Clase 5

Contenido

1.- Análisis Exploratorio de Datos.

Introducción

El análisis de datos exploratorio (EDA) lo utilizan los científicos de datos para analizar e investigar conjuntos de datos y resumir sus principales características, empleando a menudo métodos de visualización de datos. Ayuda a determinar la mejor manera de manipular los orígenes de datos para obtener las respuestas que necesita, lo que permite a los científicos de datos descubrir patrones, detectar anomalías, probar una hipótesis o comprobar supuestos.

El EDA se utiliza principalmente para ver qué datos pueden revelarse más allá de la tarea de modelado formal o las pruebas de hipótesis, y permite conocer mejor las variables de conjunto de datos y las relaciones entre ellas. También permite determinar si las técnicas estadísticas que está considerando para el análisis de datos son apropiadas.

¿Por qué es importante el análisis exploratorio de datos en la ciencia de datos?

El principal objetivo del EDA es consultar los datos antes de hacer cualquier suposición. Permite identificar errores obvios, así como comprender mejor los patrones en los datos, detectar valores atípicos o sucesos anómalos y encontrar relaciones interesantes entre las variables.

Clase 5

Los científicos de datos pueden utilizar el análisis exploratorio para garantizar que los resultados que generan sean válidos y aplicables a las conclusiones y objetivos de negocio deseados. El EDA también permite confirmar a las partes interesadas que están haciendo las preguntas correctas. El EDA ayuda a responder las preguntas sobre desviaciones estándar, variables categóricas e intervalos de confianza. Una vez que se ha completado el EDA y se ha extraído la información útil, sus características pueden utilizarse para un análisis o modelado de datos más complejo, incluido aprendizaje automático.

Herramientas de análisis exploratorio de datos.

Las funciones y técnicas estadísticas específicas que pueden realizarse con las herramientas de EDA incluyen:

- Técnicas de agrupación en clúster y reducción de dimensiones, que permiten crear visualizaciones gráficas de datos de grandes dimensiones que contienen muchas variables.
- Visualización univariante de cada campo en el conjunto de datos en bruto, con estadísticas de resumen.
- Visualizaciones bivariantes y estadísticas de resumen que le permiten evaluar la relación entre cada variable del conjunto de datos y la variable de destino que está buscando.
- Visualizaciones multivariantes, para correlacionar y comprender las interacciones entre los diferentes campos en los datos.
- K-means Clustering es un método de agrupación en clúster de aprendizaje no supervisado en el que los puntos de datos se asignan a K grupos, es decir, el número de clústeres, en función de la distancia del centroide de cada grupo. Los puntos de datos más próximos a un determinado centroide se agruparán en la misma categoría. K-means Clustering se utiliza a menudo en la segmentación de mercado, el reconocimiento de patrones y la compresión de imágenes.
- Los modelos predictivos como, por ejemplo, la regresión lineal, utilizan estadísticas y datos para predecir los resultados.

Tipos de análisis exploratorio de datos.

Hay cuatro tipos principales de EDA:

- **No gráfico univariante**. Es la forma más simple de análisis de datos, donde los datos que se analizan consisten en una sola variable. Como es una sola variable, no se ocupa de las causas o relaciones. El objetivo principal del análisis univariante es describir los datos y encontrar los patrones que existen en ellos.

- **Gráfico univariante**. Los métodos no gráficos no ofrecen una imagen completa de los datos. Por lo tanto, se requieren métodos gráficos. Los tipos más comunes de gráficos univariantes incluyen:
    - *Tramas de tallo y hoja*, que muestran todos los valores de datos y la forma de la distribución.
    - *Histogramas*, un diagrama de barras donde cada barra representa la frecuencia (recuento) o la proporción (recuento/recuento total) de casos para un rango de valores.
    - *Diagramas de caja*, que representan gráficamente el resumen de cinco números de mínimo, primer cuartil, mediana, tercer cuartil y máximo.
- **No gráfico multivariante**. Se obtienen datos multivariantes de más de una variable. Las técnicas de EDA no gráfico multivariante generalmente muestran la relación entre dos o más variables de los datos mediante tabulación cruzada o estadísticas.
- **Gráfico multivariante**. Los datos multivariantes utilizan gráficos para mostrar las relaciones entre dos o más conjuntos de datos. El gráfico más utilizado es un diagrama de barras o gráfico de barras agrupadas, en el que cada grupo representa un nivel de una de las variables y cada barra de un grupo representa los niveles de la otra variable.

Otros tipos comunes de gráficos multivariantes incluyen:

- Trama de dispersión, que se utiliza para trazar puntos de datos en un eje horizontal y uno vertical para mostrar cuánto afecta una variable a otra.
- Gráfico multivariante, que es una representación gráfica de las relaciones entre los factores y una respuesta.
- Diagrama de ejecución, que es un gráfico de líneas de datos trazados a lo largo del tiempo.
- Gráfico de burbujas, que es una visualización de datos que muestra varios círculos (burbujas) en un gráfico bidimensional.
- Mapa de calor, que es una representación gráfica de datos donde los valores se representan por color.

