The IUCN Red List

Una panoramica sulle specie in via di estinzione con Python

ad opera di **G. Z.**

Prima di iniziare

- Assicurarsi di possedere tutti i file necessari nella stessa cartella.
- Accertarsi di avere una versione Python uguale o superiore a 3.10 installata.
- Installare `pandas`, `matplotlib` e `jupyter` nel proprio ambiente di lavoro.
  A tal fine, eseguire `pip install -r requirements.txt` da terminale o scaricare manualmente i pacchetti.
- Il notebook fa uso di HTML. Per una corretta visualizzazione, usare un applicativo adatto (ad esempio Visual Studio Code).
  In alternativa, eseguire `jupyter notebook redlist.ipynb` da terminale.
- Attivare una connessione di rete funzionante.
- Buona lettura!

Cos'è la Lista Rossa IUCN?

L'International Union for Conservation of Nature (IUCN) è un'organizzazione parzialmente governativa ed è la massima autorità nel monitoraggio e nella salvaguardia dei beni naturali.[1]
Essa mette a disposizione un ampio catalogo con oltre 150'000 specie viventi, oltre a fornire descrizioni dettagliate sulle caratteristiche fisiche e tassonomiche della specie, sull'areale in cui vive e sui fattori attuali o potenziali che la minacciano.
Questo elenco è noto con il nome di Lista Rossa (in inglese, "Red List") ed è la raccolta più esaustiva esistente sullo stato di conservazione delle specie, distinte in base al grado di minaccia in[2]:

- dati insufficienti (DD - Data Deficient)
- a rischio minimo (LC - Least Concern)
- prossimo alla minaccia (NT - Near Threatened)
- vulnerabile (VU - Vulnerable)
- in pericolo (EN - Endangered)
- in pericolo critico (CE - Critically Endangered)
- estinto in natura (EW - Extinct In The Wild)
- estinto (EX - Extinct)
- non valutata (NE - Not Evaluated)

Si noti che le specie non valutate, a differenza di quelle i cui dati sono insufficienti, non sono contenute nella Lista Rossa, che, d'altro canto, contiene una percentuale minima del numero di specie esistenti, stimato a svariati milioni.
In questa sede, inoltre, per motivi didattici, ci si limiterà ad analizzare solo gli animali vertebrati, o, quantomeno, un loro sottoinsieme di 60'000 specie.

Tutte le risorse dati che verranno utilizzate sono state reperite presso il sito ufficiale della IUCN Red List.

Vediamo ora come manipolare queste informazioni con Python.

Creazione del DataFrame con gli animali

Le informazioni necessarie per cominciare sono contenute nel file `simple_summary.csv`.

Leggiamole in un DataFrame[3] della libreria pandas.

In [ ]:
```
import pandas as pd

species = pd.read_csv("simple_summary.csv")
species.head()
```

Out[ ]:

	assessmentId	internalTaxonId	scientificName	kingdomName	phylumName	orderName	className	familyName	genusName	speciesName	infraType	infraName	infraAutho
0	495630	10030	Hexanchus griseus	ANIMALIA	CHORDATA	HEXANCHIFORMES	CHONDRICHTHYES	HEXANCHIDAE	Hexanchus	griseus	NaN	NaN	
1	495907	10041	Heosemys annandalii	ANIMALIA	CHORDATA	TESTUDINES	REPTILIA	GEOEMYDIDAE	Heosemys	annandalii	NaN	NaN	
2	497499	132523146	Hubbsina turneri	ANIMALIA	CHORDATA	CYPRINODONTIFORMES	ACTINOPTERYGII	GOODEIDAE	Hubbsina	turneri	NaN	NaN	
3	498370	10767	Ictalurus australis	ANIMALIA	CHORDATA	SILURIFORMES	ACTINOPTERYGII	ICTALURIDAE	Ictalurus	australis	NaN	NaN	
4	498476	10769	Ictalurus mexicanus	ANIMALIA	CHORDATA	SILURIFORMES	ACTINOPTERYGII	ICTALURIDAE	Ictalurus	mexicanus	NaN	NaN	

Per eliminare le informazioni superflue, mancanti, ridondanti, o comunque poco interessanti in questo contesto, usiamo il metodo `drop` della classe `DataFrame`.

