Biodiversity in National Parks

Table of Contents

1: Introduction

The "**biodiversity.ipynb**" contains all the information and code for the project.
The "**biodiversity.csv**" contains the cleaned and merged data from "**observations.csv**" and "**species_info.csv**", which was curated by **National Parks Service** (https://www.nps.gov/index.htm).

The **objective** of this project was to **analyze the biodiversity data** in order to **better understand the level of endangerment of certain species in various parks** along with the **frequency of their sightings**.

Endangered species are living organisms, both plants and animals, that face a high risk of extinction.
These species encounter significant threats caused by diverse factors such as habitat loss, climate change, pollution, poaching, and the introduction of invasive species.
The declining population numbers of these species raise serious concerns about the delicate equilibrium of **biodiversity** on our planet.

Biodiversity encompasses the remarkable array of life forms, ranging from microorganisms to plants, animals, and entire ecosystems.
It encompasses the intricate network of interactions between different species and their surrounding environment. Therefore, it plays a critical role in sustaining the health and functionality of ecosystems, which, in turn, supports life on Earth.

The conservation of biodiversity is crucial for ensuring the continued well-being of our planet and the future of all living beings.

2: Modules

In [1]:
```python
import re
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import plotly.express as px
import plotly.graph_objects as go
from scipy.stats import entropy
from scipy.stats import fisher_exact
from scipy.stats import barnard_exact
from scipy.stats import boschloo_exact
from scipy.stats import chi2_contingency
```

3: Functions

In [2]:
```python
# function to get dataframe name; Useful only for missing_data_check function
def get_df_name(df):
    name =[x for x in globals() if globals()[x] is df][0]
    return name

# function that checks if any data is missing, identifies the data type, and displays the first 10 row indices where missing data is found
def missing_data_check(data):
    print(f'Missing data check for "{get_df_name(data)}" dataframe:', "\n")
    nan_values = data.isna().any().any()
    missing_values = data.empty
    empty_string_values = (data == "").any().any()
    print(f"NaN values exist: {nan_values}")
    print(f"Missing values exist: {missing_values}")
    print(f"Empty string values exist: {empty_string_values}")

    if nan_values or missing_values or empty_string_values:
        print("\n" + "Rows with missing or empty values:")

        if nan_values:
            nan_rows = data[data.isna().any(axis=1)]
            print("NaN rows:")
            print(nan_rows.index.tolist()[:10], "\n")

        if missing_values:
            missing_rows = data[data.empty]
            print("Missing rows:")
            print(missing_rows.index.tolist()[:10], "\n")

        if empty_string_values:
            empty_string_rows = data[data == ""]
            print("Empty string rows:")
            print(empty_string_rows.index.tolist()[:10], "\n")
    else:
        print("\n" + "No missing or empty values found.", "\n")
```

In [3]:
```python
# function that calculates Chi2, Fisher's, Barnard's and Boschloo's exact tests and stores the data into a "exact_tests_results_df" dataframe
# Boschloo's exact test is not considered in this analysis due to its use for ordinal categories
# example: contingency_table = [[584, 120], [288, 28]]
def calculate_exact_tests(contingency_table):
    chi2_result = chi2_contingency(contingency_table)
    fisher_result = fisher_exact(contingency_table)
    barnard_result = barnard_exact(contingency_table)
    #boschloo_result = boschloo_exact(contingency_table)

    exact_tests_results_df = pd.DataFrame(columns=["Test", "Statistic", "p-value"])
    exact_tests_results_df.loc[0] = ["Chi2", chi2_result.statistic, chi2_result.pvalue]
    exact_tests_results_df.loc[1] = ["Fisher", fisher_result.statistic, fisher_result.pvalue]
    exact_tests_results_df.loc[2] = ["Barnard", barnard_result.statistic, barnard_result.pvalue]
    #exact_tests_results_df.loc[3] = ["Boschloo", boschloo_result.statistic, boschloo_result.pvalue]

    return exact_tests_results_df
```

4: Data Cleaning

The **primary focus was placed on cleaning the data**, with the objective of obtaining a **singular dataset that is entirely pristine and suitable for analysis**.

Initially, a thorough examination and refinement process was conducted on both files individually. Subsequently, these files were merged, and an additional round of filtering was performed, ensuring the creation of a **single, definitive ".csv" file**.