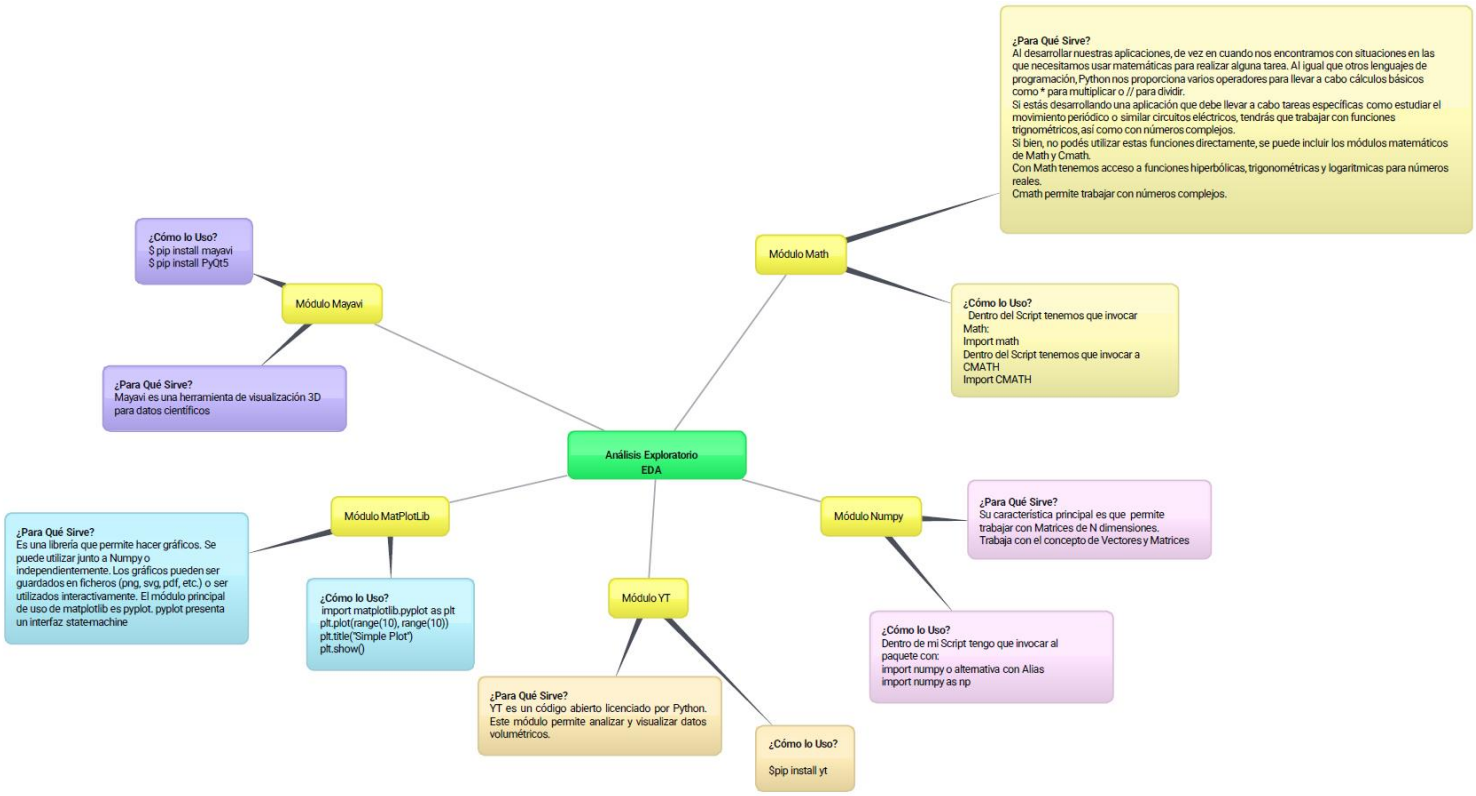
2.- Análisis exploratorio de datos usando Python.

El lenguaje Python emplea las siguientes librerías:

- numpy
- pandas
- matplotlib
- math
- yt
- mayavi

Clase 5

3.- Mapa mental EDA con Python.



4.- Librería Numpy.

Su característica más potente es que puede trabajar con matrices (array) de n dimensiones. También ofrece funciones básicas de algebra lineal, transformada de Fourier, capacidades avanzadas con números aleatorios, y herramientas de integración con otros lenguajes de bajo nivel como Fortran, C y C++.

Numpy es el paquete más básico pero poderoso para la computación científica y la manipulación de datos en Python.

Nos permite trabajar con matrices y matrices multidimensionales.

La mayoría de las otras bibliotecas que se usan en el análisis de datos con Python, como scikit-learn , SciPy y Pandas usan algunas de las características de NumPy. Para comenzar a utilizar en nuestros proyectos numpy es necesario importar, así como aquellos tiempos en los que estábamos aprendiendo a utilizar la librería random. La forma más simple de importar numpy es:

```
import numpy
```

Otra forma es:

Clase 5

```
import numpy as np
```

Las dos sentencias hacen lo mismo, la diferencia es que la última se le añade un "alias" para escribir menos. Por ejemplo, si usamos la primera opción tendrás que escribir numpy como prefijo a todas las propiedades.

```
a=numpy.array([i for i in range(6)]
```

Con el alias queda así.

```
a=np.array([i for i in range(6)]
```

Para entender el poder de esta librería, vamos a recordar las listas, que se han explicado en la unidad anterior. Son como cajitas que contenían información, bueno pues los arreglos (vectores) son listas de una o más dimensiones donde sus elementos son del mismo tipo (usualmente numérico). Si bien, en la entrada de listas usamos listas dentro de listas, en realidad eso era un arreglo de dos dimensiones. Y los arreglos de dos dimensiones lo llamamos Matrices.

