Task 1: Data collection and cleaning

(1) - Collect and extract the data from the relevant website;

Firstly, we import the needed library and set the %matplotlib inline command.

Then, we read the excel file by using pd.read_csv("iucn-animalia.csv").

```
# (1) - Collect and extract the data from the relevant website;
### Your code here ###
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df=pd.read_csv("iucn-animalia.csv")
df
```

Output:

	Name	EX	EW	Subtotal (EX+EW)	CR(PE)	CR(PEW)	Subtotal (EX+EW+ CR(PE)+CR(PEW))	CR	EN	VU	Subtotal (threatened spp.)	LR/cd	NT or LR/nt	LC or LR/lc	DD	Total
0	ACTINOPTERYGII	82	11	93	140	8	241	739	1,210	1,410	3,359	0	765	16,556	4,902	25,675
1	AMPHIBIA	37	2	39	184	1	224	795	1,265	816	2,876	0	431	3,763	911	8,020
2	ANTHOZOA	0	0	0	2	0	2	26	27	203	256	0	180	280	135	851
3	ARACHNIDA	9	0	9	21	0	30	76	116	76	268	0	25	222	67	591
4	ASTEROIDEA	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1
5	AVES	159	5	164	22	0	186	232	405	717	1,354	0	940	8,699	40	11,197
6	BIVALVIA	32	0	32	15	0	47	84	71	61	216	4	57	340	169	818
7	BRANCHIOPODA	0	0	0	0	0	0	8	13	22	43	1	1	1	1	47
8	CALCAREA	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2
9	CEPHALOPODA	0	0	0	0	0	0	1	2	2	5	0	2	324	419	750
10	CHILOPODA	0	0	0	2	0	2	3	6	1	10	0	0	1	0	11
11	CHONDRICHTHYES	1	0	1	2	0	3	91	123	184	398	0	127	543	172	1,241
12	CLITELLATA	2	0	2	4	0	6	7	16	12	35	0	20	108	192	357
13	COLLEMBOLA	0	0	0	2	0	2	3	4	2	9	0	10	6	3	28
14	DEMOSPONGIAE	0	0	0	0	0	0	0	0	0	0	0	0	1	21	22
15	DIPLOPODA	3	0	3	4	0	7	37	32	17	86	0	66	52	43	250
16	ECHINOIDEA	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
17	GASTROPODA	265	16	281	137	1	419	643	548	995	2,186	0	682	2,692	1,647	7,488
18	HEXANAUPLIA	1	0	1	0	0	1	1	0	24	25	8	0	0	4	38
19	HOLOTHUROIDEA	0	0	0	0	0	0	0	7	9	16	0	0	111	244	371
20	HYDROZOA	0	0	0	1	0	1	5	0	4	9	0	0	5	1	15
21	INSECTA	60	1	61	93	0	154	429	978	954	2,361	3	735	6,155	3,253	12,568
22	MALACOSTRACA	7	1	8	18	1	27	141	162	311	614	0	76	1,206	1,131	3,035
23	MAMMALIA	85	1	86	29	0	115	235	550	554	1,339	0	378	3,340	837	5,980
24	MAXILLOPODA	1	0	1	0	0	1	6	0	47	53	0	0	0	19	73
25	MEROSTOMATA	0	0	0	0	0	0	0	1	1	2	0	0	0	2	4
26	MONOPLACOPHORA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
27	MYXINI	0	0	0	0	0	0	1	2	6	9	0	2	36	33	80
28	NEMERTEA	1	0	1	1	0	2	1	1	1	3	0	0	1	1	6
29	OSTRACODA	2	0	2	0	0	2	2	0	9	11	0	0	0	1	14
30	PETROMYZONTI	0	0	0	1	0	1	2	4	3	9	0	3	24	2	38
31	POLYCHAETA	0	0	0	0	0	0	1	0	0	1	0	0	0	1	2
32	POLYPLACOPHORA	0	0	0	0	0	0	0	1	0	1	0	1	0	0	2
33	REPTILIA	32	2	34	48	0	82	434	789	625	1,848	2	566	6,312	1,492	10,254
34	SARCOPTERYGII	0	0	0	0	0	0	1	1	1	3	0	0	5	0	8
35	SOLENOGASTRES	0	0	0	0	0	0	0	0	1	1	0	2	1	0	4
36	TURBELLARIA	1	0	1	0	0	1	0	0	0	0	0	0	0	1	2
37	UDEONYCHOPHORA	0	0	0	0	0	0	3	2	4	9	0	1	0	1	11
38	Total	780	39	819	726	11	1,556	4,008	6,336	7,072	17,416	18	5,071	50,785	15,747	89,856

(2a) What is the data structure (data type) you have used to store the data (e.g., DataFrame, Series, Array, List)?

