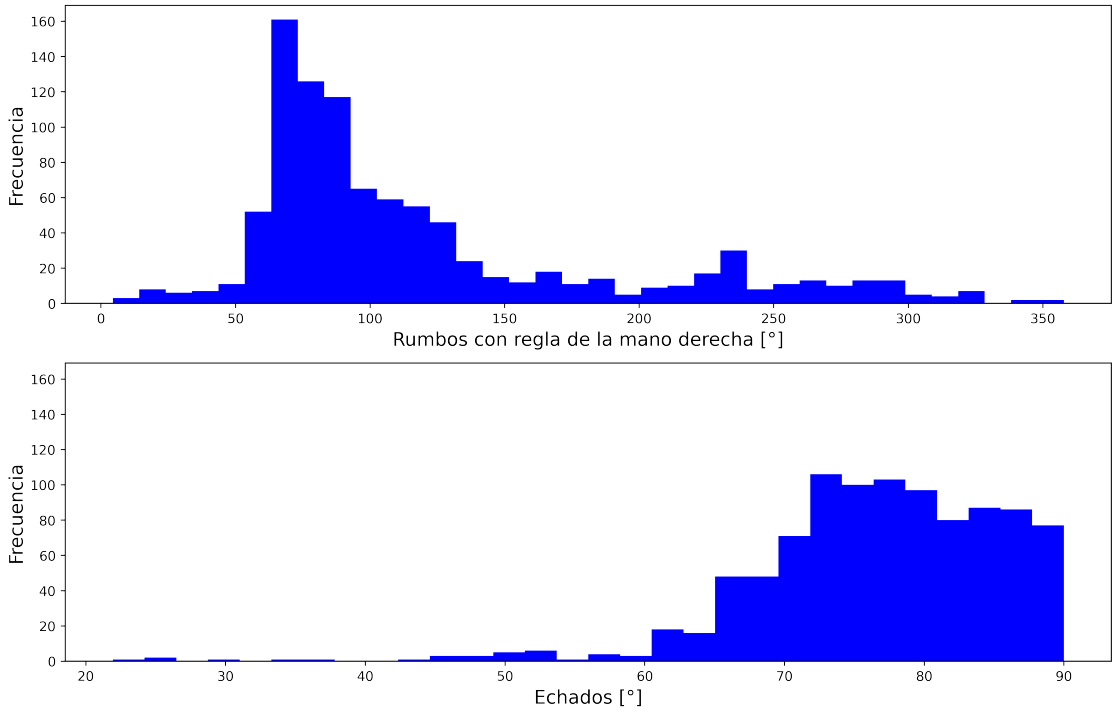
[16]:
```
strikes=[]
dips=[]
for ii in list(range(len(c))):

    strike, dip=calc_strikedip(list([list(xyz1[c[ii][0],:]),␣
 ↪list(xyz1[c[ii][1],:]), list(xyz1[c[ii][2],:])]))
    strikes.append(strike)
    dips.append(dip)
```

Ploteando los rumbos y echados de los planos generados por las tríadas de puntos

[17]:
```
fig, axs = plt.subplots(2, sharex=False, sharey=True)
fig.suptitle('Frecuencia de los rumbos y echados de planos formados a partir de␣
 ↪tríadas de puntos',y=1.05, fontsize=15)
axs[0].hist(strikes,bins = 36, color='blue');
```

```
axs[0].set_ylabel('Frecuencia',Fontsize=14)
axs[0].set_xlabel('Rumbos con regla de la mano derecha [°]',Fontsize=14)
axs[1].hist(dips,bins = 30, color='blue');
axs[1].set_ylabel('Frecuencia',Fontsize=14)
axs[1].set_xlabel('Echados [°]',Fontsize=14);
#fig.tight_layout()
```

