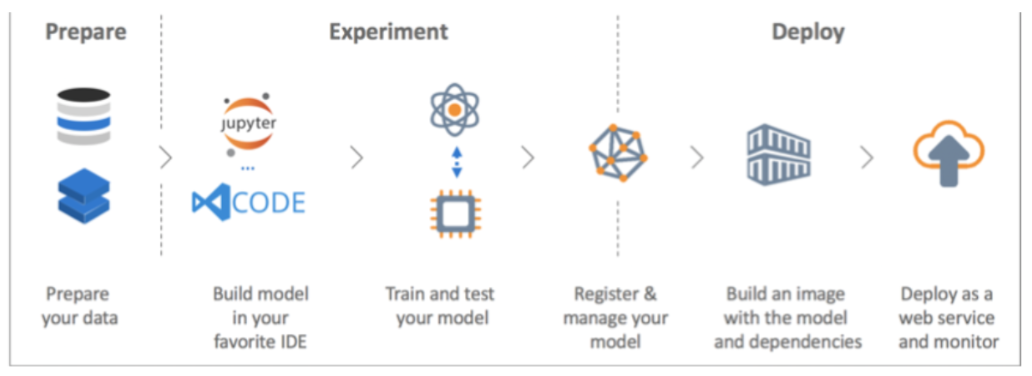
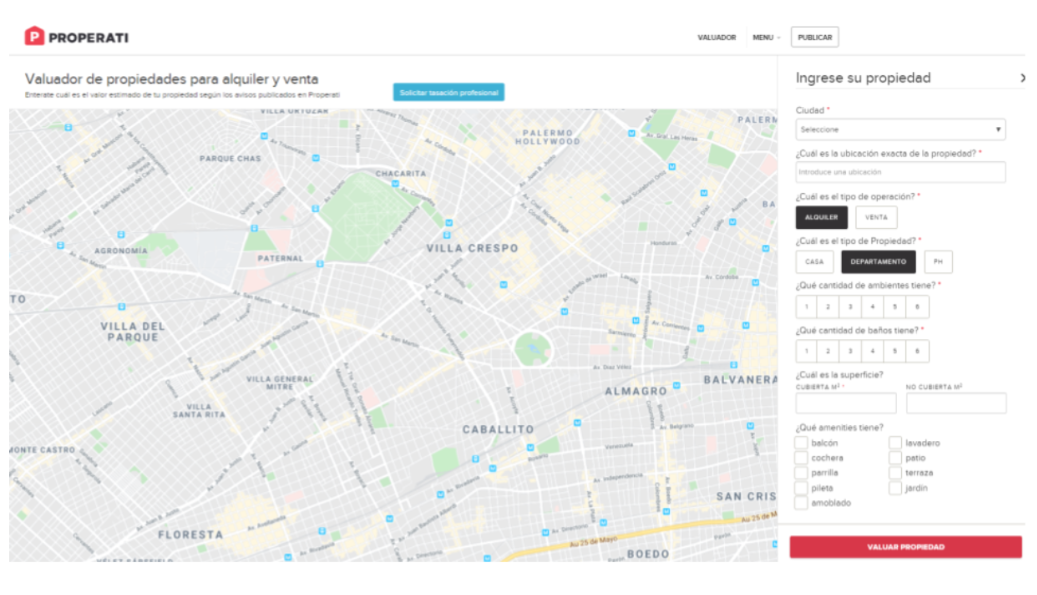
El **deployment** de un algoritmo de Machine Learning es la **integración del modelo en un entorno de producción existente**, el cual puede tomar una entrada y devolver una salida útil para tomar decisiones comerciales prácticas.