The data structure is DataFrame.

```
# (2a) What is the data structure (data type) you have used to store the data (e.g., DataFrame, Series, Array, List)?
### Your code here ###
#DataFrame
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

(2b) How many rows and columns are there in the data?

There are 39 rows and 16 columns in the data.

```
# (2b) How many rows and columns are there in the data?
### Your code here ###
#39 rows and 16 columns are there in the data
print(df.shape[0])
print(df.shape[1])
```

```
39
16
```

(2c) What is the dimension of the data?

The dimension of the data is 2.

```
# (2c) What is the dimension of the data?
### Your code here ###
print(df.ndim)
```

```
2
```

(2d) What are the data types (e.g., integer, float) of the data?

'Name', 'Subtotal (EX+EW+ CR(PE)+CR(PEW))', 'CR', 'EN', 'VU', 'Subtotal (threatened spp.)', 'NT or LR/nt', 'LC or LR/lc', 'DD', and 'Total' is object.

'EX', 'EW', 'Subtotal (EX+EW)', 'CR(PE)', 'CR(PEW)', and 'LR/cd' are integer.

```
# (2d) What are the data types (e.g., integer, float) of the data?
### Your code here ###
print(df.dtypes)
```

```
Name                                  object
EX                                     int64
EW                                     int64
Subtotal (EX+EW)                       int64
CR(PE)                                 int64
CR(PEW)                                int64
Subtotal (EX+EW+ CR(PE)+CR(PEW))      object
CR                                    object
EN                                    object
VU                                    object
Subtotal (threatened spp.)            object
LR/cd                                  int64
NT or LR/nt                           object
LC or LR/lc                           object
DD                                    object
Total                                 object
dtype: object
```

(2e) Are there any null values or non-numeric data in your dataset?

There is non-numeric data in the dataset, because some columns (E.g. CR) contain commas as thousands separators, which makes them non-numeric.

is_null = df.isnull().values.any(): This line checks if there are any null values in the dataset.

is_non_numeric = not np.issubdtype(df.values.dtype, np.number): This line checks if the dataset contains non-numeric data.

The if statement checks the values of is_null and is_non_numeric and prints the appropriate message based on the results.

The result shows that the dataset does not contain null values and contains non-numeric data.

```
# (2e) Are there any null values or non-numeric data in your dataset?
### Your code here ###
#There is non-numeric data in the dataset, because some columns (E.g. CR) contain commas as thousands separators,
#which makes them non-numeric.
is_null = df.isnull().values.any()
is_non_numeric = not np.issubdtype(df.values.dtype, np.number)
if is_null:
    print("The dataset contains null values.")
else:
    print("The dataset does not contain null values.")

if is_non_numeric:
    print("The dataset contains non-numeric data.")
else:
    print("The dataset does not contain non-numeric data.")
```

```
The dataset does not contain null values.
The dataset contains non-numeric data.
```

(3) Perform data cleaning. You are suggested to write functions for the data cleaning process.

Since I found ‘Subtotal (EX+EW+ CR(PE)+CR(PEW))’, ‘CR’, ‘EN’, ‘VU’, ‘Subtotal (threatened spp.)’, ‘NT or LR/nt’, ‘LC or LR/lc’, ‘DD’, and ‘Total’ is object, I need to convert the data type from object to int64.

Therefore, I wrote a function called ‘clean_values’ to do data cleaning.

The for loop iterates over each column name in the columns_to_clean list.

df[col] = df[col].str.replace(',' , '') removes commas from the values in the current column (col), which uses the str.replace() method to replace the commas (',') with an empty string ('').

df[col] = pd.to_numeric(df[col]) converts the values in the current column (col) to numeric type, which uses the pd.to_numeric() function from the pandas library to convert the column to a numeric data type.

return df means returns the updated DataFrame.

columns_to_clean = ['Total', 'Subtotal (EX+EW+ CR(PE)+CR(PEW))', 'CR', 'EN', 'VU', 'Subtotal (threatened spp.)', 'NT or LR/nt', 'LC or LR/lc', 'DD'] defines a list of column names (columns_to_clean) that need to be cleaned.