Arreglo de una dimensión

```
a=np.array([10,20,30])
```

Arreglo de dos dimensiones (matrices)

```
b=np.array([[10,20,30],[40,50,60]])
```

Como en las matemáticas, una matriz debe tener el mismo número de columnas. Como los arreglos son listas, es lógico que comienzan a contarse desde el índice 0.

```
a[0]=10
```

```
b[0][0]=10
```

Desde la consola instalamos numpy de la siguiente manera:

```
$ pip install numpy
```

Clase 5

Algunos ejemplos:

1. Crear una matriz 3x3 con números aleatorios

```
a = np.arange(0, 9).reshape(3, 3)
```

2. Arreglo con números aleatorios del 1 al 9 de dos en dos

```
x = np.arange(1, 10, 2)
```

3. Suma del Arreglo X

```
suma = np.sum(x)
```

4. Matriz Identidad de 3x3

```
y = np.identity(3)
```

5. Matriz a la potencia 3

```
z = 3**x
```

6. Convierto dos array en una matriz 2x2

```
f = np.array([[1, 2, 3], [2, 3, 4]])
```

En el siguiente enlace puede accederse a un tutorial de numpy:

Tutorial de NumPy en Español

5.- Librería Pandas.

Es un paquete de Python que proporciona estructuras de datos similares a los dataframes de R. Pandas depende de Numpy, la librería que añade un potente tipo matricial a Python.

Los principales tipos de datos que pueden representarse con pandas son:

- Datos tabulares con columnas de tipo heterogéneo con etiquetas en columnas y filas.

Clase 5

- Series temporales.

Pandas proporciona herramientas que permiten:

- leer y escribir datos en diferentes formatos: CSV, Microsoft Excel, bases SQL y formato HDF5
- seleccionar y filtrar de manera sencilla tablas de datos en función de posición, valor o etiquetas
- fusionar y unir datos
- transformar datos aplicando funciones tanto en global como por ventanas
- manipulación de series temporales
- hacer gráficas

En pandas existen tres tipos básicos de objetos todos ellos basados a su vez en Numpy:

- Series (listas, 1D),
- DataFrame (tablas, 2D) y
- Panels (tablas 3D).

En el siguiente enlace puede accederse a un tutorial de pandas:

Tutorial de Pandas en Español

6.- Librería MatPlotLib.

matplotlib

Es una librería que permite hacer gráficos. Se puede utilizar junto a Numpy o independientemente. Los gráficos pueden ser guardados en archivos (png, svg, pdf, etc.) o ser utilizados interactivamente. El módulo principal de uso de matplotlib es pyplot. pyplot presenta un interfaz state-machine.

Otra interfaz similar es pylab que combina pyplot con numpy.

Además de los tipos de gráficos incluidos directamente en matplotlib hay extensiones que añaden más funcionalidad como:

- Basemap. Mapas geográficos.
- Mplot3d: Gráficos 3D.

Links interesantes:

https://matplotlib.org/2.0.2/api/pyplot_api.html

Clase 5

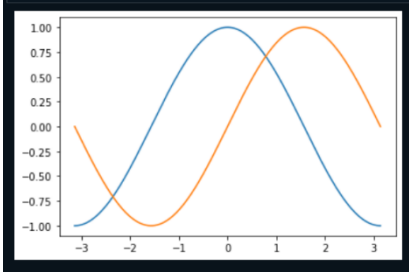
Matplotlib viene con un conjunto de valores predeterminados que permiten la personalización de todos los tipos de propiedades. Puede controlar los valores predeterminados de casi todas las propiedades en matplotlib: tamaño de figura y dpi, grosor de línea, color y estilo, ejes, ejes y propiedades de cuadrícula, propiedades de texto y fuente, etc.

A continuación, vamos a ver algunos ejemplos y como configurar sus Parámetros.

Gráfica Seno/Coseno:

```
import pylab as pl
import numpy as np

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

pl.plot(X, C)
pl.plot(X, S)

pl.show()
```