4.0: Personal Preference Options

In [4]:
```
# personal preference settings
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
pd.set_option("display.max_colwidth", None)
pd.set_option("expand_frame_repr", False)

# load the csv files into DataFrames
observations = pd.read_csv("observations.csv")
species = pd.read_csv("species_info.csv")
```

4.1: observations.csv

In [5]:
```
# general dataframe structure
observations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23296 entries, 0 to 23295
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   scientific_name  23296 non-null  object
 1   park_name        23296 non-null  object
 2   observations     23296 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 546.1+ KB
```

In [6]:
```
# a quick glance into the data
observations.head()
```

Out[6]:

	scientific_name	park_name	observations
0	Vicia benghalensis	Great Smoky Mountains National Park	68
1	Neovison vison	Great Smoky Mountains National Park	77
2	Prunus subcordata	Yosemite National Park	138
3	Abutilon theophrasti	Bryce National Park	84
4	Githopsis specularioides	Great Smoky Mountains National Park	85

In [7]:
```
missing_data_check(observations)
```

```
Missing data check for "observations" dataframe:

NaN values exist: False
Missing values exist: False
Empty string values exist: False

No missing or empty values found.
```

In [8]:
```
# number of unique park places
observations.park_name.nunique()
```

Out[8]: 4

In [9]:
```
# unique park places names
observations.park_name.unique()
```

Out[9]:
```
array(['Great Smoky Mountains National Park', 'Yosemite National Park',
       'Bryce National Park', 'Yellowstone National Park'], dtype=object)
```

In [10]:
```
# number of distinct species
observations.scientific_name.nunique()
```

Out[10]: 5541

The "**observations.csv**" file contains three columns:

- **Scientific Name**

  > Latin name of species. There are 5541 distinct species in this data.

- **Park Name**

  > There are four different parks in the observations file:
  >
  > 1. Great Smoky Mountains National Park
  > 2. Yosemite National Park
  > 3. Bryce National Park
  > 4. Yellowstone National Park

- **Observations**

> An integer type data representing sightings of species in a park.

There are no null/NaN/missing values.

4.2: species.csv

In [11]:
```
# general file structure
species.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5824 entries, 0 to 5823
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   category             5824 non-null   object
 1   scientific_name      5824 non-null   object
 2   common_names         5824 non-null   object
 3   conservation_status  191 non-null    object
dtypes: object(4)
memory usage: 182.1+ KB
```

In [12]:
```
# a quick glance into the data
species.head()
```

Out[12]:

	category	scientific_name	common_names	conservation_status
0	Mammal	Clethrionomys gapperi gapperi	Gapper's Red-Backed Vole	NaN
1	Mammal	Bos bison	American Bison, Bison	NaN
2	Mammal	Bos taurus	Aurochs, Aurochs, Domestic Cattle (Feral), Domesticated Cattle	NaN
3	Mammal	Ovis aries	Domestic Sheep, Mouflon, Red Sheep, Sheep (Feral)	NaN
4	Mammal	Cervus elaphus	Wapiti Or Elk	NaN

In [13]:
```
missing_data_check(species)
```

```
Missing data check for "species" dataframe:

NaN values exist: True
Missing values exist: False
Empty string values exist: False

Rows with missing or empty values:
NaN rows:
[0, 1, 2, 3, 4, 5, 6, 10, 11, 12]
```

In [14]:
```
# where are NaN values? conservation_status!
species.isna().sum()
```

Out[14]:
```
category                 0
scientific_name          0
common_names             0
conservation_status    5633
dtype: int64
```

In [15]:
```
# number of unique conservation status groups
species.conservation_status.nunique()
```

Out[15]: 4

In [16]:
```
# unique conservations status group names
species.conservation_status.unique()
```

Out[16]:
```
array([nan, 'Species of Concern', 'Endangered', 'Threatened',
       'In Recovery'], dtype=object)
```

In [17]:
```
# number of unique species in the data
# !WRONG! Although there is a correct number of distinct species in the "species.csv" file,
# it should be noted that there is a discrepancy based on the output of "species.info()"!
species.scientific_name.nunique()
```

Out[17]: 5541

In [18]:
```
# there is a discrepancy in the number od species in both datasets (5541 in "observations.csv" / 5824 in "species.csv"),
# this can be observed with the "species.info()" function as well!
len(species.scientific_name)
```