In [ ]:
```
species.drop(columns=['infraType', 'infraName', 'infraAuthority',
             'redlistCriteria', 'criteriaVersion', 'speciesName', 'scopes'], inplace=True)

species.head()
```

Out[ ]:

	assessmentId	internalTaxonId	scientificName	kingdomName	phylumName	orderName	className	familyName	genusName	authority	redlistCategory	populationTrend
0	495630	10030	Hexanchus griseus	ANIMALIA	CHORDATA	HEXANCHIFORMES	CHONDRICHTHYES	HEXANCHIDAE	Hexanchus	(Bonnaterre, 1788)	Near Threatened	Decreasing
1	495907	10041	Heosemys annandalii	ANIMALIA	CHORDATA	TESTUDINES	REPTILIA	GEOEMYDIDAE	Heosemys	(Boulenger in Annandale & Robinson, 1903)	Critically Endangered	Decreasing
2	497499	132523146	Hubbsina turneri	ANIMALIA	CHORDATA	CYPRINODONTIFORMES	ACTINOPTERYGII	GOODEIDAE	Hubbsina	(de Buen, 1940)	Critically Endangered	Decreasing
3	498370	10767	Ictalurus australis	ANIMALIA	CHORDATA	SILURIFORMES	ACTINOPTERYGII	ICTALURIDAE	Ictalurus	(Meek, 1904)	Data Deficient	Decreasing
4	498476	10769	Ictalurus mexicanus	ANIMALIA	CHORDATA	SILURIFORMES	ACTINOPTERYGII	ICTALURIDAE	Ictalurus	(Meek, 1904)	Vulnerable	Unknown

Ora si vuole effettuare la ricerca di uno specifico animale. Ad esempio dell'orso bruno (*Ursus arctos*).

In [ ]:
```
species.loc[species['scientificName'] == 'Ursus arctos']
```

Out[ ]:

	assessmentId	internalTaxonId	scientificName	kingdomName	phylumName	orderName	className	familyName	genusName	authority	redlistCategory	populationTrend
43659	121229971	41688	Ursus arctos	ANIMALIA	CHORDATA	CARNIVORA	MAMMALIA	URSIDAE	Ursus	Linnaeus, 1758	Least Concern	Stable

Costruzione di una classe `Animal`

Per agevolare le operazioni successive, creiamo una classe `Animal` che prende come parametro di inizializzazione il nome scientifico della specie ed in base ad esso crea i dovuti attributi.

Per accedere ad ulteriori informazioni sulla specie, si può far uso degli attributi `assessmentId` e `internalTaxonId` per accedere alla pagina web, per come essa è strutturata, ovvero `https://www.iucnredlist.org/species/[internalTaxonId]/[assessmentId]`.

In [ ]:
```
WEBSITE = "https://www.iucnredlist.org"

class Animal:
    def __init__(self, name: str):
        self.name = name.capitalize()
        self.info = species.loc[species['scientificName'] == self.name]
        self.is_listed = False if len(self.info) == 0 else True
        if self.is_listed:
            self.kingdom = self.info.loc[self.info.index[0], 'kingdomName']
            self.phylum = self.info.loc[self.info.index[0], 'phylumName']
            self.classis = self.info.loc[self.info.index[0], 'className'] # "class" avrebbe provocato conflittualità
            self.order = self.info.loc[self.info.index[0], 'orderName']
            self.family = self.info.loc[self.info.index[0], 'familyName']
            self.genus = self.info.loc[self.info.index[0], 'genusName']
            self.authority = self.info.loc[self.info.index[0], 'authority']
            self.status = self.info.loc[self.info.index[0], 'redlistCategory']
            self.trend = self.info.loc[self.info.index[0], 'populationTrend']
            self.assessmentId = self.info.loc[self.info.index[0], 'assessmentId']
            self.internalTaxonId = self.info.loc[self.info.index[0], 'internalTaxonId']
            self.url = f"{WEBSITE}/species/{self.internalTaxonId}/{self.assessmentId}"


    def __str__(self) -> str:
        if self.is_listed:
            return (
                f"Name: {self.name} ({self.authority})\n"
                f"Class: {self.classis}\n"
                f"Order: {self.order}\n"
                f"Family: {self.family}\n"
                f"Conservation: {self.status} ({self.trend})\n"
                f"Info: {self.url}"
            )
        return f"{self.name} (not listed)"

animal = Animal("Ursus arctos")
print(animal)
```

```
Name: Ursus arctos (Linnaeus, 1758)
Class: MAMMALIA
Order: CARNIVORA
Family: URSIDAE
Conservation: Least Concern (Stable)
Info: https://www.iucnredlist.org/species/41688/121229971
```