Vamos a ver 3 herramientas que nos ayudan a hacer esto: **Pickle, Shelve** y **Flask.**

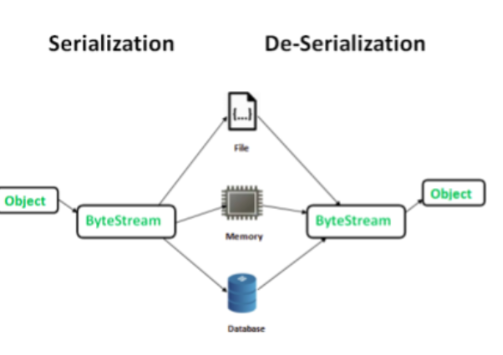
****

****

****

**Serialización**: A veces, necesitamos enviar un objeto Python de un programa a otro, a otra computadora o guardarlo para usarlo después, o ir grabando sus diferentes estados. Existen diferentes soluciones para hacer estas acciones, en función del tipo de objeto a enviar o guardar. Ejemplo: Tenemos un texto, el cual podemos representar en formato json o xml; podemos convertir un dataframe de Pandas en un archivo csv o Excel.

La **serialización** es una técnica donde se representan objetos a través de una codificación binaria. Entonces podemos codificar sobre objetos y estructuras de datos; y enviar un objeto serializado como un conjunto de bytes a través de la red para después guardarlo, cargarlo en la memoria y volver a usarlo pudiendo volver a transmitirlo si así lo necesitamos.



Podemos usar la **librería Pickle** de Python para serializar objetos. Al ser un estándar de Python, **no es compatible con otros lenguajes de programación**. Con pickle podemos serializar Eneros, Float, Tuplas, Listas, Diccionarios, Conjuntos, Funciones, Clases e Instancias de Clases. Pickle no es seguro contra información errónea o maliciosa. Entonces **conviene no deserializar información proveniente de fuentes desconocidas**.

**En Python:**

Para guardar dos listas en 2 archivos de Python:

import pickle

animales = [‘piton’,’mono’,’camello’]

alimentos = [‘vegetales’,’mani’,’agua’]

with open(‘../Data/animales.pkl’, ‘wb’) as f\_animales:

pickle.dump(animales, f\_animales)

with open(‘../Data/alimentos.pkl’, ‘wb’) as f\_alimentos:

pickle.dump(alimentos, f\_alimentos)

Suponiendo que ya tenemos los objetos serializados y lo que queremos hacer es levantarlos del disco, tenemos que hacer lo siguiente:

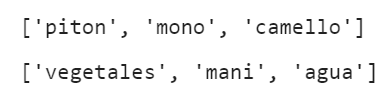
with open(‘../Data/animales.pkl’, ‘wb’) as f\_animales:

animales\_from\_disk = pickle.load(f\_animales)

with open(‘../Data/alimentos.pkl’, ‘wb’) as f\_alimentos:

alimentos\_from\_disk = pickle.load(f\_alimentos)

display(animales\_from\_disk, alimentos\_from\_disk)



**Shelve** es **otra biblioteca** que podemos usar **para serializar**. Extiende Pickle para poder realizar la serialización de una forma más clara y sencilla, en la cual vamos a poder acceder a la versión serializada de un objeto mediante un string asociada, a través de una estructura parecida a un diccionario. Entonces, **con Shelve podemos guardar en disco más de un objeto en el mismo archivo**. También **podemos acceder a cada uno de esos archivos** usando **claves** **como si fuera un diccionario**.

**En Python:**

Vamos a guardar 2 listas en un mismo archivo en disco, de forma tal que cada lista esté identificada con una clave:

import shelve

s = shelve.open(‘../Data/listas.db’)

s[‘animales’] = [‘piton’, ‘mono’, ‘camello’]

s[‘alimentos’] = [‘vegetales’, ‘mani’, ‘agua’]

s.close()

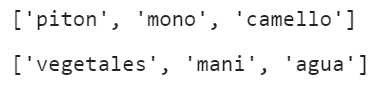
Ahora, si tenemos guardada una database y queremos levantarla del disco, lo vamos a hacer así:

s = shelve.open(‘../Data/listas.db’)

