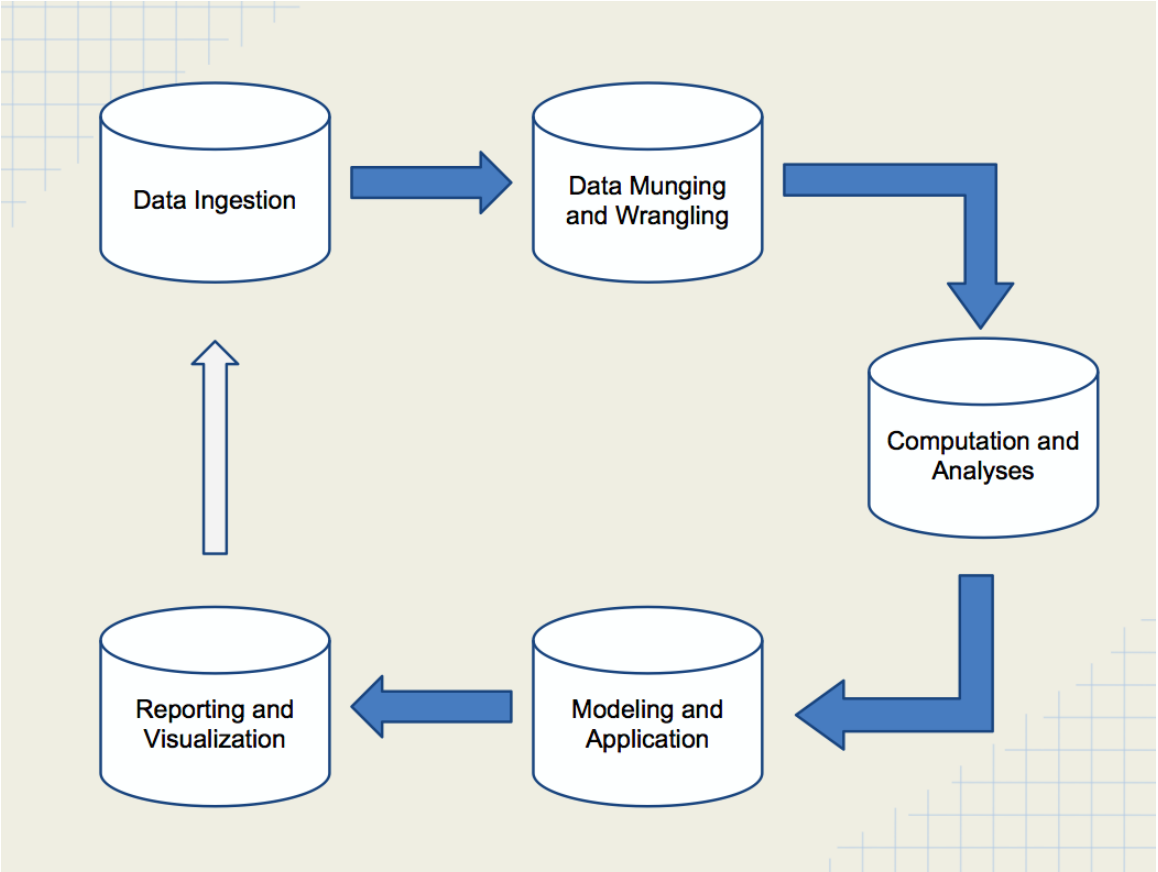
Archivos

Para trabajar con archivos o con ficheros hay muchas herramientas en python, inclusive algunas propiedad de google que permiten el contacto directo con el google drive y sus planillas de cálculo, pero sea como sea que se acceda a los archivos, estos son una herramienta fundamental en cualquier aplicación seria de análisis de datos. No se pueden leer en tiempo real los datos siempre, y los resultados no se pueden producir cada vez que se quiere visualizarlos.