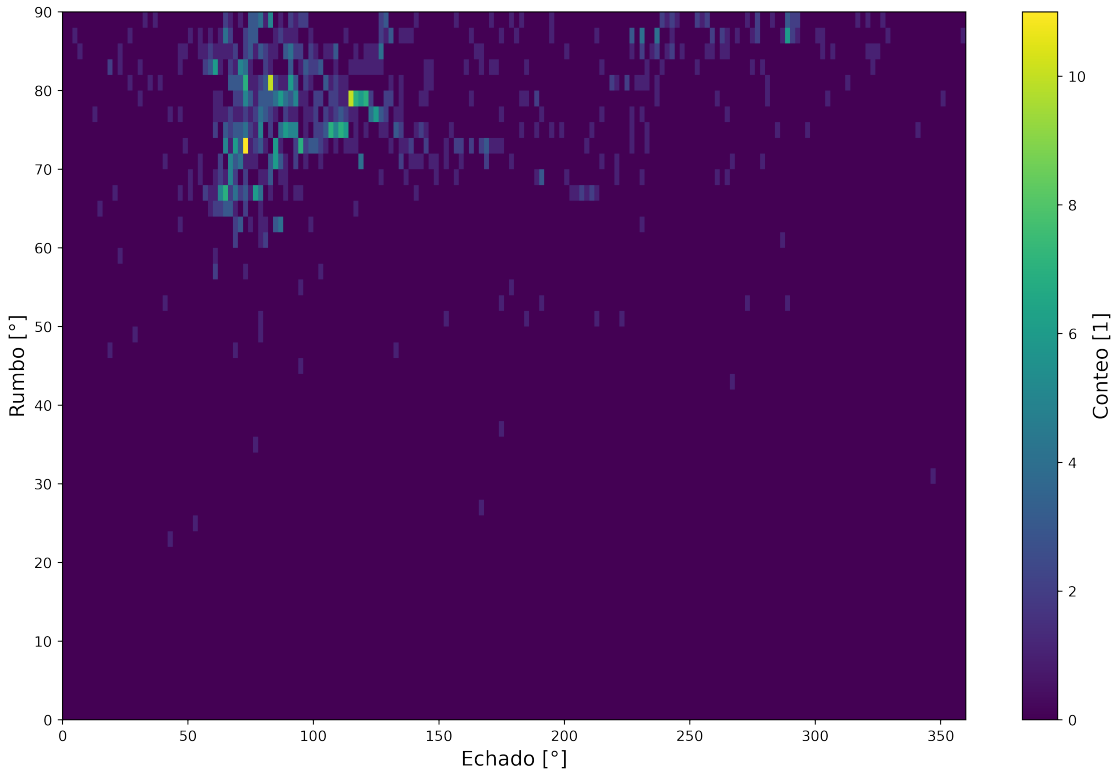
Frecuencia de los rumbos y echados de planos formados a partir de tríadas de puntos



Generando una matriz de conteo

[18]:
```
def idx_closest(list, Number):
    aux = []
    for valor in list:
        aux.append(abs(Number-valor))
    return aux.index(min(aux))

#create empty 2d arrays
step_rumbos=2
step_echados=2
rumbos_grid=list(range(0,360+1,step_rumbos))
echados_grid=list(range(0,90+1,step_echados))
```

```
RU_EC=np.zeros((len(echados_grid),len(rumbos_grid))) #orden matricial

for ii in range(len(strikes)):
    idx_rumbos=idx_closest(rumbos_grid,round(strikes[ii]))
    idx_echados=idx_closest(echados_grid,round(dips[ii]))
    RU_EC[idx_echados,idx_rumbos]=RU_EC[idx_echados,idx_rumbos]+1 #orden␣
 ↪matricial


Ri, Ei = np.meshgrid(rumbos_grid, echados_grid) #orden cartesiano

# Make the plot
plt.pcolormesh( Ri, Ei, RU_EC)  #orden cartesiano
plt.xlabel('Echado [°]', Fontsize=14);
plt.ylabel('Rumbo [°]', Fontsize=14);
cbar=plt.colorbar();
cbar.set_label('Conteo [1]', Fontsize=14)
```



[19]:
```
maximum = np.max(RU_EC)
index_of_maximum = np.where(RU_EC == maximum)
echado_most_counts=Ei[list(index_of_maximum[0])[0],␣
 ↪list(index_of_maximum[1])[0]]+ step_echados/2
```

```
rumbo_most_counts=Ri[list(index_of_maximum[0])[0],␣
 ↪list(index_of_maximum[1])[0]]+ step_rumbos/2
print("El plano que se puede generar el mayor número de veces con las tripletas␣
 ↪de puntos es","Rumbo", rumbo_most_counts,"Echado", echado_most_counts,␣
 ↪sep='\n')
```

```
El plano que se puede generar el mayor número de veces con las tripletas de
puntos es
Rumbo
73.0
Echado
73.0
```