✓ 0.1s

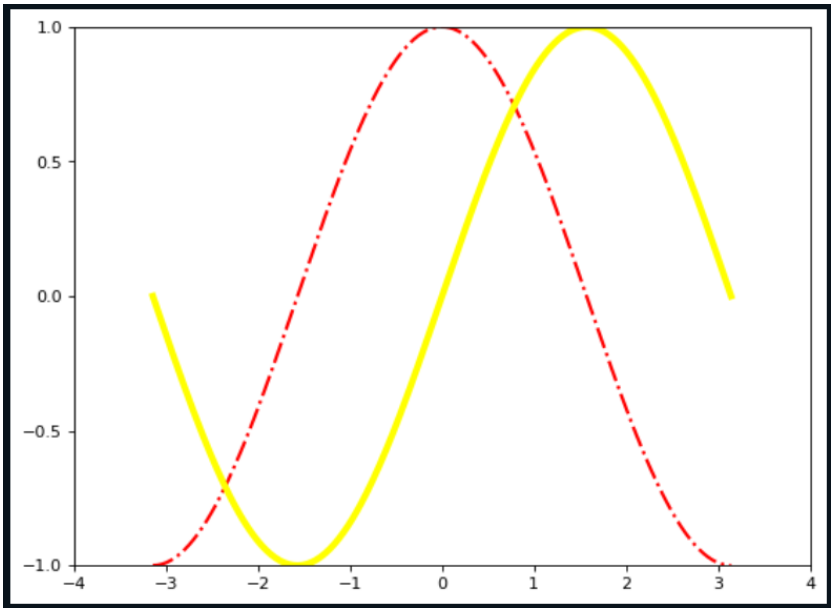


Al Ejemplo Anterior voy a ir agregando funcionalidad y cambiando parámetros para ir mejorando el gráfico.

```
# Crear una figura de 8x6 puntos de tamaño, 80 puntos por pulgada
pl.figure(figsize=(8, 6), dpi=80)

# Crear una nueva subgráfica de 1x1
pl.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

# En uno de los parámetros puedo especificar el color y el grosor de la línea p ara la gráfica en este caso elijo Rojo "Red"
# y Grosor 2, el parámetro lineStyle define el tipo de Línea. En este caso voy a elegir -. para que se diferencie con la
# segunda gráfica

pl.plot(X, C, color="Red", linewidth=2.0, linestyle="-.")

# Voy a cambiar de la segunda gráfica el color a Amarillo, Yellow y el Grosor d e La línea a 4, tipor de línea -.

pl.plot(X, S, color="Yellow", linewidth=4.0, linestyle="-")

# Puedo poner límites para el Eje x. En el punto anterior la gráfica no tenía l ímites en este caso va a ir de -4 a 4
pl.xlim(-4.0, 4.0)

# Ticks en x - Con xticks puedo cambiar la graduación del eje
pl.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Puedo poner límites para el Eje y. En el punto anterior la gráfica no tenía l ímites en este caso va a ir de -1 a 1
pl.ylim(-1.0, 1.0)

# Ticks en y Con yticks puedo cambiar la graduación del eje
pl.yticks(np.linspace(-1, 1, 5, endpoint=True))

# Muestro el Resultado por Pantalla
pl.show()
```

Y la ejecución del código nos quedaría:

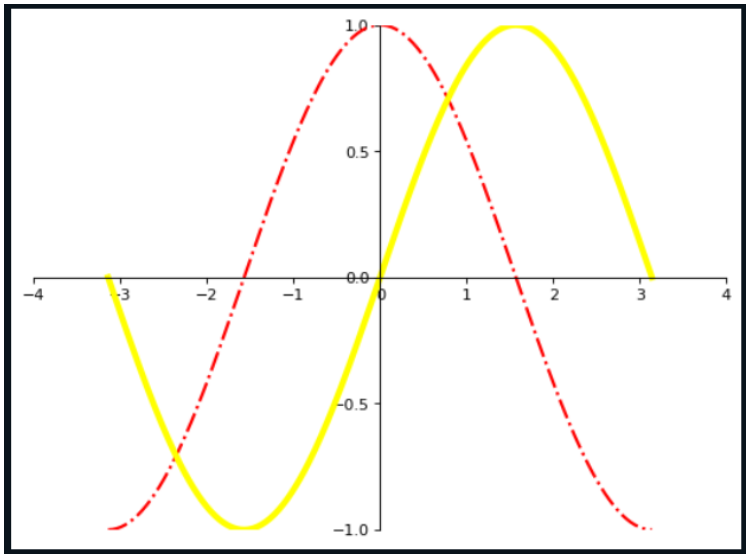
Clase 5



Podríamos querer cambiar los Spines que unen las marcas de graduación del eje y muestran los límites de datos. En la instancia anterior estaban sobre el eje podemos cambiar eso con las opciones que tenemos disponibles arriba, abajo, izquierda, derecha.

```
import pylab as pl
import numpy as np

# Crear una figura de 8x6 puntos de tamaño, 80 puntos por pulgada
pl.figure(figsize=(8, 6), dpi=80)

# Crear una nueva subgráfica en una rejilla de 1x1
pl.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

# En uno de los parámetros puedo especificar el color y el grosor de l a línea para la gráfica en este caso elijo Rojo "Red"
# y Grosor 2, el parámetro LineStyle define el tipo de Linea. En este caso voy a elegir -. para que se diferencie con la
# segunda gráfica

pl.plot(X, C, color="Red", linewidth=2.0, linestyle="-.")

# Voy a cambiar de la segunda gráfica el color a Amarillo, Yellow y el Grosor de la Línea a 4, tipor de línea -.

pl.plot(X, S, color="Yellow", linewidth=4.0, linestyle="-")
#Configuro los Spines de la siguiente manera

ax = pl.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

# Puedo poner límites para el Eje x. En el punto anterior la gráfica n o tenía límites en este caso va a ir de -4 a 4
pl.xlim(-4.0, 4.0)

# Ticks en x - Con xticks puedo cambiar la graduación del eje
pl.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Puedo poner límites para el Eje y. En el punto anterior la gráfica n o tenía límites en este caso va a ir de -1 a 1
pl.ylim(-1.0, 1.0)

# Ticks en y Con yticks puedo cambiar la graduación del eje
pl.yticks(np.linspace(-1, 1, 5, endpoint=True))

# Muestro el Resultado por Pantalla
pl.show()
```