Out[18]: 5824

In [19]:
```
# more depth for the "NaN" values
display(species.conservation_status.value_counts())
```

```
conservation_status
Species of Concern    161
Endangered             16
Threatened             10
In Recovery             4
Name: count, dtype: int64
```

In [20]:
```
# percentage of "nan" values for conservation status
print("\n" + f'"NaN" values in "conservation_status" constitute ' + str(round(5633 / 5824 * 100, 2)) + "% of the whole dataset!")
```

```
"NaN" values in "conservation_status" constitute 96.72% of the whole dataset!
```

In [21]:
```
# Discarding over 95 % of the data isn't favorable.
# the "NaN" conservation status has been replaced with "Least concern," enabling its inclusion in the analysis.
species["conservation_status"].fillna("Least concern", inplace=True)
```

In [22]:
```
# number of unique categories
species.category.nunique()
```

Out[22]: 7

In [23]:
```
# unique categories
species.category.unique()
```

Out[23]:
```
array(['Mammal', 'Bird', 'Reptile', 'Amphibian', 'Fish', 'Vascular Plant',
       'Nonvascular Plant'], dtype=object)
```

In [24]:
```
# a quick glance at the common names
species["common_names"].head()
```

Out[24]:
```
0                                     Gapper's Red-Backed Vole
1                                        American Bison, Bison
2    Aurochs, Aurochs, Domestic Cattle (Feral), Domesticated Cattle
3                Domestic Sheep, Mouflon, Red Sheep, Sheep (Feral)
4                                                  Wapiti Or Elk
Name: common_names, dtype: object
```

In [25]:
```
# by observing the common names from the "species.csv", we can see the usual delimiters in these examples
species["common_names"].iloc[[2, 4, 16, 232, 233, 750, 997, 2295, 2554]]
```

Out[25]:
```
2       Aurochs, Aurochs, Domestic Cattle (Feral), Domesticated Cattle
4                                                  Wapiti Or Elk
16                                        Panther (Mountain Lion)
232                            Baltimore Oriole, Northern Oriole
233                                             Orchard Oriole
750                                              Curtis? Aster
997                                       Venus? Looking-Glass
2295                                     A Bluegrass, Bluegrass
2554                               A Bramble, Truculent Blackberry
Name: common_names, dtype: object
```

In [26]:
```
# regular expression that matches ",", letters "Or" together and not in a word (so not 'abort' and such) and "("
regex = r",|\bOr\b|\("

# clean the common names column and keep only the first common name
species["common_names"] = species["common_names"].apply(lambda x: re.split(regex, x)[0].strip())

# remove the capital letter "A" with a whitespace after it
species["common_names"] = species["common_names"].apply(lambda x: re.sub(r"A\s", "", x))

# substitute "?" with whitespace
species["common_names"] = species["common_names"].apply(lambda x: re.sub(r"\?", "", x))
```

In [27]:
```
# check if the correction works
species["common_names"].iloc[[2, 4, 16, 232, 233, 750, 997, 2295, 2554]]
```

Out[27]:
```
2                 Aurochs
4                  Wapiti
16                Panther
232       Baltimore Oriole
233         Orchard Oriole
750           Curtis Aster
997     Venus Looking-Glass
2295              Bluegrass
2554                Bramble
Name: common_names, dtype: object
```

In [28]:
```
# initially, the regex method removed entire common names for some species, which is the reason for conducting this check and for making the function in the first place.
# Empty string (" ") isn't the same as NaN, so ".isna()" isn't viable here...
missing_data_check(species)
```

```
Missing data check for "species" dataframe:

NaN values exist: False
Missing values exist: False
Empty string values exist: False

No missing or empty values found.
```

In [29]:
```
# number of duplicate scientific names in the data
duplicated_scientific_names = species["scientific_name"].duplicated()
print(duplicated_scientific_names.value_counts(), "\n")

# number of duplicate common names in the data.
# Not relevant since I'm planning to keep only the first common name to make the dataset cleaner!
duplicated_common_names = species["common_names"].duplicated()
print(duplicated_common_names.value_counts())
```

```
scientific_name
False    5541
True      283
Name: count, dtype: int64

common_names
False    5291
True      533
Name: count, dtype: int64
```

In [30]:
```
# store the duplicates to better understand how to clean them
duplicates = species[species["scientific_name"].duplicated()]

# a quick glance into the duplicates
duplicates.head()
```