```
# 3. Perform data cleaning. You are suggested to write functions for the data cleaning process.
# You can get additional cells in this Jupyter Notebook, if necessary

def clean_values(df, columns_to_clean):
    for col in columns_to_clean:
        df[col] = df[col].str.replace(',', '')  # Step 1: Remove commas
        df[col] = pd.to_numeric(df[col])  # Step 2: Convert to numeric type
    return df

#Select the range we need to clean
columns_to_clean = ['Total', 'Subtotal (EX+EW+ CR(PE)+CR(PEW))', 'CR', 'EN', 'VU', 'Subtotal (threatened spp.)', 'NT or LR/nt',
                    'LC or LR/lc', 'DD']
df = clean_values(df, columns_to_clean)
# Verify the updated data types
print(df.dtypes)
```

```
Name                                object
EX                                   int64
EW                                   int64
Subtotal (EX+EW)                     int64
CR(PE)                               int64
CR(PEW)                              int64
Subtotal (EX+EW+ CR(PE)+CR(PEW))     int64
CR                                   int64
EN                                   int64
VU                                   int64
Subtotal (threatened spp.)           int64
LR/cd                                int64
NT or LR/nt                          int64
LC or LR/lc                          int64
DD                                   int64
Total                                int64
dtype: object
```

As a result, the data type of 'Total', 'Subtotal (EX+EW+ CR(PE)+CR(PEW))', 'CR', 'EN', 'VU', 'Subtotal (threatened spp.)', 'NT or LR/nt', 'LC or LR/lc', 'DD' have successfully been converted into int64.

Task 2: Data Processing and Visualization

1. Compare the number of species that are in the nine IUCN Red List Categories for the class “MAMMALIA” ;

Firstly, the values of ‘CR(PE)’, ‘CR(PEW)’, and ‘CR’ was added by df['CR(PE)'] + df['CR'] + df['CR(PEW)'].

Secondly, the unnecessary columns were dropped, including 'Subtotal (EX+EW)', 'Subtotal (EX+EW+ CR(PE)+CR(PEW))', 'Subtotal (threatened spp.)', 'Total', 'CR(PE)', and 'CR(PEW)'.

Thirdly, we plot the bar chart in a bid to compare the number of species.

bars = plt.bar(df.columns[1:], df.iloc[23, 1:]) creates a bar chart by calling the plt.bar() function. The first argument df.columns[1:] specifies the x-axis values. The second argument df.iloc[23, 1:] specifies the y-axis values.

plt.xlabel('Red List Category') sets the label for the x-axis as 'Red List Category'.

plt.ylabel('Count') sets the label for the y-axis as 'Count'.

plt.title('Number of Species in IUCN Red List Categories for MAMMALIA') sets the title of the chart as 'Number of Species in IUCN Red List Categories for MAMMALIA'.

xlabel = ['EX', 'EW', 'CR', 'EN', 'VU', 'LR', 'NT', 'LC', 'DD'] defines a list called xlabel that contains the desired x-axis tick labels.

plt.xticks(range(len(xlabel)), xlabel) sets the x-axis tick labels using the plt.xticks() function. The first argument (range(len(xlabel))) specifies the positions of the tick labels on the x-axis, and the second argument (xlabel) provides the actual tick labels.

for bar in bars:

    height = bar.get_height()

    plt.text(bar.get_x() + bar.get_width() / 2, height, height, ha='center', va='bottom').

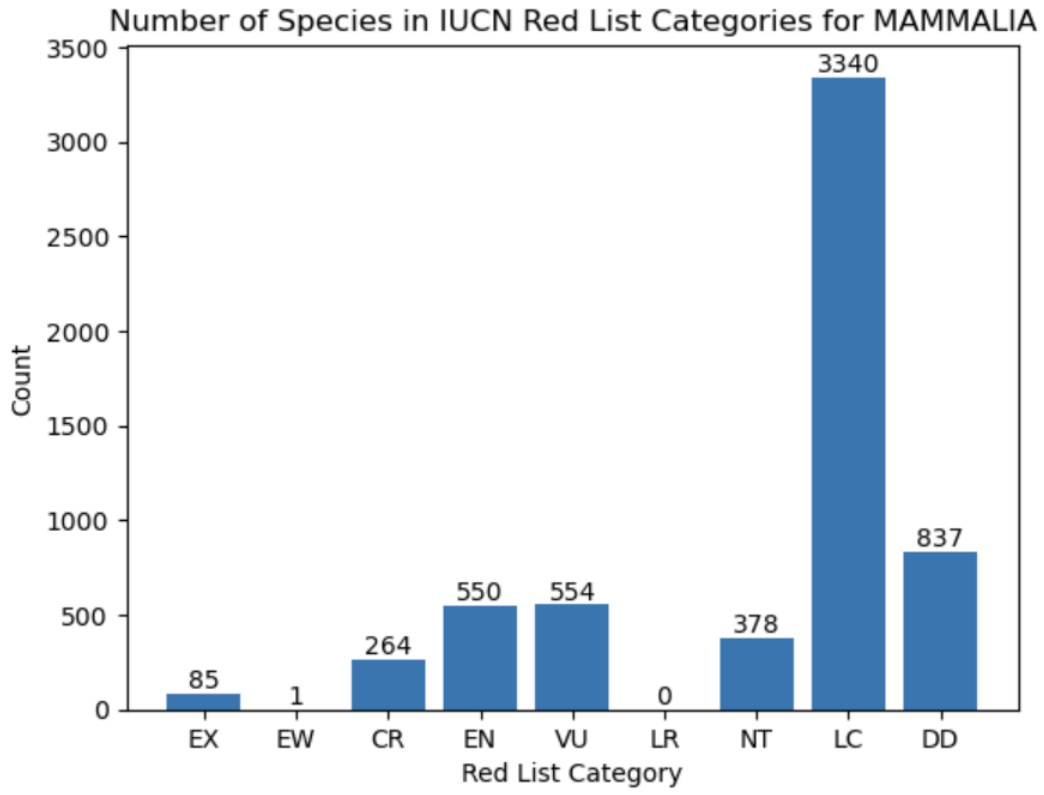
This block of code adds value labels on top of each bar. It iterates over each bar

in the bars variable, retrieves the height of the bar using bar.get_height(), and uses plt.text() to add the height as a label on top of the bar.