Interazione con altri DataFrame

Cercare un animale tramite il solo nome scientifico può risultare complicato, perché non sempre è noto all'utente.
Di solito è più semplice conoscere il nome comune. A questo scopo il dataset `common_names.csv` può essere d'aiuto.

In [ ]:
```
names = pd.read_csv("common_names.csv")
names.head()
```

Out[ ]:

	internalTaxonId	scientificName	name	language	main
0	10030	Hexanchus griseus	Bluntnose Sixgill Shark	English	True
1	10041	Heosemys annandalii	Yellow-headed Temple Turtle	English	True
2	132523146	Hubbsina turneri	Mexclapique Michoacano	Spanish; Castilian	False
3	132523146	Hubbsina turneri	Highland Splitfin	English	True
4	10767	Ictalurus australis	Bagre del Panuco	Spanish; Castilian	False

Per accedere alla nomenclatura inglese dell'animale è consigliabile usare la colonna `main` e non `language`.
È intuitivo capire il motivo eseguendo la linea di codice qui sotto:

In [ ]:
```
names.loc[names['scientificName'] == 'Ursus arctos']
```

Out[ ]:

	internalTaxonId	scientificName	name	language	main
84342	41688	Ursus arctos	Grizzly Bear	English	False
84343	41688	Ursus arctos	Oso Pardo	Spanish; Castilian	False
84344	41688	Ursus arctos	Ours brun	French	False
84345	41688	Ursus arctos	Brown Bear	English	True

`main` ha valore True in corrispondenza del nome canonico (in questo caso 'Brown Bear'), False altrimenti, mentre con `language` è possibile trovare sinonimi dell'animale, anche in inglese ('Grizzly Bear'), dunque non è un indicatore univoco.

In [ ]:
```
more_info = names.query("main and scientificName == \'Ursus arctos\'")
# scrittura compatta di
# more_info = names.loc[(names['main']) & (names['scientificName'] == 'Ursus arctos')]

# more_info = names.query("language == \'English\' and scientificName == \'Ursus arctos\'")
# restituirebbe un DataFrame a più righe, con 'Grizzly Bear' come prima occorrenza

animal.vernacular = more_info.loc[more_info.index[0], 'name']
animal.vernacular
```

Out[ ]:
```
'Brown Bear'
```

Può tornare estremamente utile effettuare l'operazione inversa, ossia ricavare il nome scientifico a partire dal nome comune, operazione che non verrà eseguita in questa trattazione, ma disponibile sull'applicativo finale `launcher.py`.

Si noti invece come tra i file nella cartella sia presente anche un file più pesante degli altri denominato `assessments.csv`, contenente molte altre informazioni aggiuntive, come la descrizione, la distribuzione e l'habitat, e le minacce a cui è sottoposto.