1.7 Usando mplstereonet para representar los datos en la red estereográfica

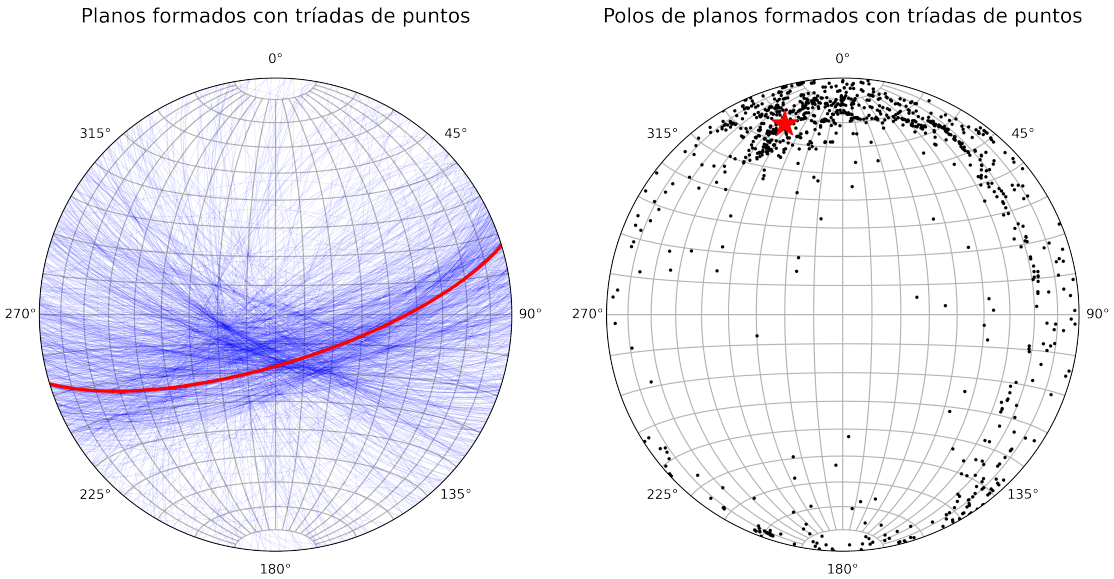
1.7.1 Instalando mplstereonet

[20]:
```
#import sys
#sys.path
#!/opt/anaconda3/bin/pip install mplstereonet
```

[21]:
```
import mplstereonet
```

1.7.2 Representando los planos en la red estreográfica

[22]:
```
fig = plt.figure()
my_step=1;
ax0 = fig.add_subplot(121, projection='stereonet')
ax0.plane(strikes[1:-1:my_step], dips[1:-1:my_step], c='blue', linewidth=0.05)
ax0.plane(rumbo_most_counts, echado_most_counts, c='r',linewidth=2.5);
ax0.grid(True)
ax0.set_title('Planos formados con tríadas de puntos', y=1.10, fontsize=15)

ax1 = fig.add_subplot(122, projection='stereonet')
ax1.pole(strikes[1:-1:my_step], dips[1:-1:my_step], c='k',Marker = '.',␣
 ↪MarkerSize=3)
ax1.pole(rumbo_most_counts, echado_most_counts, c='r', Marker='*',␣
 ↪MarkerSize=20);
ax1.grid(True)
ax1.set_title('Polos de planos formados con tríadas de puntos', y=1.10,␣
 ↪fontsize=15);
```



1.8 Encontrando el vector promedio con diferentes métodos de geología estructural

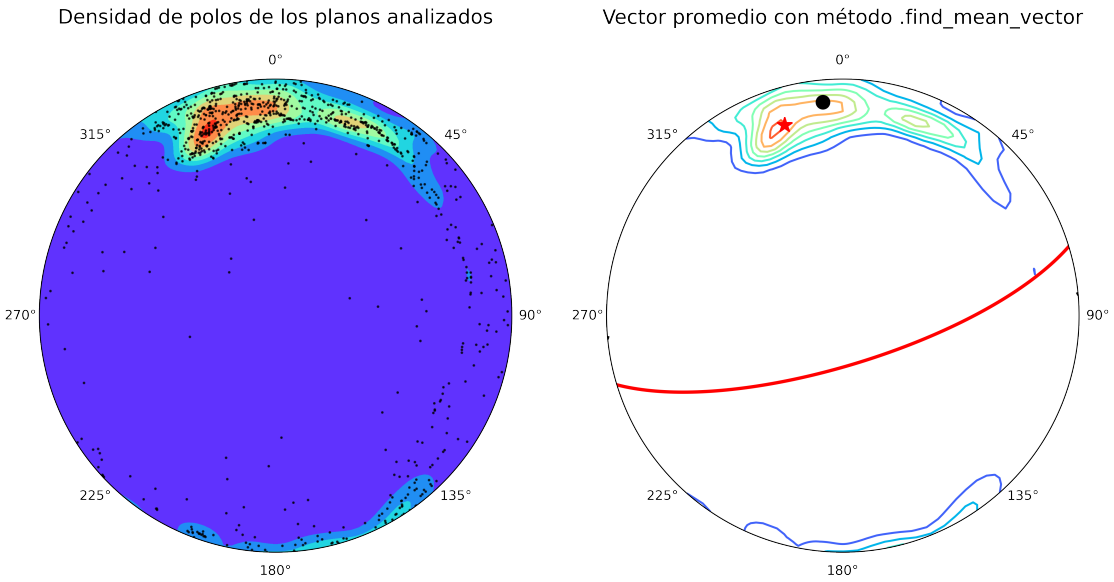
1.8.1 Mean vector (.find_mean_vector): promedio en estadística esférica