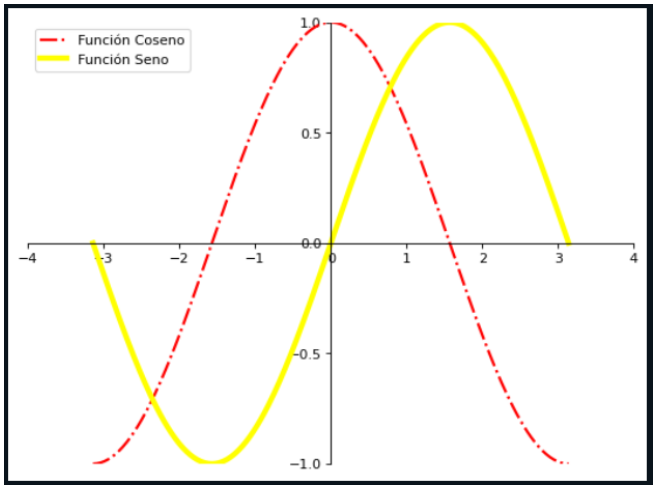
Clase 5



Si seguimos personalizando el gráfico podemos agregar una leyenda.

```
import pylab as pl
import numpy as np

# Crear una figura de 8x6 puntos de tamaño, 80 puntos por pulgada
pl.figure(figsize=(8, 6), dpi=80)

# Crear una nueva subgráfica en una rejilla de 1x1
pl.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

# En uno de los parámetros puedo especificar el color y el grosor de l a línea para la gráfica en este caso elijo Rojo "Red"
# y Grosor 2, el parámetro LineStyle define el tipo de Linea. En este caso voy a elegir -. para que se diferencie con la
# segunda gráfica. Agrego el Parámetro Label para luego mostrar en la leyenda

pl.plot(X, C, color="Red", linewidth=2.0, linestyle="-.", label="Función Coseno")

# Voy a cambiar de la segunda gráfica el color a Amarillo, Yellow y el Grosor de la Línea a 4, tipor de línea -.
#Agrego el parámetro label para luego mostrar la leyenda

pl.plot(X, S, color="Yellow", linewidth=4.0, linestyle="-", label="Función Seno")

#Configuro los Spines de la siguiente manera

ax = pl.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

# Puedo poner límites para el Eje x. En el punto anterior la gráfica n o tenía límites en este caso va a ir de -4 a 4
pl.xlim(-4.0, 4.0)

# Ticks en x - Con xticks puedo cambiar la graduación del eje
pl.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Puedo poner límites para el Eje y. En el punto anterior la gráfica n o tenía límites en este caso va a ir de -1 a 1
pl.ylim(-1.0, 1.0)

# Ticks en y Con yticks puedo cambiar la graduación del eje
pl.yticks(np.linspace(-1, 1, 5, endpoint=True))

pl.legend(loc='upper left')

# Muestro el Resultado por Pantalla
pl.show()
```