In [ ]:
```
# Essendo assessmentId univoco lo si può utilizzare come chiave
assessments = pd.read_csv("assessments.csv", index_col = 'assessmentId')
assessments.head()
```

Out[ ]:

assessmentId	internalTaxonId	scientificName	redlistCategory	redlistCriteria	yearPublished	assessmentDate	criteriaVersion	language	rationale	habitat	...	populat
495630	10030	Hexanchus griseus	Near Threatened	A2bd	2020	2019-11-21 00:00:00 UTC	3.1	English	<p>The Bluntnose Sixgill Shark (Hexan...	<p>The B...	...	D
495907	10041	Heosemys annandalii	Critically Endangered	A2cd+4cd	2021	2018-03-13 00:00:00 UTC	3.1	English	<p>Heosemys annandalii is considered ...	<p>Heosemys annandalii inhabits lowla...	...	D
497499	132523146	Hubbsina turneri	Critically Endangered	B1ab(i,ii,iii,iv)+2ab(i,ii,iii,iv)	2019	2018-04-17 00:00:00 UTC	3.1	English	The Highland Splitfin is now only known to be ...	<p>This species lives in quiet waters with cur...	...	D
498370	10767	Ictalurus australis	Data Deficient	NaN	2019	2018-12-06 00:00:00 UTC	3.1	English	<p>Ictalurus australis occurs in east...	<p>I. australisinhabits quiet and sl...	...	D
498476	10769	Ictalurus mexicanus	Vulnerable	D2	2019	2018-12-06 00:00:00 UTC	3.1	English	I. mexicanusis herein categorized as...	<p>Specifics details are unknown for this spec...	...	

5 rows × 22 columns

Volendo, è possibile creare una pagina html per visualizzare in modo comprensivo queste informazioni, senza dover accedere al sito.
Ad esempio:

In [ ]:
```
far_more_info = assessments.loc[animal.assessmentId]

html_content = (
    f"<main = style=\"background-color: #444444; color: white;\">"
    f"<h1>{animal.vernacular}</h1>"
    f"<h2><i>{animal.name}</i></h2>"
    f"<h2>Assessment</h2>"
    f"<p><b>{animal.status}</b> ({animal.trend})."
    f"{far_more_info['rationale']}</p>"
    f"</main>"
)

### Queste due linee di codice permettono un output per ipynb.
from IPython.display import display, HTML
display(HTML(html_content))
```

Brown Bear

Ursus arctos

Assessment

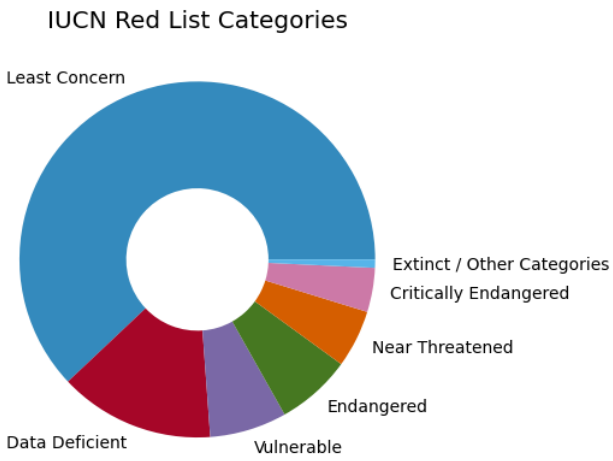
Least Concern (Stable).

The range of the Brown Bear has historically declined in North America, Europe, and Asia, and the species has been extirpated in North Africa. However, it remains widespread across three continents, and is still one of the world's most widely distributed terrestrial mammals. Globally the population remains large, and is not significantly declining and may be increasing in some areas (Swenson *et al.* 1998, Schwartz *et al.* 2006, Mace *et al.* 2012, Kaczensky *et al.* 2013, Chapron *et al.* 2014). There are many small, isolated subpopulations that are in jeopardy of extirpation, but others, under more protection and management, are expanding.

Rappresentazioni grafiche

Finora ci si è limitati ad accedere e visualizzare i dati testuali, ma `pandas.DataFrame` fornisce molti metodi compatibili con la libreria `matplotlib` che permettono di avere un'analisi grafica delle informazioni contenute in tabella.