[23]:
```
meanv,length=mplstereonet.find_mean_vector(strikes, dips, measurements='poles')
print(meanv)
#Returns the mean vector for a set of measurments. By default, this expects the␣
↪input to be plunges and bearings, but the type of input can be controlled␣
↪through the measurement kwarg.
```

```
(84.6683979967766, 259.6453169968552)
```

[24]:
```
fig = plt.figure()
ax0 = fig.add_subplot(121, projection='stereonet')

ax0.density_contourf(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100)

ax0.pole(strikes, dips, c='k', Marker='.', MarkerSize=2,label='Planes', alpha=0.
↪7)
ax0.set_title('Densidad de polos de los planos analizados', y=1.10, fontsize=15)


ax1 = fig.add_subplot(122, projection='stereonet')

ax1.density_contour(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=[50, 50])
```

```
ax1.pole(meanv[0], meanv[1], c='k', Marker='.', MarkerSize=20)
ax1.plane(meanv[0], meanv[1], c='k',linewidth=2.5)
ax1.grid(False)
ax1.set_title('Vector promedio con método .find_mean_vector', y=1.10,␣
 ↪fontsize=15);

ax1.pole(rumbo_most_counts, echado_most_counts, c='r', Marker='*',␣
 ↪MarkerSize=12);
ax1.plane(rumbo_most_counts, echado_most_counts, c='r',linewidth=2.5);
```



[25]:
```
dg=mplstereonet.density_grid(strikes, dips, measurement='poles',gridsize=10)
```

[26]:
```
print(dg[2][9])
```

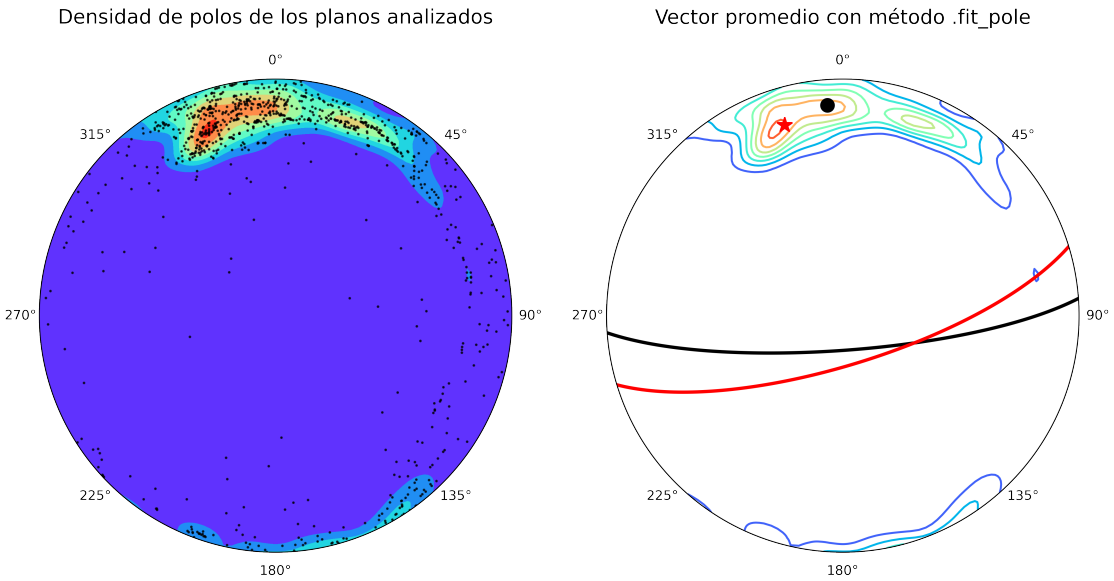
```
[12.74405907 14.0385348   7.32904996  1.12369102  0.85872843  1.21397576
  1.09833625  6.56344117 10.23606171 12.74405907]
```