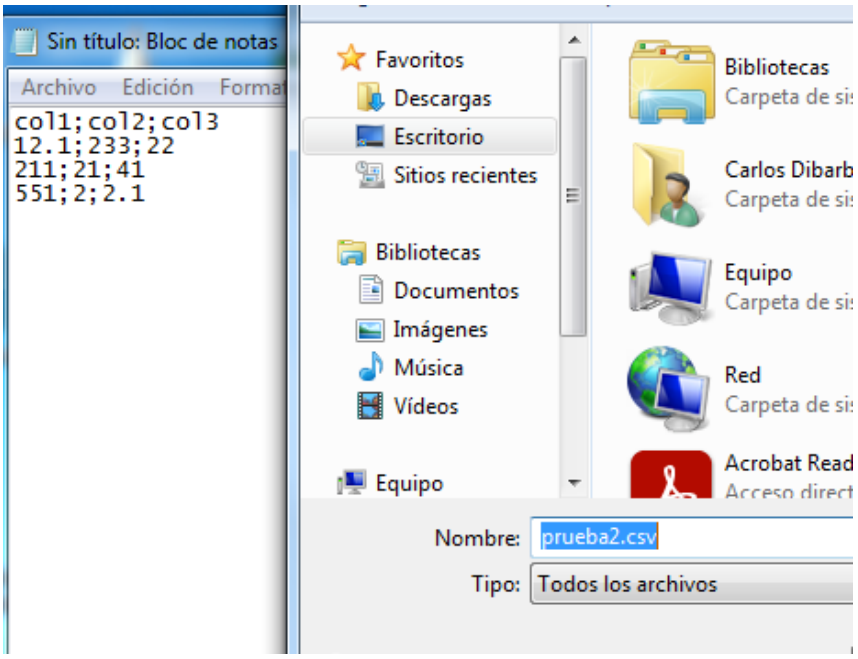
Esquema de procesamiento elemental de datos en ciencia de datos

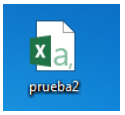
Nosotros en esta materia solo vamos a trabajar un tipo de archivo, el procesamiento que realizaremos será bastante básico y orientado a lo que nos pueda servir en Física, y cuando queramos exportar los datos ya procesados lo realizaremos utilizando el mismo tipo de archivo.

El formato que utilizaremos será el csv, que es un acrónimo de valores separados por coma. En realidad dependiendo de la convención los archivos pueden estar separados por comas o por “punto y coma”, pero es convencional también visualizar estos datos como tablas. Este formato de archivo no es el mejor para trabajar con datos, ya que en el más bajo nivel es un archivo de texto. Esto quiere decir que los caracteres están guardados en modo texto en su interior, y uno puede abrirlos y modificarlos con el bloc de notas u otros editores de

texto. Esto hace que el archivo sea más pesado que si uno guardará la información con una representación binaria, como archivos .dat, .raw o sin ir más lejos .xls. En estos archivos los datos están almacenados en forma de números, códigos, y letras, ordenados según una convención que conoce el fabricante, y para abrirse e interpretar su contenido se requiere el uso de programas especiales.

Ejemplo creación de csv: Crear archivo con el bloc de notas, y al guardarlo seleccionar "Tipo: Todos los archivos" y poner en su nombre una extensión .csv



El windows nos mostrará estos archivos de esta manera: 

Esto significa que de clickear en ellos, windows interpreta que es un archivo csv y que la manera de abrirlo y mostrarlo es con el MsExcel en forma de tablas.

Pandas y numpy

En python levantar un archivo es relativamente simple, aunque la dificultad mayor radica en trabajar con las estructuras que se realizan. Utilizaremos una biblioteca denominada "Pandas" que contiene todas las funciones y estructuras para leer, filtrar, ordenar y guardar datos. Además, algo que no es necesario pero que realizaremos será la conversión de las estructuras de datos de pandas (la serie de datos o dataseries y la tabla o dataframe) en vectores o arreglos, de forma de poder realizar el procesamiento de datos con las técnicas que hemos estudiado. Para esto, utilizaremos los vectores que se definen en la biblioteca numpy, por ser más versátiles que los vectores nativos de python.