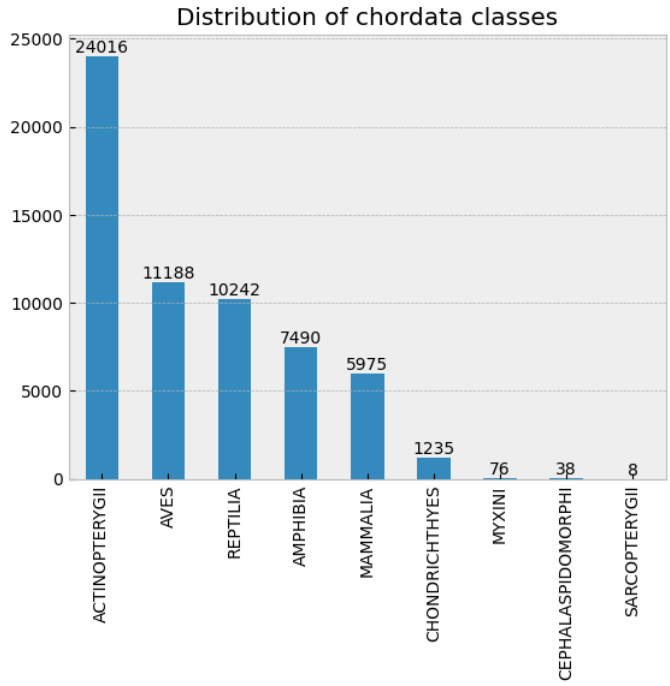
In [ ]:
```
import matplotlib.pyplot as plt
plt.style.use('bmh')

tmp = species.value_counts('redlistCategory', normalize = True)

### unisco le voci minori sotto un'unica categoria
gt_percentile = tmp[tmp>=0.01]
gt_percentile['Extinct / Other Categories'] = tmp[tmp<0.01].sum()
###

gt_percentile.plot.pie(title = 'IUCN Red List Categories', ylabel='', wedgeprops=dict(width=0.6))
plt.show()
```



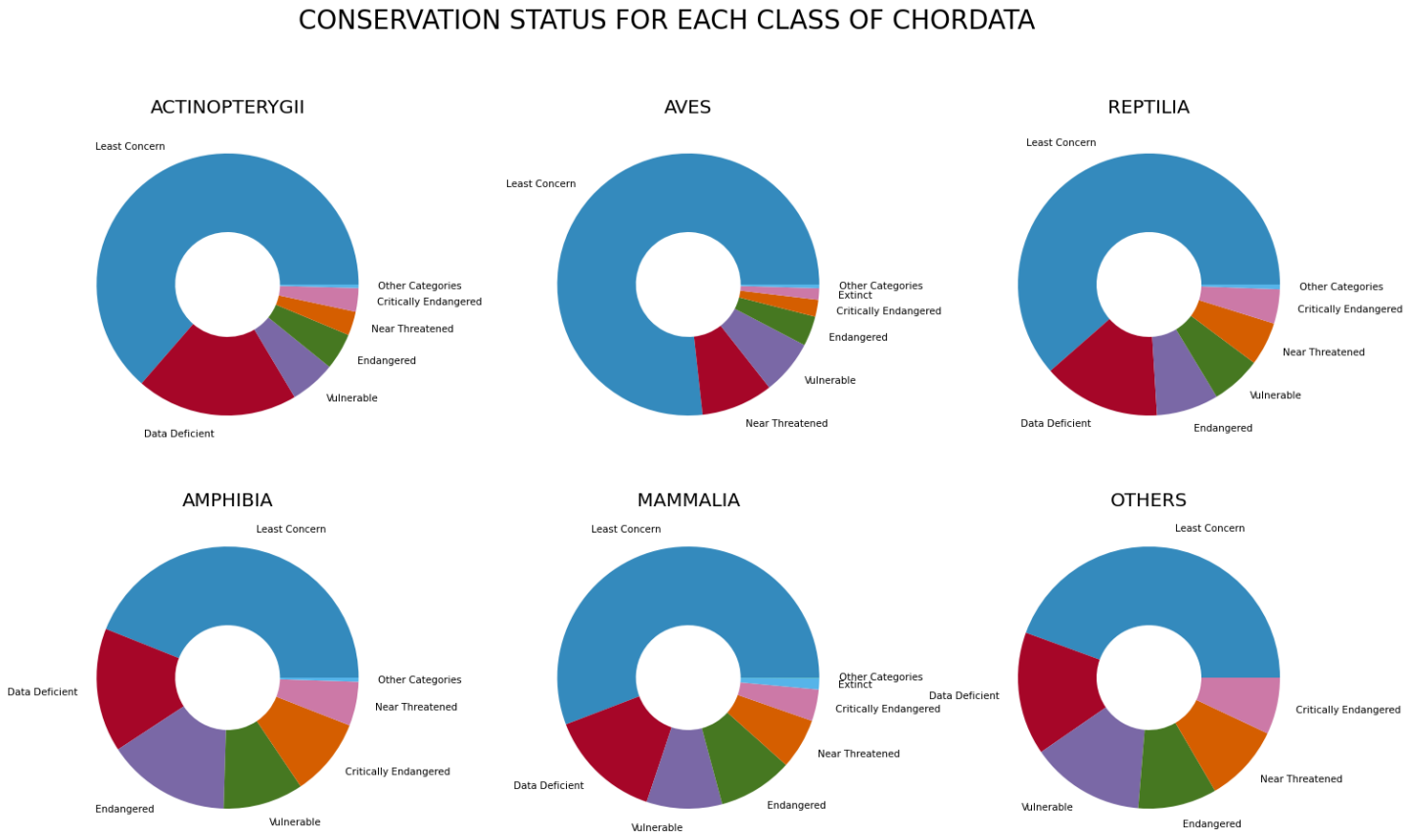
Grazie a questo areogramma, si può subito intuire come quasi un quarto delle specie sia esposto a un qualche livello di minaccia.