1.8.2 Fit pole (.fit_pole): usando eigenvectores y eigenvalores

[27]:
```
centers2=mplstereonet.fit_pole(strikes, dips,measurement='poles')
print(centers2)

#Fits the pole to a plane to a "bullseye" of points on a stereonet.
#The pole to the best-fit plane is extracted by calculating the largest␣
 ↪eigenvector of the covariance matrix of the input measurements in cartesian␣
 ↪3D space.
```

```
(85.82963515077938, 78.09480187870672)
```

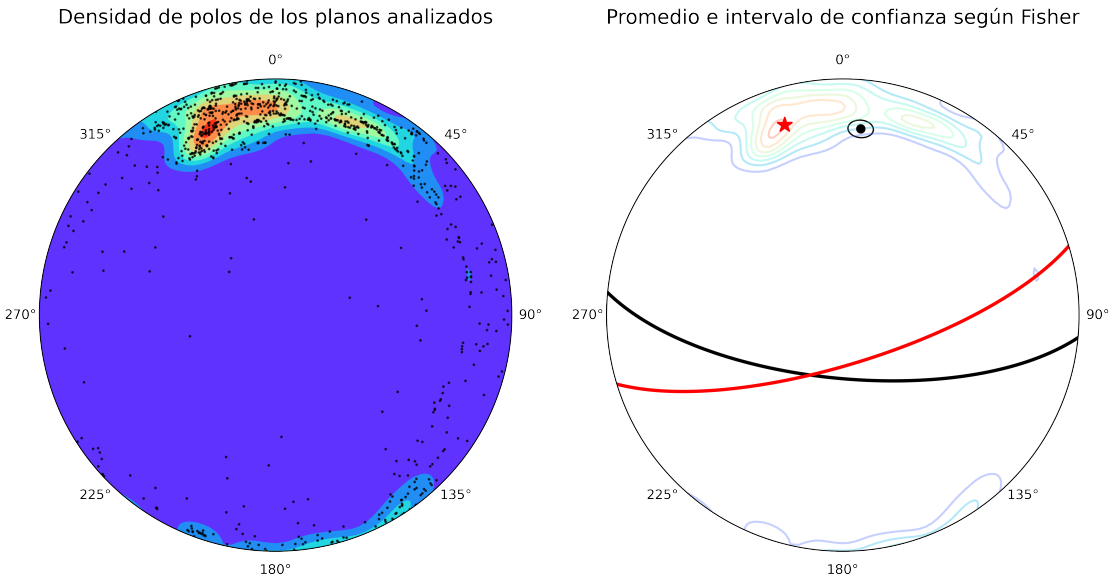
[28]:
```
fig = plt.figure()
ax0 = fig.add_subplot(121, projection='stereonet')
ax0.density_contourf(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100)
ax0.pole(strikes, dips, c='k', Marker='.', MarkerSize=2,label='Planes', alpha=0.
↪7)
ax0.set_title('Densidad de polos de los planos analizados', y=1.10, fontsize=15)

ax1 = fig.add_subplot(122, projection='stereonet')

ax1.density_contour(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100)

ax1.pole(centers2[0], centers2[1], c='k', Marker='.', MarkerSize=20)
ax1.plane(centers2[0], centers2[1], c='k',linewidth=2.5)
ax1.grid(False)
ax1.set_title('Vector promedio con método .fit_pole', y=1.10, fontsize=15);

ax1.pole(rumbo_most_counts, echado_most_counts, c='r', Marker='*',␣
↪MarkerSize=12);
ax1.plane(rumbo_most_counts, echado_most_counts, c='r',linewidth=2.5);
```



1.8.3 Fisher stats (.find_fisher_stats): encontrando el promedio según Fisher

[29]:
```
confidence=95
vector, stats = mplstereonet.find_fisher_stats(strikes, dips, conf=confidence,␣
 ↪measurement='poles')
fmean=mplstereonet.plunge_bearing2pole(vector[0], vector[1])
```