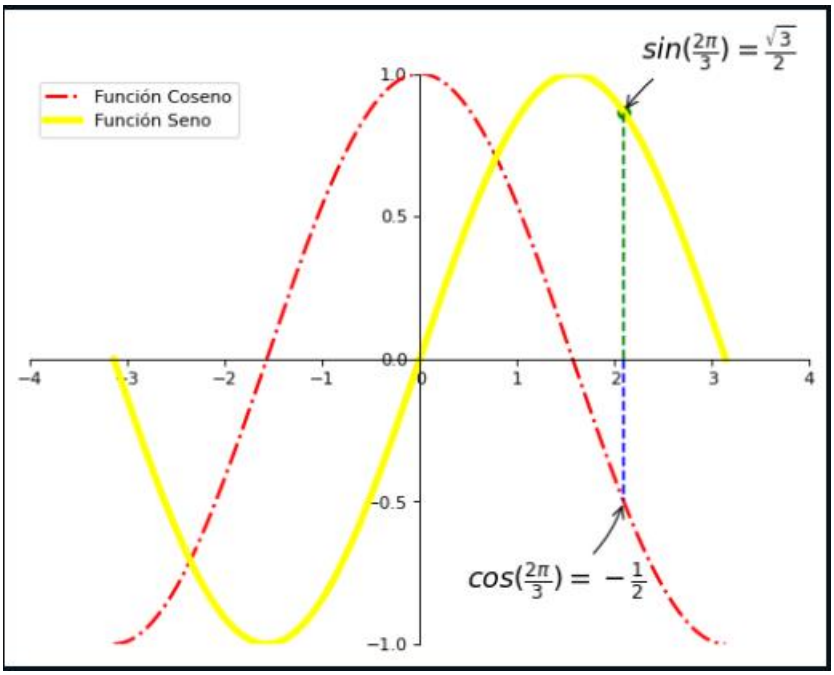
Clase 5



Seguimos personalizando, podríamos querer poner anotaciones en el gráfico.

```
import pylab as pl
import numpy as np

# Crear una figura de 8x6 puntos de tamaño, 80 puntos por pulgada
pl.figure(figsize=(8, 6), dpi=80)

# Crear una nueva subgráfica en una rejilla de 1x1
pl.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

# En uno de los parámetros puedo especificar el color y el grosor de l a línea para la gráfica en este caso elijo Rojo "Red"
# y Grosor 2, el parámetro lineStyle define el tipo de línea. En este caso voy a elegir -. para que se diferencie con la
# segunda gráfica. Agrego el Parámetro Label para luego mostrar en la leyenda

pl.plot(X, C, color="Red", linewidth=2.0, linestyle="-.", label="Función Coseno")
# Voy a cambiar de la segunda gráfica el color a Amarillo, Yellow y el Grosor de la Línea a 4, tipor de línea -.
#Agrego el parámetro label para luego mostrar la leyenda

pl.plot(X, S, color="Yellow", linewidth=4.0, linestyle="-", label="Función Seno")

#Configuro los Spines de la siguiente manera

ax = pl.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

# Puedo poner límites para el Eje x. En el punto anterior la gráfica n o tenía límites en este caso va a ir de -4 a 4
pl.xlim(-4.0, 4.0)

# Ticks en x - Con xticks puedo cambiar la graduación del eje
pl.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Puedo poner límites para el Eje y. En el punto anterior la gráfica n o tenía límites en este caso va a ir de -1 a 1
pl.ylim(-1.0, 1.0)

# Ticks en y Con yticks puedo cambiar la graduación del eje
pl.yticks(np.linspace(-1, 1, 5, endpoint=True))

pl.legend(loc='upper left')

#Anotaciones en el Gráfico.
t = 2*np.pi/3
#Trazo la Línea de la función coseno hasta el Eje en Color Azul
pl.plot([t, t], [0, np.cos(t)],
color='blue', linewidth=1.5, linestyle="--") #Grafico el punto Azul de la función Coseno pl.scatter([t, ], [np.cos(t), ], 50, color="blue")
pl.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$', xy=(t, np.sin(t)), xycoords='data',
xytext=(+10, +30), textcoords='offset points', fontsize=16, arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

#Trazo la Línea de la función seno hasta el Eje en Color Verde
pl.plot([t, t], [0, np.sin(t)], color='Green', linewidth=1.5, linestyle="--")

#Trazo el Punto de la función Seno en Verde
pl.scatter([t, ], [np.sin(t), ], 50, color="Green")
pl.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$', xy=(t, np.cos(t)), xycoords='data', xytext=(-90, -50), textcoords='offset points',fontsize=16, arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

# Muestro el Resultado por Pantalla
pl.show()
```

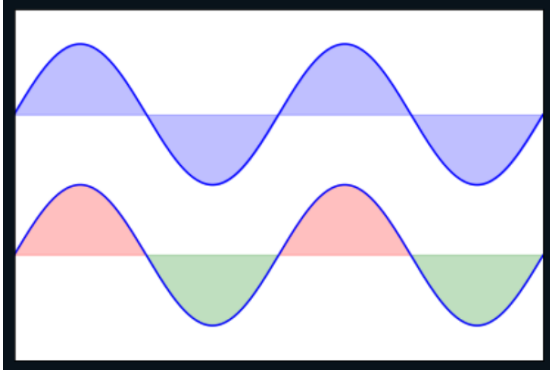
Clase 5



Gráficos Regulares.

```
import numpy as np

n = 256
X = np.linspace(-np.pi, np.pi, n, endpoint=True)
Y = np.sin(2 * X)

pl.axes([0.025, 0.025, 0.95, 0.95])

pl.plot(X, Y + 1, color='blue', alpha=1.00)
pl.fill_between(X, 1, Y + 1, color='blue', alpha=.25)

pl.plot(X, Y - 1, color='blue', alpha=1.00)
pl.fill_between(X, -1, Y - 1, (Y - 1) > -1, color='red', alpha=.25)
pl.fill_between(X, -1, Y - 1, (Y - 1) < -1, color='green', alpha=.25)

pl.xlim(-np.pi, np.pi)
pl.xticks(())
pl.ylim(-2.5, 2.5)
pl.yticks(())

pl.show()
```

✓ 0.0s



Clase 5

Para el relleno de las zonas hacemos uso de la función fill between

Gráfico de dispersión.

```
import numpy as np

n = 1024
X = np.random.normal(0, 1, n)
Y = np.random.normal(0, 1, n)
T = np.arctan2(Y, X)

pl.axes([0.025, 0.025, 0.95, 0.95])

#Invoco al gráfico de dispersión
pl.scatter(X, Y, s=75, c=T, alpha=.5)

#Defino los límites del gráfico pl.xlim(-1.5, 1.5)
pl.xticks(())
pl.ylim(-1.5, 1.5)
pl.yticks(())

pl.show()
```

✓ 0.0s

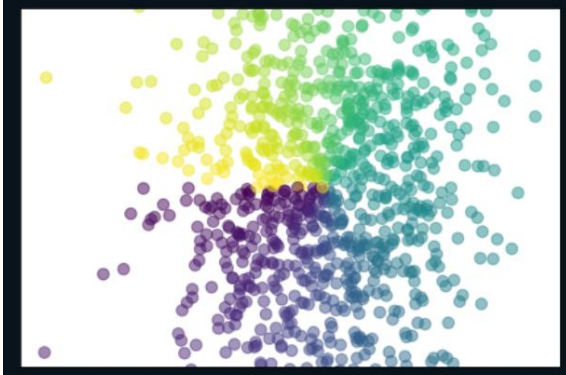
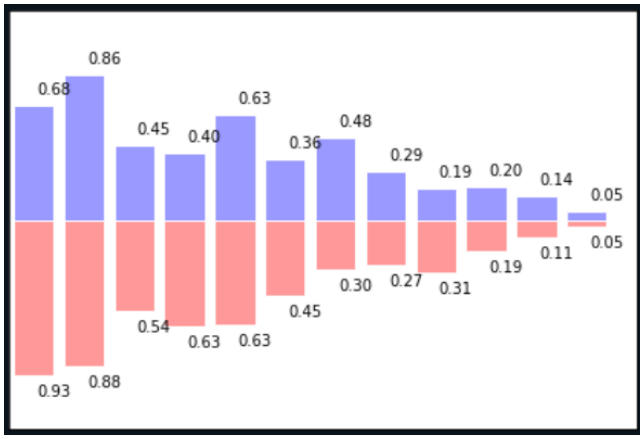


Gráfico de barras.