Out[30]:

	category	scientific_name	common_names	conservation_status
3017	Mammal	Cervus elaphus	Rocky Mountain Elk	Least concern
3019	Mammal	Odocoileus virginianus	White-Tailed Deer	Least concern
3020	Mammal	Canis lupus	Gray Wolf	In Recovery
3022	Mammal	Puma concolor	Cougar	Least concern
3025	Mammal	Lutra canadensis	River Otter	Least concern

In [31]:
```
# seeing if all duplicates are the same, chosen at random from the .head() display
display(species[species["scientific_name"] == "Cervus elaphus"])
display(species[species["scientific_name"] == "Lutra canadensis"])

# there is a discrepancy in the conservation status of some (how many?) duplicates
display(species[species["scientific_name"] == "Canis lupus"])
```

	category	scientific_name	common_names	conservation_status
4	Mammal	Cervus elaphus	Wapiti	Least concern
3017	Mammal	Cervus elaphus	Rocky Mountain Elk	Least concern

	category	scientific_name	common_names	conservation_status
20	Mammal	Lutra canadensis	Northern River Otter	Least concern
3025	Mammal	Lutra canadensis	River Otter	Least concern

	category	scientific_name	common_names	conservation_status
8	Mammal	Canis lupus	Gray Wolf	Endangered
3020	Mammal	Canis lupus	Gray Wolf	In Recovery
4448	Mammal	Canis lupus	Gray Wolf	Endangered

In [32]:
```
# find duplicates based on scientific name
dups = species[species.duplicated(subset="scientific_name", keep=False)]

# find duplicates with different conservation status.
# "~" is a negation operator which ensures that only duplicates with different conservation statuses are stored
dups_with_diff_conservation_status = dups[dups.duplicated(subset="scientific_name", keep=False) & ~dups.duplicated(subset="conservation_status", keep=False)]

display(dups_with_diff_conservation_status)
```

	category	scientific_name	common_names	conservation_status
3020	Mammal	Canis lupus	Gray Wolf	In Recovery
3283	Fish	Oncorhynchus mykiss	Rainbow Trout	Threatened

In [33]:
```
species[species["scientific_name"] == "Oncorhynchus mykiss"]
```

Out[33]:

	category	scientific_name	common_names	conservation_status
560	Fish	Oncorhynchus mykiss	Rainbow Trout	Least concern
3283	Fish	Oncorhynchus mykiss	Rainbow Trout	Threatened

In [34]:
```
# since each of the first duplicate species has the correct conservation status, this should yield a clean dataset.
species.drop_duplicates(subset=["scientific_name"], keep="first", inplace=True)
```

In [35]:
```
# double-check
display(species[species["scientific_name"] == "Canis lupus"])
display(species[species["scientific_name"] == "Oncorhynchus mykiss"])
```

	category	scientific_name	common_names	conservation_status
8	Mammal	Canis lupus	Gray Wolf	Endangered

	category	scientific_name	common_names	conservation_status
560	Fish	Oncorhynchus mykiss	Rainbow Trout	Least concern

In [36]:
```
# verify the number of distinct species in each dataset; Checks out!
print(observations.scientific_name.nunique())
print(species.scientific_name.nunique())
print(len(species.scientific_name))
```

```
5541
5541
5541
```

The "**species.csv**" file contains four columns:

- **Category**

  > There are seven distinct categories in the data:
  >
  > 1. Mammal
  > 2. Bird
  > 3. Reptile
  > 4. Amphibian
  > 5. Fish
  > 6. Vascular Plant
  > 7. Nonvascular Plant

- **Scientific Name**

  > Apperas to be the same naming as in "observations.csv", although with a different number of distinct species (before cleaning).

- **Common Names**

  > One or more common names for different species. Only the first common name is preserved after cleaning.