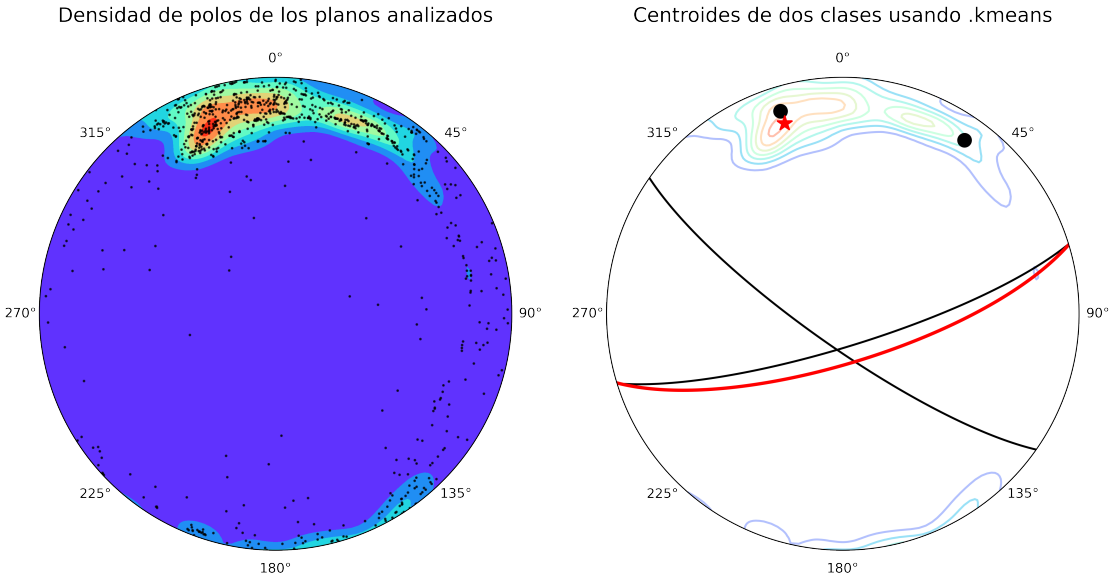
[30]:
```
fig = plt.figure()
ax0 = fig.add_subplot(121, projection='stereonet')
ax0.density_contourf(strikes, dips, measurement='poles', cmap='rainbow',␣
 ↪gridsize=100)
ax0.pole(strikes, dips, c='k', Marker='.', MarkerSize=2,label='Planes', alpha=0.
 ↪7)
ax0.set_title('Densidad de polos de los planos analizados', y=1.10, fontsize=15)

ax1 = fig.add_subplot(122, projection='stereonet')
ax1.density_contour(strikes, dips, measurement='poles', cmap='rainbow',␣
 ↪gridsize=100, alpha=0.3)

ax1.line(vector[0], vector[1], color="red");
ax1.cone(vector[0], vector[1], stats[1], facecolor="None", edgecolor="k");
ax1.plane(fmean[0], fmean[1], color="black", linewidth=2.5);
ax1.pole(fmean[0], fmean[1], color="black");
ax1.set_title('Promedio e intervalo de confianza según Fisher', y=1.10,␣
 ↪fontsize=15);

ax1.pole(rumbo_most_counts, echado_most_counts, c='r', Marker='*',␣
 ↪MarkerSize=12);
ax1.plane(rumbo_most_counts, echado_most_counts, c='r',linewidth=2.5);
```



1.8.4 K-means (.kmeans)

[31]:
```
centers3 = mplstereonet.kmeans(strikes,dips, measurement='poles')
centers3 = mplstereonet.geographic2pole(*zip(*centers3))
```

[32]:
```
fig = plt.figure()
ax0 = fig.add_subplot(121, projection='stereonet')
ax0.density_contourf(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100)
ax0.pole(strikes, dips, c='k', Marker='.', MarkerSize=2,label='Planes', alpha=0.
↪7)
ax0.set_title('Densidad de polos de los planos analizados', y=1.10, fontsize=15)


ax1 = fig.add_subplot(122, projection='stereonet')
ax1.density_contour(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100, alpha=0.4)

ax1.pole(centers3[0], centers3[1], c='k', Marker='.', MarkerSize=20);
ax1.plane(centers3[0], centers3[1], c='k');
ax1.set_title('Centroides de dos clases usando .kmeans', y=1.10, fontsize=15);

ax1.pole(rumbo_most_counts, echado_most_counts, c='r', Marker='*',␣
↪MarkerSize=12);
ax1.plane(rumbo_most_counts, echado_most_counts, c='r',linewidth=2.5);
plt.savefig('hough_vs_kmeans.png', bbox_inches='tight')
```