```
import matplotlib.pyplot as plt

df['CR'] = df['CR(PE)'] + df['CR'] + df['CR(PEW)']
df = df.drop('Subtotal (EX+EW)', axis=1)
df = df.drop('Subtotal (EX+EW+ CR(PE)+CR(PEW))', axis=1)
df = df.drop('Subtotal (threatened spp.)', axis=1)
df = df.drop('Total', axis=1)
df = df.drop('CR(PE)', axis=1)
df = df.drop('CR(PEW)', axis=1)
```

```
# Plotting the bar chart
bars = plt.bar(df.columns[1:], df.iloc[23, 1:])
plt.xlabel('Red List Category')
plt.ylabel('Count')
plt.title('Number of Species in IUCN Red List Categories for MAMMALIA')

# Set the x-axis tick labels
xlabel = ['EX', 'EW', 'CR', 'EN', 'VU', 'LR', 'NT', 'LC', 'DD']
plt.xticks(range(len(xlabel)), xlabel)

# Add value labels on top of each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, height, ha='center', va='bottom')

# Display the chart
plt.show()
```

Output:



According to the bar chart, LC (Least Concern) has the highest number of species(3340 species). LC is much higher than VU(Vulnerable), which only contains 554 species. It means most species are not threatened under the risk of extinct.

The consideration of Gestalt principles of perception:

The Gestalt principles of Proximity and Similarity influence how we group and categorize information in a bar chart. Placing bars close together or using similar visual attributes (like color) helps viewers perceive them as belonging to the same group or category. By leveraging these principles, we can improve the chart's clarity and facilitate quick identification and differentiation of data subsets.

The principle of Continuity affects how we perceive the overall pattern and structure of a bar chart. By aligning and arranging bars along the x-axis in a continuous manner, we create a visual pattern that connects each bar to its corresponding category. This continuity enables viewers to perceive trends, comparisons, and patterns more easily. Ensuring a smooth and uninterrupted flow of bars along the x-axis enhances the viewer's understanding of category relationships.

The principle of Closure influences how we perceive individual bars as complete objects, even without physical boundaries. When plotting a bar chart, viewers mentally close the top of each bar, perceiving them as solid shapes. This principle assists in understanding the magnitude or value represented by each bar, facilitating comparison and interpretation of the data. Ensuring that bars extend to the baseline or axis line without interruptions leverages the principle of Closure for more accurate data comprehension.

2. Compare the number of species that are classified as Extinct / Extinct in Wild, including those are “Possibly” Extinct for the classes “AVES”, “ MAMMALIA” and “INSECTA”.