Nelle successive due caselle di codice si prova a complicare leggermente questa operazione effettuando una distinzione tra le classi.

In [ ]:
```
### Diagramma a barre
tmp = species.value_counts('className')
fig = tmp.plot.bar(title = 'Distribution of chordata classes', xlabel='')
fig.bar_label(fig.containers[0]) # scrive il valore sopra la rispettiva barra
fig.grid(axis = 'x')
plt.show()
```



In [ ]:
```
### Areogrammi "a ciambella"

### riunisco le classi minori in un'unica categoria 'OTHERS'
sel = species[['className', 'redlistCategory']].copy()
tmp = sel.value_counts('className')
gt1300 = tmp[tmp>=1300]
gt1300['OTHERS'] = tmp[tmp<1300].sum()
sel.loc[~(sel['className'].isin(gt1300.index)), 'className'] = 'OTHERS'
###

fig, axes = plt.subplots(2,3)
plt.suptitle('CONSERVATION STATUS FOR EACH CLASS OF CHORDATA', size = 20, y=1)

from itertools import product #prodotto cartesiano

# itero sulle classi di animale e sulle posizioni dei
# grafici nella figura di output (0,0), (0,1), ..., (1,2)
for cls, pos in zip(gt1300.index, product((0,1),(0,1,2))):

    tmp = sel.loc[sel['className'] == cls]
    tmp = tmp.value_counts('redlistCategory', normalize = True)
    gt_percentile = tmp[tmp>=0.01]
    gt_percentile['Other Categories'] = tmp[tmp<0.01].sum()

    # creazione del del subplot relativo alla classe su cui si sta iterando
    gt_percentile.plot.pie(title = cls, ylabel = '', ax = axes[*pos],
                           figsize = (18, 10), fontsize = 7.5,
                           labeldistance = 1.15, wedgeprops = dict(width=0.6))
```