- **Conservation Status**

  > **Refers to the assessment of the risk level faced by a species or ecosystem, indicating the extent to which it is threatened or protected based on scientific evaluation and criteria**.
  > The dataset includes **four distinct conservation status groups**, along with the majority classified as "NaN" (before cleaning):
  >
  > 1. **Species of Concern**
  > 2. **Endangered**
  > 3. **Threatened**
  > 4. **In Recovery**
  > 5. "NaN"

According to Wikipedia (https://en.wikipedia.org/wiki/Conservation_status), species are classified by the **IUCN Red List** into **nine groups**:

1. **Extinct (EX)** – No known living individuals.
2. **Extinct in the wild (EW)** – Known only to survive in captivity, or as a naturalized population outside its historic range.
3. **Critically Endangered (CR)** – Highest risk of extinction in the wild.
4. **Endangered (EN)** – Higher risk of extinction in the wild.
5. **Vulnerable (VU)** – High risk of extinction in the wild.
6. **Near Threaened (NT)** – Likely to become endangered in the near future.
7. **Conservation Dependent (CD)** – Low risk; is conserved to prevent being near threatened, certain events may lead it to being a higher risk level.
8. **Least concern (LC)** – Very Low risk; does not qualify for a higher risk category and not likely to be threatened in the near future. Widespread and abundant taxa are included in this category.

9. **Data deficient (DD)** – Not enough data to make an assessment of its risk of extinction.
10. **Not evaluated (NE)** – Has not yet been evaluated against the criteria.

The "**NaN**" group is replaced with "**Least concern**" in the "**species.csv**".

There is a **discrepancy between the number of distinct species and the conservation status of some of them**.

U.S. Fish & Wildlife Service categorizes **Canis lupus** as "Endangered" (https://www.fws.gov/initiative/protecting-wildlife/gray-wolf-recovery-news-and-updates).

The species **Oncorhynchus mykiss** is not listed (https://ecos.fws.gov/ecp/species/757).
However, in certain states, it is classified as a "Pest Species." Therefore, considering the existing distributions, the "Least Concern" group is appropriate.

Duplicates have been identified and subsequently removed, thereby preserving the accurate conservation status of each species based on its initial occurrence and/or through external verification.

4.3: Merged Data

In [37]:
```
# merge column will be "scientific_name"
display(observations.head())
display(species.head())

# merge the data into a single dataframe
biodiversity_dirty = pd.merge(observations, species, on="scientific_name")
```

	scientific_name	park_name	observations
0	Vicia benghalensis	Great Smoky Mountains National Park	68
1	Neovison vison	Great Smoky Mountains National Park	77
2	Prunus subcordata	Yosemite National Park	138
3	Abutilon theophrasti	Bryce National Park	84
4	Githopsis specularioides	Great Smoky Mountains National Park	85

	category	scientific_name	common_names	conservation_status
0	Mammal	Clethrionomys gapperi gapperi	Gapper's Red-Backed Vole	Least concern
1	Mammal	Bos bison	American Bison	Least concern
2	Mammal	Bos taurus	Aurochs	Least concern
3	Mammal	Ovis aries	Domestic Sheep	Least concern
4	Mammal	Cervus elaphus	Wapiti	Least concern

In [38]:
```
# check if the numbers match; They do!
print(len(observations))
print(len(biodiversity_dirty))

print(observations.scientific_name.nunique())
print(species.scientific_name.nunique())
print(biodiversity_dirty.scientific_name.nunique(), "\n")

# the expected number of rows could be four times the number of distinct species in the data if each species is observed in each park!
print(f"Possible expected number of rows: {biodiversity_dirty.scientific_name.nunique() * 4}")
print(f"Difference: " + str(23296 - 22164))
```

```
23296
23296
5541
5541
5541

Possible expected number of rows: 22164
Difference: 1132
```

In [39]:
```
# general dataframe structure
biodiversity_dirty.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23296 entries, 0 to 23295
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   scientific_name      23296 non-null  object
 1   park_name            23296 non-null  object
 2   observations         23296 non-null  int64 
 3   category             23296 non-null  object
 4   common_names         23296 non-null  object
 5   conservation_status  23296 non-null  object
dtypes: int64(1), object(5)
memory usage: 1.1+ MB
```

In [40]:
```
missing_data_check(biodiversity_dirty)
```

```
Missing data check for "biodiversity_dirty" dataframe:

NaN values exist: False
Missing values exist: False
Empty string values exist: False

No missing or empty values found.
```

In [41]:
```
# a quick glance into the data
biodiversity_dirty.head()
```

Out[41]:

	scientific_name	park_name	observations	category	common_names	conservation_status
0	Vicia benghalensis	Great Smoky Mountains National Park	68	Vascular Plant	Purple Vetch	Least concern
1	Vicia benghalensis	Yosemite National Park	148	Vascular Plant	Purple Vetch	Least concern
2	Vicia benghalensis	Yellowstone National Park	247	Vascular Plant	Purple Vetch	Least concern
3	Vicia benghalensis	Bryce National Park	104	Vascular Plant	Purple Vetch	Least concern
4	Neovison vison	Great Smoky Mountains National Park	77	Mammal	American Mink	Least concern