We still compare the number of species by using bar chart.

extinct_values = df.loc[[5, 23, 21], ['EX', 'EW']] selects specific rows and columns from the DataFrame df, which uses the loc accessor to select rows with indices 5, 23, and 21, and columns with labels 'EX' and 'EW'. The resulting subset of data is assigned to the extinct_values variable.

red_list_categories = ['AVES', 'MAMMALIA', 'INSECTA'] defines a list called red_list_categories that contains the desired x-axis labels for the bar chart.

x = list(range(len(red_list_categories))) creates a list of x-values for the bars, which generates a sequence of numbers from 0 to the length of red_list_categories and converts it to a list.

bars1 = plt.bar([val - width/2 for val in x], extinct_values['EX'], width=width, label='EX') creates the first group of bars, which uses the plt.bar() function to create bars with x-values shifted to the left by width/2. The heights of the bars are taken from the 'EX' column of the extinct_values DataFrame. The width parameter specifies the width of the bars, and the label parameter assigns a label to this group of bars. The resulting bars are assigned to the bars1 variable.

bars2 = plt.bar([val + width/2 for val in x], extinct_values['EW'], width=width, label='EW') creates the second group of bars, which almost as the same as bars1 but bars2 create bars with x-values shifted to the right by width/2 and labelled by 'EW'.

plt.xlabel(' The name of species') sets the label for the x-axis as ' The name of species'.

plt.ylabel('Count') sets the label for the y-axis as 'Count'.

plt.title('Number of Extinct / Extinct in Wild Species') sets the title of the chart as 'Number of Extinct / Extinct in Wild Species'.

plt.xticks(x, red_list_categories) sets the x-axis tick labels using the plt.xticks()

function. The first argument (x) specifies the positions of the tick labels on the x-axis, and the second argument (red_list_categories) provides the actual tick labels.

plt.legend()adds a legend to the chart, which displays the labels for the different groups of bars.

The following block of code adds value labels on top of each bar.

for bar1, bar2 in zip(bars1, bars2):

    height1 = bar1.get_height()

    height2 = bar2.get_height()

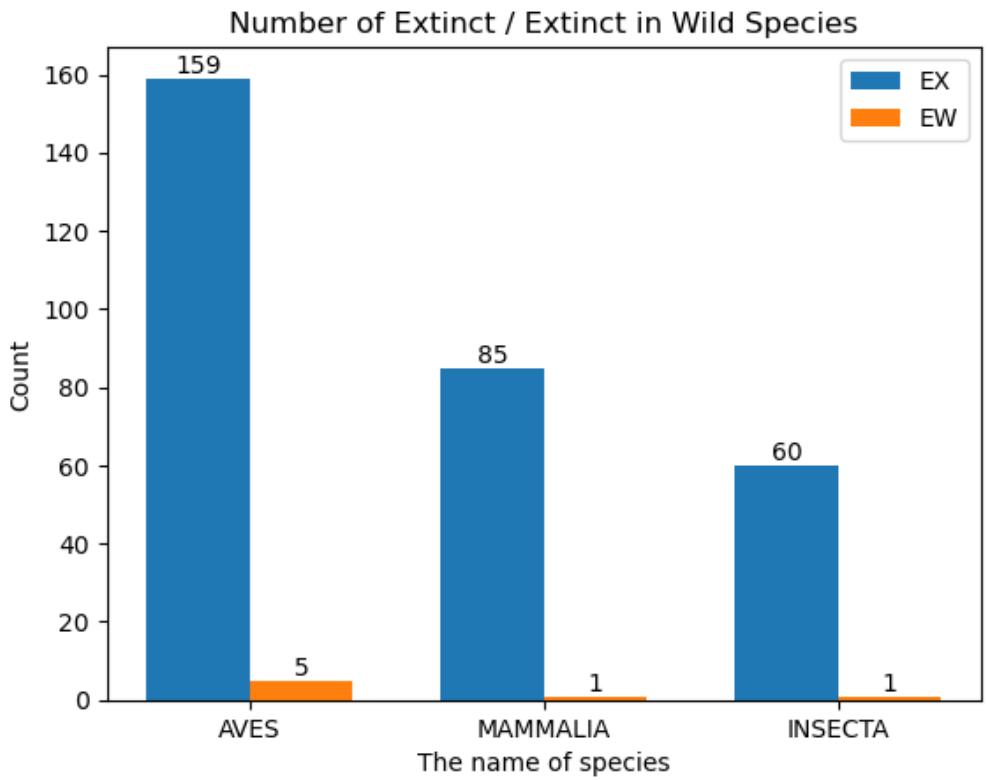
    plt.text(bar1.get_x() + bar1.get_width() / 2, height1, height1, ha='center', va='bottom')

    plt.text(bar2.get_x() + bar2.get_width() / 2, height2, height2, ha='center', va='bottom')

It iterates over each pair of bars in bars1 and bars2, retrieves the heights of the bars using bar1.get_height() and bar2.get_height(), and uses plt.text() to add the heights as labels on top of the bars.