L'albero tassonomico

Nel 1700 il naturalista svedese Carlo Linneo, propose di catalogare piante e animali secondo principi rigorosi, introducendo di fatto quella che è la sistematica moderna.
Questa classificazione delle creature è di natura gerarchica, in virtù della quale è possibile affermare che una specie animale sia il ramo terminale o foglia (escludendo eventuali ulteriori ramificazioni quali sottospecie o popolazioni) di una struttura ad albero, solitamente non dicotomica, così strutturata:
Regno > Phylum (o Tipo) > Classe > Ordine > Famiglia > *Genere* > *Specie*,
con eventuali ripartizioni intermedie.
Riportando il solito esempio dell'orso bruno, esso è scientificamente così classificato:
Animalia > Chordata > Mammalia > Carnivora > Ursidae > *Ursus* > *Ursus arctos*.

L'albero è stato implementato nella classe `TaxonTree` del modulo `trees.py`.
Questo albero eredita molti metodi e attributi da una struttura più generica `Tree`, la cui caratteristica più interessante è quella di poter inserire un ramo in tempo O(log N) grazie all'albero di ricerca binaria in cui sono disposti i figli, mantenendo dunque intatto il loro ordinamento, e di conseguenza permettendo l'estrazione di un ramo figlio sempre in tempo logaritmico, tramite ricerca binaria.
Di conseguenza la complessità computazionale per la creazione dell'albero è O(NlogN), al posto di $O(N^2)$ qualora si fosse scelto di utilizzare una lista classica. Per N > 60'000, come in questo caso, si tratta di un risparmio significativo.
Anche l'utilizzo di un insieme o un dizionario sarebbe stata una valida alternativa, tuttavia in fase di stampa dovrebbe comunque essere convertito in una lista ed ordinato, impiegando in ogni caso un tempo O(NlogN), oltre ad essere didatticamente meno attraente di un *binary search tree*, implementato in maniera assai simile a quella presentata nel libro *Data Structures and Algorithms with Python* di K.D. Lee e S. Hubbard[4].

Qui sotto viene data una dimostrazione di come agisce la classe.

In [ ]:
```python
# i moduli completi del progetto, precedentemente solo abbozzati
from trees import TaxonTree
from animals import Animal as An # evita conflitti con la classe Animal dichiarata localmente
An.species = species
An.names = names

animals = [
            An('Panthera tigris'),          # tigre
            An('Istiophorus platypterus'),  # pesce vela
            An('Panthera leo'),             # leone
            An('Puma concolor'),            # puma
            An('Panthera Leo'),             # leone (duplicato)
            An('Makaira nigricans')         # marlin azzurro
          ]

print('Length of animals list:', len(animals))
taxontree = TaxonTree('ANIMALIA')
for animal in animals:
    taxontree.add_animal(animal) # non aggiunge duplicati!

taxontree.print() # visita in pre-ordine: prima la radice e poi i figli, in ordine alfabetico
print('There are only 5 different species in the tree!')
```

```
Length of animals list: 6
ANIMALIA
  CHORDATA
    ACTINOPTERYGII
      PERCIFORMES
        ISTIOPHORIDAE
          Istiophorus
            Istiophorus platypterus
          Makaira
            Makaira nigricans
    MAMMALIA
      CARNIVORA
        FELIDAE
          Panthera
            Panthera leo
            Panthera tigris
          Puma
            Puma concolor

There are only 5 different species in the tree!
```

Menù dinamico

L'interfaccia con il menù contestuale è gestita da una classe `Menu`, che si basa su un dizionario di dizionari, nonché della caratteristica di Python di trattare le funzioni come oggetti.
La voce principale del dizionario è il "tasto" che va premuto dall'utente, a cui sono associati una stringa di descrizione e la funzione che l'utente vuole chiamare.