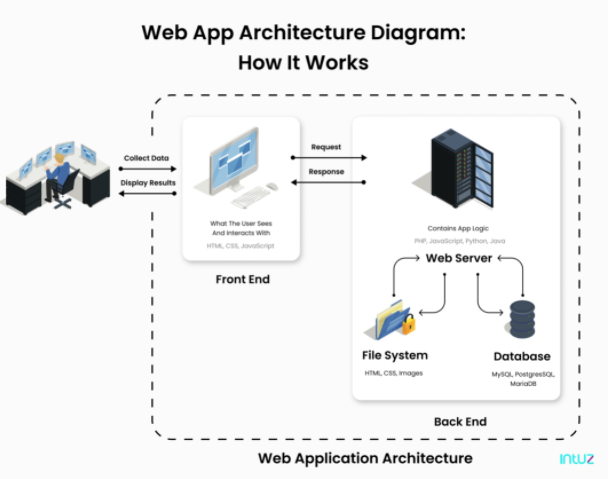
animales = s[‘animales’]

alimentos = s[‘alimentos’]

display(animales, alimentos)



**Flask:** Arquitectura de una aplicación web



**Flask** es una **API de Python** que nos permite crear aplicaciones web.

**Web Framework / Framework de Aplicaciones Web**: Es la **colección de módulos y bibliotecas que ayuda al desarrollador a recibir aplicaciones sin escribir los códigos de bajo nivel** (protocolos, administración de subprocesos, entre otros).

Con Flask, podemos tomar una función Python y asociarla a una URL a través del Routing usando el diseño REST. Con Flask podemos crear APIs que al recibir un request con los datos de las features de una instancia puedan devolver la predicción de un modelo para esta instancia.

**Repaso de URL:**

****

* **Protocolo**: indica el protocolo que será utilizado para acceder (HTTP, FTP, HTTPS)
* **Host**: Indica cómo encontrar en la red el servidor que tiene el recurso.
* **Path**: Indica la ruta para localizar el recurso dentro del servidor.
* **Query**: Indica cuál es la consulta que se está realizando.

Sobre este recurso se puede definir un método http, tal como:

* **GET** Se solicita una representación de un recurso específico (sólo se recuperan datos).
* **PUT** Reemplaza la representación de los datos en la petición.
* **POST** Se usa para enviar una entidad a un recurso (puede generar un cambio).
* **DELETE** Borra la representación de un recurso.

Ejemplo **en Python**: Vamos a crear una aplicación web que exponga la función **mi\_saludo** como url http:/localhost:5010/hola

Podemos acceder a este recurso con los métodos GET y POST.

from flask import Flask

app = Flask(‘my\_first\_webapp’)

@app.route(‘/hola’, methods = [‘GET’, ‘POST’])

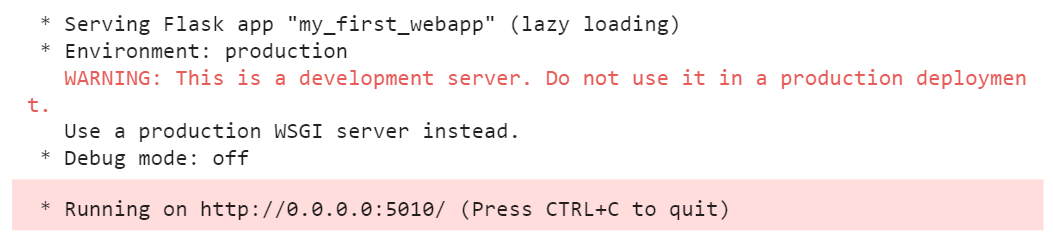
def mi\_saludo():

texto = ‘¡Hola Mundo!’

return texto

Vamos a iniciar la aplicación en el puerto 5010:

app.run (host=’0.0.0.0.’, port = 5010)



Abrimos en un browser esta url (GET)



**Conclusión:**

Aprendimos sobre serialización y representación de recursos en una aplicación web con Python usando Flask, Pickle y Shelve.

Esto nos sirve para entender cómo se realiza la puesta en producción de un modelo.