```
# Selecting the desired columns from the DataFrame
extinct_values = df.loc[[5, 23, 21], ['EX', 'EW']]

# Define the Red List categories as x-axis labels
red_list_categories = ['AVES', 'MAMMALIA', 'INSECTA']

# Plotting the bar plot
x = list(range(len(red_list_categories)))
width = 0.35

bars1 = plt.bar([val - width/2 for val in x], extinct_values['EX'], width=width, label='EX')
bars2 = plt.bar([val + width/2 for val in x], extinct_values['EW'], width=width, label='EW')

plt.xlabel('The name of species')
plt.ylabel('Count')
plt.title('Number of Extinct / Extinct in Wild Species')
plt.xticks(x, red_list_categories)
plt.legend()

# Add value labels on top of each bar
for bar1, bar2 in zip(bars1, bars2):
    height1 = bar1.get_height()
    height2 = bar2.get_height()
    plt.text(bar1.get_x() + bar1.get_width() / 2, height1, height1, ha='center', va='bottom')
    plt.text(bar2.get_x() + bar2.get_width() / 2, height2, height2, ha='center', va='bottom')

plt.show()
```

Output:



According to the bar chart, AVES has the highest number of Extinct (159 species) and Extinct in Wild Species (5 species), which obviously shows that AVES are threatened by the risk of extinct. For EX, AVES has 74 extinct species higher than MAMMALIA, which only contains 85 extinct species.

The consideration of Gestalt principles of perception:

Pattern and Continuity: The principle of Continuity plays a role in how we perceive the overall pattern and structure of the bar chart. The alignment and arrangement of bars along the x-axis create a continuous visual pattern that helps viewers associate bars with their respective categories. This principle helps in perceiving trends, comparisons, or patterns in the data.

Completeness and Closure: The principle of Closure affects how we perceive individual bars as complete objects, despite the absence of any physical boundaries. Viewers tend to mentally close the top of each bar, perceiving them as solid shapes. This principle helps in perceiving the magnitude or value represented by each bar, making it easier to compare and interpret the data.

Grouping and Categorization: The principles of Proximity and Similarity influence how we perceive groups and categories within a bar chart. Bars that are close together or share similar visual attributes are perceived as belonging to the same group or category, such as the color of blue and orange. This can help viewers quickly identify and differentiate different groups or subsets of data in the chart.