In [ ]:
```python
class Menu():
    def __init__(self, title:str = '', entries: list[tuple[str|int, str, callable]] = None):
        self.title = title
        self._menu = {}
        if entries: # permette di convertire una lista di tuple (indice, etichetta, azione) in un menù
            for idx, label, action in entries:
                self.add_action(idx, label, action)

    def add_action(self, idx: str|int, label: str, action: callable):
        self._menu[idx] = {'label' : label, 'action': action}

    def execute(self, idx: str|int) -> None:
        if self._menu.get(idx): # get serve a non alzare KeyError quando viene passata una chiave inesistente
            self._menu.get(idx)['action']() # le parentesi tonde finali permettono la chiamata della funzione
            # nelle parentesi tonde si possono anche passare dei parametri, se necessari

    def __str__(self) -> str:
        string = f"\n{self.title}\nActions:\n"
        for key, value in self._menu.items():
           string += f"   {key} - {value['label']}\n"
        return string

def fun1():
    print('fun1 works!')
def fun2():
    print('fun2 works!')
actions = [('1', 'call fun1', fun1),
           ('2', 'call fun2', fun2)]
```

```
menu = Menu('My Menu', actions)
print(menu)
choice = input('Enter choice: ')
menu.execute(choice)
```

```
My Menu
Actions:
   1 - call fun1
   2 - call fun2

fun1 works!
```

Uso e struttura dell'applicativo

Associati a questo notebook, sono presenti anche alcuni file Python, dei quali `launcher.py` è il pannello di controllo, ossia quello che l'utente dovrebbe lanciare.
In esso vengono invocate tutte funzionalità presentate fino ad ora, in particolare:
`menu.py` predispone di una classe `Menu` che semplifica la struttura del launcher;
`animals.py` contiene la classe `Animal`, in cui i DataFrame vengono assegnati come variabili di classe, per ottimizzare i tempi e lo spazio di lettura dei file csv. La classe contiene inoltre i metodi per istanziare un oggetto indifferentemente se a partire dal nome comune o dal nome scientifico;
`explore.py` contiene le funzioni per creare, salvare ed aprire una pagina html con le informazioni contenute nel DataFrame `assessments` relative all'animale passato come parametro, e una funzione per aprire semplicemente sul proprio browser predefinito il sito internet dell'animale in questione;
`downloader.py` fa uso di `urllib.request` della libreria standard per provare a cercare l'immagine dell'animale online e scaricarne i byte che la compongono, trasponendoli su un file di output jpg tramite scrittura binaria. In caso di eventuali errori in fase di download o salvataggio, le eccezioni vengono gestite senza interrompere il programma;
`charts.py` crea i tre grafici mostrati nella sezione precedente, a cui si aggiunge una ricerca per classe;
`trees.py` dispone di un contenitore `TaxonTree` adatto al salvataggio e alla stampa di un albero tassonomico.

Conclusioni

Questo progetto nasce da un personale interesse verso le specie animali e la biodiversità in generale, che mi ha condotto ad approfondire questi temi con l'analisi dati in Python, in parte anche a dimostrazione di come la programmazione e l'elaborazione di strutture dati siano applicabili ai più disparati settori di studio e non solo.
Spero inoltre di essere riuscito ad intrattenere il lettore e/o l'utilizzatore con un'opera gradevole ed intrigante, che comunichi una gamma di informazioni relativamente ampia sull'utilizzo di dati aggregati e formalmente esatta e coerente per quanto concerne il codice, ma che possa al contempo far scaturire una nuova consapevolezza sulla ricchezza della diversità biologica presente sul nostro Pianeta, e su come una rilevante porzione di essa sia vicina alla scomparsa, senza che mai nessuno, spesso neppure il mondo della biologia, venga a conoscenza dell'esistenza stessa di moltissime specie che ne fanno parte.

Riferimenti

1. IUCN. 2022. International Union for Conservation of Nature. https://www.iucn.org. Consultato il 18 aprile 2023.
2. IUCN. 2022. The IUCN Red List of Threatened Species. Version 2022-2. https://www.iucnredlist.org. Consultato il 18 aprile 2023.
3. DataFrame - pandas 2.0 documentation, su https://pandas.pydata.org/docs/reference/frame.html.
4. K. D. Lee and S. Hubbard. "6.5.1 The BinarySearchTree Class". In *Data Structures and Algorithms with Python*, 170-176. Springer, 2015.