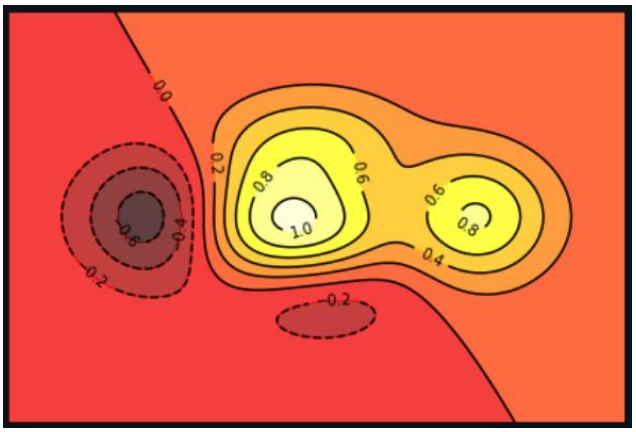
```
n = 12
X = np.arange(n)
Y1 = (1 - X / float(n)) * np.random.uniform(0.5, 1.0, n)
Y2 = (1 - X / float(n)) * np.random.uniform(0.5, 1.0, n)

pl.axes([0.025, 0.025, 0.95, 0.95])
pl.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
pl.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')

for x, y in zip(X, Y1):
    pl.text(x + 0.4, y + 0.05, '%.2f' % y, ha='center', va= 'bottom')

for x, y in zip(X, Y2):
    pl.text(x + 0.4, -y - 0.05, '%.2f' % y, ha='center', va= 'top')

pl.xlim(-.5, n)
pl.xticks(())
pl.ylim(-1.25, 1.25)
pl.yticks(())

pl.show()
```

Clase 5



Otros gráficos.

```
import numpy as np

def f(x,y):
    return (1 - x / 2 + x**5 + y**3) * np.exp(-x**2 -y**2)

n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X,Y = np.meshgrid(x, y)

pl.axes([0.025, 0.025, 0.95, 0.95])

pl.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap=pl.cm.hot)
C = pl.contour(X, Y, f(X, Y), 8, colors='black', linewidth=.5)
pl.clabel(C, inline=1, fontsize=10)

pl.xticks(())
pl.yticks(())

pl.show()
```



Clase 5

Gráfico de tortas.

```
import numpy as np

n = 10
Z = np.ones(n)
Z[-1] *= 2

pl.axes([0.025, 0.025, 0.95, 0.95])

pl.pie(Z, explode=Z*.05, colors = ['%f' % (i/float(n)) for i in range( n)])
pl.axis('equal')
pl.xticks(())
pl.yticks()

pl.show()
```

✓ 0.0s

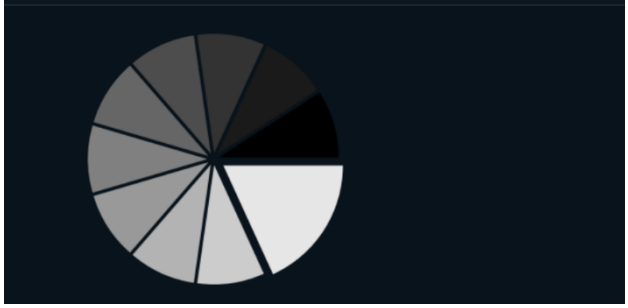
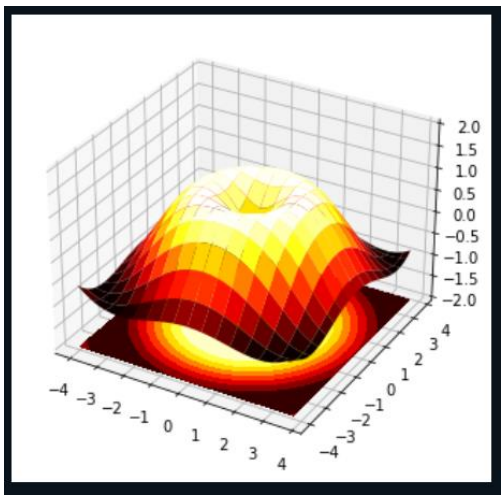


Gráfico 3D.

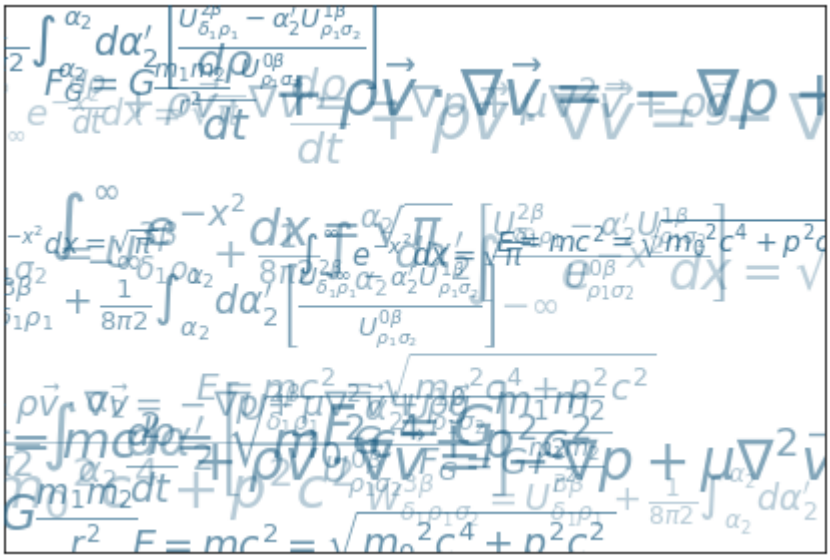
```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

fig = pl.figure()
ax = Axes3D(fig)
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X ** 2 + Y ** 2)
Z = np.sin(R)

ax.plot_surface(X, Y, Z, rstride=2, cstride=2, cmap=pl.cm.hot)
ax.contourf(X, Y, Z, zdir='z', offset=-2, cmap=pl.cm.hot)
ax.set_zlim(-2, 2)

pl.show()
```