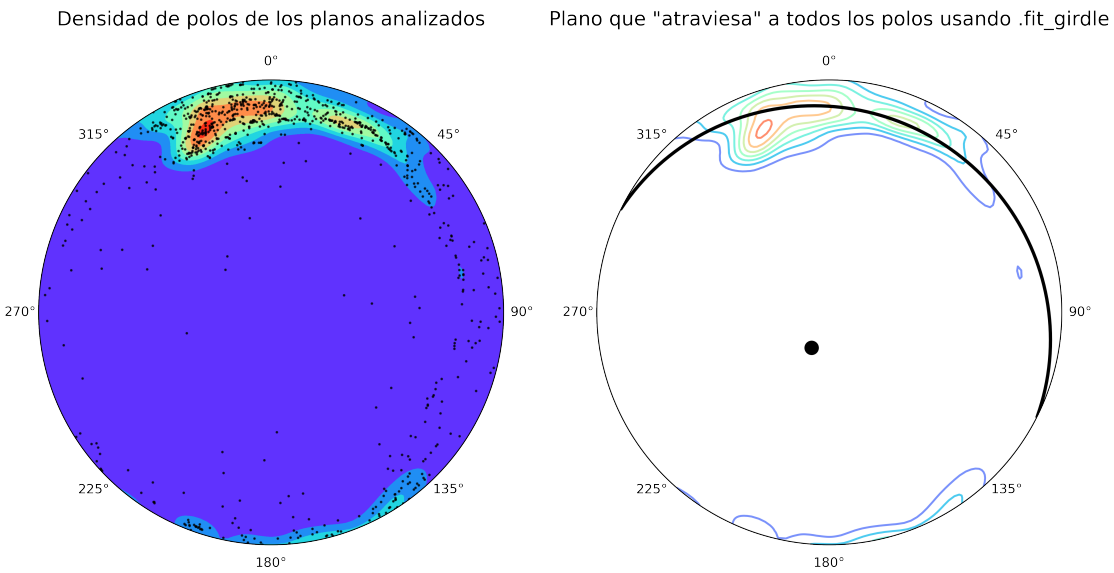


1.9 Bonus: Fit girdle (.fit_girdle)

[33]:
```
centers1=mplstereonet.fit_girdle(strikes, dips, measurement='poles')
print(centers1)
```

```
(296.57126999865886, 13.78233298258921)
```

[34]:
```
fig = plt.figure()
ax0 = fig.add_subplot(121, projection='stereonet')
ax0.density_contourf(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100)
ax0.pole(strikes, dips, c='k', Marker='.', MarkerSize=2,label='Planes', alpha=0.
↪7)
ax0.set_title('Densidad de polos de los planos analizados', y=1.10, fontsize=15)

ax1 = fig.add_subplot(122, projection='stereonet')

ax1.density_contour(strikes, dips, measurement='poles', cmap='rainbow',␣
↪gridsize=100, alpha=0.7)

ax1.pole(centers1[0], centers1[1], c='k', Marker='.', MarkerSize=20)
ax1.plane(centers1[0], centers1[1], c='k', linewidth=2.5)
ax1.grid(False)
ax1.set_title('Plano que "atraviesa" a todos los polos usando .fit_girdle', y=1.
↪10, fontsize=15);
```