Lectura de archivo: El siguiente código importa las bibliotecas necesarias y levanta el archivo ubicado en la carpeta "sample_data". Nosotros podemos subir otros archivos, o

copiarlos en otros lados, y leerlos desde python con esta función modificando la ruta al archivo.

```
import pandas as pd
import numpy as np

df=pd.read_csv("sample_data/california_housing_test.csv")
```

Este código lee el archivo completo y lo almacena en memoria, en la variable de nombre df, que corresponde a una tabla. Si quisiera levantar otro archivo que estuviera separado por ; en vez de , (por ejemplo) podría utilizar algo parecido a esto especificando el separador:

```
df=pd.read_csv("sample_data/prueba2.csv",";")
```

Si realizan una impresión podrán ver la estructura de la tabla. La tabla o dataframe es una colección de datos indexados. No es lo mismo que un vector. Este tipo de datos tiene un nombre o índice por columna, y un índice por fila que también puede nombrarse.

```
[17] print(df)

     col1  col2  col3
0    12.1   233  22.0
1   211.0    21  41.0
2   551.0     2   2.1
```

Por supuesto que se puede trabajar sobre esta estructura, aunque otra opción es grabar cada una de las series de datos en un vector independiente.
Lo primero se puede lograr así, donde se imprime toda la columna 2 con sus índices o solo el tercer elemento de la columna 1:

```
print(df["col2"]))
print(df["col1"][2])
```

Otra opción es copiar completamente Vamos a proceder de esta manera.
Sabiendo el nombre de las columnas (en este caso col1, col2…) podemos crear nuevos vectores con los datos extraídos de la tabla de la siguiente manera:

```
c1=df["col1"].to_numpy()
c2=df["col2"].to_numpy()
```

Buscar cómo hallar el valor medio, la desviación estándar, y el total para un vector numpy y probar estas funciones.

Con los valores en un vector también se puede realizar gráficos. El siguiente ejemplo realiza un gráfico de dispersión de puntos, y otro gráfico realizando interpolación entre los puntos y pintando la línea de color rojo

```
plt.figure(1)
plt.plot(c1,c2,".b")
plt.figure(2)
plt.plot(c2,c1,"r")
```

Si además se desea asignar título y etiquetas a los gráficos se pueden utilizar comandos como los siguientes.

```
plt.title("Gráfico de …")
plt.xlabel("Eje x en kg/h")
plt.ylabel("Eje y en toneladas métricas")
```

Ajuste de curvas

El último tema a trabajar en este texto y relacionado directamente con lo anterior, es el análisis más básico que podemos querer realizar como estudiantes de física sobre un conjunto de datos además de graficarlo. En general queremos conocer el ajuste de los datos contra una función particular que no necesariamente sea la línea. Para esto debemos importar la biblioteca correspondiente a la optimización y definir la función de prueba que queremos ajustar a nuestros datos. La función puede tener el nombre que deseemos, en este caso elegí test. La función recibe como parámetro la variable independiente y los parámetros que yo desee ajustar (que pueden ser tantos como desee. Yo elegí ajustar con una parábola que tenga solamente coeficiente principal a)

```
import scipy.optimize as sp

def test(x,a):
  return a*x**2
```

Una vez realizada la función de prueba se llamará a la función *curve_fit* pasándole 3 parámetros. La función a la que quiero ajustar, el conjunto o vector con la variable independiente y el vector con los datos de la variable dependiente. Esta función siempre arroja dos resultados. Un vector con los valores para los parámetros que ajustan la curva, y otro vector con las covarianzas de esos parámetros (la covarianza es una medida de la dispersión de los datos. En este caso se me está ofreciendo el parámetro que mejor ajusta y una medida del error para ese parámetro)

```
param, param_cov= sp.curve_fit(test, x, y)

print(param, param_cov)
```

Además de obtener los parámetros que estarán ubicados en orden en el vector `param[0]`, `param[1]`, etc… puede interesarme crear una gráfica con la dispersión de puntos, y otra con la curva “analítica” que ajusta a esos datos.
Para esto se puede graficar directamente:

```
plt.plot(x,y,".r")
plt.plot(x,test(x,param[0]))
```

La segunda línea grafica el vector x en el eje x, y en el eje y la imagen de la función test evaluada sobre el vector x y recibiendo como parámetro a el resultado del ajuste de curvas