Clase 5

Gráfico de texto.

```
import numpy as np

eqs = []
eqs.append((r"$W^{3\beta}_{\delta_1 \rho_1 \sigma_2} = U^{3\beta}_{\delta_1 \rho_1} + \frac{1}{8 \pi 2} \int^
{\alpha_2}_{\alpha_2} d \alpha^\prime_2 \left[\frac{ U^{2\beta}_{\delta_1 \rho_1} - \alpha^\prime_2U^{1\beta}_
{\rho_1 \sigma_2} }{U^{0\beta}_{\rho_1 \sigma_2}}\right]$"))
eqs.append((r"$\frac{d\rho}{d t} + \rho \vec{v}\cdot\nabla\vec{v} = -\nabla p + \mu\nabla^2 \vec{v} + \rho \vec
{g}$"))
eqs.append((r"$\int_{-\infty}^\infty e^{-x^2}dx=\sqrt{\pi}$"))
eqs.append((r"$E = mc^2 = \sqrt{{m_0}^2c^4 + p^2c^2}$"))
eqs.append((r"$F_G = G\frac{m_1m_2}{r^2}$"))

pl.axes([0.025, 0.025, 0.95, 0.95])

for i in range(24):
    index = np.random.randint(0, len(eqs))
    eq = eqs[index]
    size = np.random.uniform(12, 32)
    x,y = np.random.uniform(0, 1, 2)
    alpha = np.random.uniform(0.25, .75)
    pl.text(x, y, eq, ha='center', va='center', color="#11557c", alpha=alpha, transform=pl.gca().transAxes,
    fontsize=size, clip_on=True)

pl.xticks(())
pl.yticks(())

pl.show()
```



En el siguiente enlace puede accederse a un tutorial de matplotlib:

[Tutorial de Matplotlib en Español]

Clase 5

7.- Librería Math.

Este módulo provee acceso a funciones matemáticas definidas en C. Estas funciones no pueden ser usadas con números complejos. En este caso se deben usar las funciones del módulo CMATH para números complejos.

La distinción entre funciones que admiten números complejos y aquellas que no lo hacen, ya que la mayoría de los usuarios no quieren aprender tantas matemáticas como sea necesario para comprender números complejos. Recibir una excepción en lugar de un resultado complejo permite una detección más temprana del número complejo inesperado utilizado como parámetro, para que el programador pueda determinar cómo y por qué se generó en primer lugar.

Para utilizar estos módulos en el script, debo invocar la sentencia Import Math para el módulo Math e import Cmath para el módulo Cmath.

Ejemplos:

```
import math

def getsin(x):
    multiplier = 1
    result = 0

    for i in range(1,20,2):
        result += multiplier*pow(x,i)/math.factorial(i)
        multiplier *= -1

    return result

print(getsin(math.pi/2)) # Devuelve 1.0
print(getsin(math.pi/4)) # Devuelve 0.7071067811865475
```

✓ 0.0s

```
1.0
0.7071067811865475
```

```
import math

print(math.ceil(1.001)) # Devuelve 2
print(math.floor(1.001)) # Devuelve 1
print(math.factorial(10)) # Devuelve 3628800
print(math.gcd(10,125)) # Devuelve 5
print(math.trunc(1.001)) # Devuelve 1
print(math.trunc(1.999)) # Devuelve 1
```

✓ 0.0s

```
2
1
3628800
5
1
1
```

Clase 5

8.- Librería yt.

YT es un código abierto licenciado por Python. Este módulo permite analizar y visualizar datos volumétricos.

YT soporta:

- Variables estructuradas y no estructuradas.
- Datos discretos o muestreados, como partículas. Centrado en la indagación física significativa, se ha aplicado en dominios como la astrofísica, la sismología, la ingeniería nuclear, la dinámica molecular y la oceanografía.

Para instalar el paquete desde la consola se debe hacer lo siguiente:

```
$ pip install yt
```

9.- Librería Mayavi.

Mayavi es una herramienta de visualización 3D para datos científicos. Utiliza el kit de herramientas de visualización o VTK '*debajo del capó*'. Usando el poder de VTK, MayaVI es capaz de producir una variedad de gráficos y figuras tridimensionales. Está disponible como una aplicación de software independiente y también como una biblioteca. Similar a Matplotlib, esta biblioteca proporciona una interfaz de lenguaje de programación orientada a objetos para crear gráficos sin tener que saber acerca de VTK.

Para instalar el paquete desde la consola se debe hacer lo siguiente:

```
$ pip install mayavi
```