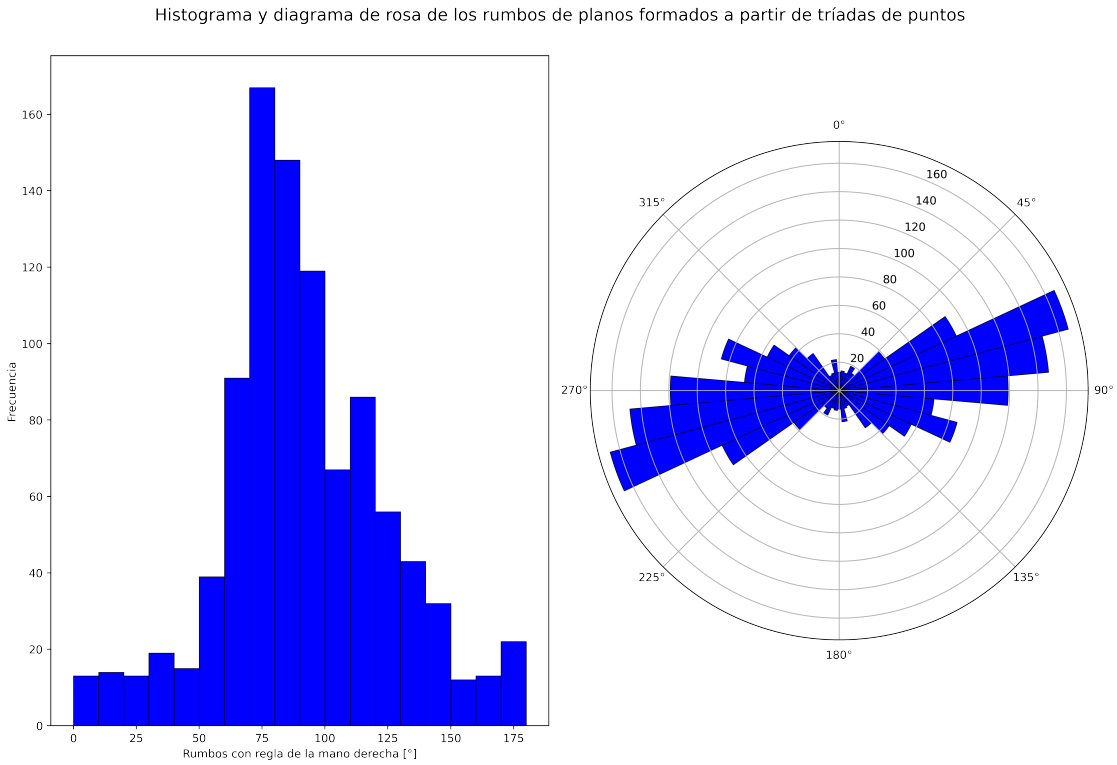


1.9.1 Bonus: Convirtiendo los datos para crear un diagrama de rosa de los rumbos

[35]:
```
bin_edges = np.arange(-5, 366, 10)
number_of_strikes, bin_edges = np.histogram(strikes, bin_edges)
number_of_strikes[0] += number_of_strikes[-1]
half = np.sum(np.split(number_of_strikes[:-1], 2), 0)
two_halves = np.concatenate([half, half])
```

1.9.2 Comparando histograma vs diagrama de rosas

[36]:
```
fig = plt.figure()

fig.suptitle('Histograma y diagrama de rosa de los rumbos de planos formados a␣
 ↪partir de tríadas de puntos', y=1.05, fontsize=15)
ax0 = fig.add_subplot(121)
ax0.bar(np.arange(0, 180, 10), half,
       width=10, bottom=0.0, color='blue', edgecolor='k',linewidth=0.
 ↪5,align='edge')
ax0.set_ylabel('Frecuencia')
ax0.set_xlabel('Rumbos con regla de la mano derecha [°]')

ax1 = fig.add_subplot(122, projection='polar');
ax1.bar(np.deg2rad(np.arange(0, 360, 10)), two_halves,
       width=np.deg2rad(10), bottom=0.0, color='blue',␣
 ↪edgecolor='k',linewidth=0.5);
ax1.set_theta_zero_location('N');
ax1.set_theta_direction(-1);
ax1.grid(True)
fig.tight_layout()
```



Referencias:

Canal de Youtube de Darío: https://www.youtube.com/channel/UCmeMXDwmAyZfmgi3ZiXffYw

1. Pubben, T. (2021). Computationally calculating strike and dip from 3 points – T.J. Scientific. Retrieved 27 March 2021, from http://www.tjscientific.com/2017/08/16/computationally-calculating-strike-and-dip-from-3-points/
2. Kington, J. (2021). mplstereonet Package — mplstereonet 0.6-dev documentation. Retrieved 27 March 2021, from https://mplstereonet.readthedocs.io/en/latest/mplstereonet.html
3. Arellano Gil, J., Llata Romero, R., Carreón Méndez, M., & Morales Barrera, W. (2002). Ejercicios de geología estructural (1st ed.). México, D.F.: Universidad Nacional Autónoma de México, Facultad de Ingeniería, http://www.dict.unam.mx/images/upload/libros/Ejercicios_de_Geologia_Estructural_JAG-SC.pdf
4. Arellano Gil, J., Llata Romero, R., Carreón Méndez, M., & Morales Barrera, W. (2002). Ejercicios de geología estructural (1st ed.). México, D.F.: Universidad Nacional Autónoma de México, Facultad de Ingeniería.

Este producto docente ha sido financiado por la DGAPA-UNAM a través del proyecto PAPIME PE101020 "Geología desde los ojos de un dron: una poderosa herramienta educativa para los actuales ingenieros de la Tierra".