In [42]:
```
# strip the "National Park" from park names
biodiversity_dirty["park_name"] = biodiversity_dirty["park_name"].map(lambda x: x.rstrip("National Park"))

biodiversity_dirty.head()
```

Out[42]:

	scientific_name	park_name	observations	category	common_names	conservation_status
0	Vicia benghalensis	Great Smoky Mountains	68	Vascular Plant	Purple Vetch	Least concern
1	Vicia benghalensis	Yosemite	148	Vascular Plant	Purple Vetch	Least concern
2	Vicia benghalensis	Yellowstone	247	Vascular Plant	Purple Vetch	Least concern
3	Vicia benghalensis	Bryce	104	Vascular Plant	Purple Vetch	Least concern
4	Neovison vison	Great Smoky Mountains	77	Mammal	American Mink	Least concern

In [43]:
```
# check the dataframe for random species; Notice that observations can be summed up to reduce the number of rows in the dataset!
display(biodiversity_dirty[biodiversity_dirty["scientific_name"] == "Helianthus annuus"])
display(biodiversity_dirty[biodiversity_dirty["scientific_name"] == "Plantago lanceolata"])
```

	scientific_name	park_name	observations	category	common_names	conservation_status
268	Helianthus annuus	Yellowstone	265	Vascular Plant	Common Sunflower	Least concern
269	Helianthus annuus	Bryce	140	Vascular Plant	Common Sunflower	Least concern
270	Helianthus annuus	Yellowstone	235	Vascular Plant	Common Sunflower	Least concern
271	Helianthus annuus	Bryce	118	Vascular Plant	Common Sunflower	Least concern
272	Helianthus annuus	Yosemite	123	Vascular Plant	Common Sunflower	Least concern
273	Helianthus annuus	Yosemite	169	Vascular Plant	Common Sunflower	Least concern
274	Helianthus annuus	Great Smoky Mountains	44	Vascular Plant	Common Sunflower	Least concern
275	Helianthus annuus	Great Smoky Mountains	54	Vascular Plant	Common Sunflower	Least concern

	scientific_name	park_name	observations	category	common_names	conservation_status
20900	Plantago lanceolata	Great Smoky Mountains	51	Vascular Plant	English Plantain	Least concern
20901	Plantago lanceolata	Yosemite	119	Vascular Plant	English Plantain	Least concern
20902	Plantago lanceolata	Bryce	119	Vascular Plant	English Plantain	Least concern
20903	Plantago lanceolata	Yosemite	136	Vascular Plant	English Plantain	Least concern
20904	Plantago lanceolata	Great Smoky Mountains	79	Vascular Plant	English Plantain	Least concern
20905	Plantago lanceolata	Bryce	137	Vascular Plant	English Plantain	Least concern
20906	Plantago lanceolata	Yellowstone	260	Vascular Plant	English Plantain	Least concern
20907	Plantago lanceolata	Yellowstone	264	Vascular Plant	English Plantain	Least concern

In [44]:
```
# count the number of times a species occurs in the data
scientific_name_counts = biodiversity_dirty["scientific_name"].value_counts()

# sort the counts from smallest to largest
sorted_counts = scientific_name_counts.sort_values()

# display least and most occuring species
print(sorted_counts[1:6], "\n")
print(sorted_counts[-6:-1])
```

```
scientific_name
Limosa fedoa              4
Juncus uncialis           4
Thaspium barbinode        4
Polystichum scopulinum    4
Comandra umbellata        4
Name: count, dtype: int64

scientific_name
Castor canadensis        12
Streptopelia decaocto    12
Holcus lanatus           12
Hypochaeris radicata     12
Puma concolor            12
Name: count, dtype: int64
```

In [45]:
```
# check the observations for least and most occurrences
display(biodiversity_dirty[biodiversity_dirty["scientific_name"] == "Limosa fedoa"])
display(biodiversity_dirty[biodiversity_dirty["scientific_name"] == "Holcus lanatus"])
```

	scientific_name	park_name	observations	category	common_names	conservation_status
1388	Limosa fedoa	Yellowstone	266	Bird	Marbled Godwit	Species of Concern
1389	Limosa fedoa	Yosemite	131	Bird	Marbled Godwit	Species of Concern
1390	Limosa fedoa	Great Smoky Mountains	82	Bird	Marbled Godwit	Species of Concern
1391	Limosa fedoa	Bryce	96	Bird	Marbled Godwit	Species of Concern

	scientific_name	park_name	observations	category	common_names	conservation_status
11632	Holcus lanatus	Yellowstone	262	Vascular Plant	Common Velvet Grass	Least concern
11633	Holcus lanatus	Yosemite	146	Vascular Plant	Common Velvet Grass	Least concern
11634	Holcus lanatus	Bryce	83	Vascular Plant	Common Velvet Grass	Least concern
11635	Holcus lanatus	Yosemite	179	Vascular Plant	Common Velvet Grass	Least concern
11636	Holcus lanatus	Yosemite	138	Vascular Plant	Common Velvet Grass	Least concern
11637	Holcus lanatus	Bryce	117	Vascular Plant	Common Velvet Grass	Least concern
11638	Holcus lanatus	Bryce	96	Vascular Plant	Common Velvet Grass	Least concern
11639	Holcus lanatus	Great Smoky Mountains	65	Vascular Plant	Common Velvet Grass	Least concern
11640	Holcus lanatus	Yellowstone	256	Vascular Plant	Common Velvet Grass	Least concern
11641	Holcus lanatus	Great Smoky Mountains	77	Vascular Plant	Common Velvet Grass	Least concern
11642	Holcus lanatus	Great Smoky Mountains	74	Vascular Plant	Common Velvet Grass	Least concern
11643	Holcus lanatus	Yellowstone	287	Vascular Plant	Common Velvet Grass	Least concern

In [46]:
```
# check if there is any species that isn't observed in every park in the data

# a list of all park names
all_park_names = biodiversity_dirty["park_name"].unique()

# check which species aren't observed in every park
not_in_every_park = []
for species in biodiversity_dirty["scientific_name"].unique():
    species_parks = biodiversity_dirty[biodiversity_dirty["scientific_name"] == species]["park_name"].unique()
    if len(species_parks) != len(all_park_names):
        not_in_every_park.append(species)

# print the list of species not found in every park or a message if all species are observed in every park
if not_in_every_park:
    print("The following species are not observed in every park:")
    for species in not_in_every_park:
        print(species)
else:
    print("All species are observed in every park!")
```

```
All species are observed in every park!
```

In [47]:
```
# group by park name and sum observations for each scientific name
biodiversity = biodiversity_dirty.groupby(["park_name", "scientific_name", "category", "common_names", "conservation_status"])["observations"].sum().reset_index()

# inspect the cleaned DataFrame
display(biodiversity.head())

# double-check the observations for random species
display(biodiversity[biodiversity["scientific_name"] == "Helianthus annuus"])
display(biodiversity[biodiversity["scientific_name"] == "Holcus lanatus"])
```

	park_name	scientific_name	category	common_names	conservation_status	observations
0	Bryce	Abies bifolia	Vascular Plant	Rocky Mountain Alpine Fir	Least concern	109
1	Bryce	Abies concolor	Vascular Plant	Balsam Fir	Least concern	83
2	Bryce	Abies fraseri	Vascular Plant	Fraser Fir	Species of Concern	109
3	Bryce	Abietinella abietina	Nonvascular Plant	Abietinella Moss	Least concern	101
4	Bryce	Abronia ammophila	Vascular Plant	Wyoming Sand Verbena	Species of Concern	92

	park_name	scientific_name	category	common_names	conservation_status	observations
2368	Bryce	Helianthus annuus	Vascular Plant	Common Sunflower	Least concern	258
7909	Great Smoky Mountains	Helianthus annuus	Vascular Plant	Common Sunflower	Least concern	98
13450	Yellowstone	Helianthus annuus	Vascular Plant	Common Sunflower	Least concern	500
18991	Yosemite	Helianthus annuus	Vascular Plant	Common Sunflower	Least concern	292

	park_name	scientific_name	category	common_names	conservation_status	observations
2452	Bryce	Holcus lanatus	Vascular Plant	Common Velvet Grass	Least concern	296
7993	Great Smoky Mountains	Holcus lanatus	Vascular Plant	Common Velvet Grass	Least concern	216
13534	Yellowstone	Holcus lanatus	Vascular Plant	Common Velvet Grass	Least concern	805
19075	Yosemite	Holcus lanatus	Vascular Plant	Common Velvet Grass	Least concern	463

In [48]:
```
# general dataframe structure
biodiversity.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22164 entries, 0 to 22163
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   park_name            22164 non-null  object
 1   scientific_name      22164 non-null  object
 2   category             22164 non-null  object
 3   common_names         22164 non-null  object
 4   conservation_status  22164 non-null  object
 5   observations         22164 non-null  int64 
dtypes: int64(1), object(5)
memory usage: 1.0+ MB
```

In [49]:
```
# double-check species number
print(biodiversity["scientific_name"].nunique())

# expected number of rows; It is confirmed in the cells above that each species is observed in each park,
# therefore the expected number of rows is four times the number of species, after summing the observations for each park!
print(biodiversity["scientific_name"].nunique() * 4)

# double-check row number without multiple observations for the same park; Checks out!
print(len(biodiversity))
```

```
5541
22164
22164
```

In [50]:
```
missing_data_check(biodiversity)
```

```
Missing data check for "biodiversity" dataframe:

NaN values exist: False
Missing values exist: False
Empty string values exist: False

No missing or empty values found.
```

In [51]:
```python
# export the data into a "biodiversity.csv" file

biodiversity.to_csv("biodiversity.csv", index=False)
```

The "**biodiversity.csv**" file contains six columns, cleaned and merged from "**observations.csv**" and "**species.csv**":

- **Scientific Name**

  > There are **5541** distinct species in this dataset with corresponding Latin names.

- **Park Name**

  > There are **four distinct parks** in this dataset:
  >
  > 1. **Great Smoky Mountains National Park**
  > 2. **Yosemite National Park**
  > 3. **Bryce National Park**
  > 4. **Yellowstone National Park**

- **Observations**

  > The count of species sightings per park for each individual species. Each species appeares in each park.

- **Category**

  > There are seven distinct categories in the data:
  >
  > 1. **Mammal**
  > 2. **Bird**
  > 3. **Reptile**
  > 4. **Amphibian**
  > 5. **Fish**
  > 6. **Vascular Plant**
  > 7. **Nonvascular Plant**

- **Common Names**

  > Well-known names commonly recognized by the general population.

- **Conservation Status**

  > Refers to the assessment of the risk level faced by a species or ecosystem, indicating the extent to which it is threatened or protected based on scientific evaluation and criteria.
  > The dataset includes **five distinct conservation status groups**:
  >
  > 1. **Species of Concern**
  > 2. **Endangered**
  > 3. **Threatened**
  > 4. **In Recovery**
  > 5. **Least Concern**

Title Figure 1:

Biodiversity in Focus: Exploring Species Endangerment and Sightings across Four American National Parks

In [52]:
```python
# national park names, states and coordinates
data = {
    "Location": ["Bryce Canyon, Utah", "Great Smoky Mountains, Tennessee", "Yosemite, California", "Yellowstone, Wyoming"],
    "Latitude": [37.5930, 35.6131, 37.8651, 44.4279],
    "Longitude": [-112.1871, -83.5532, -119.5383, -110.5885]
}

# make a dataframe and plot the USA map with the data as annotated dots
df = pd.DataFrame(data)

# define the desired colors
colors = ["limegreen", "dodgerblue", "orange", "red"]

# generate a scatter plot on a geographical map of the USA, with customized appearance and layout settings
fig = px.scatter_geo(df, lat="Latitude", lon="Longitude", hover_name="Location")
fig.update_geos(visible=True, resolution=110, scope="usa", showcountries=True, countrycolor="black", showsubunits=True, subunitcolor="gray", landcolor="#f8f8f8")
fig.update_layout(autosize=True, margin={"r":0,"t":20,"l":0,"b":0})

# customize the dot colors
fig.update_traces(marker=dict(size=8, color=colors), selector=dict(type="scattergeo"))

# location annotation
fig.add_trace(
    go.Scattergeo(
        lat=df["Latitude"] + 1,
        lon=df["Longitude"],
        text=[
            "<b>Bryce Canyon, Utah</b>",
            "<b>Great Smoky Mountains, Tennessee</b>",
            "<b>Yosemite, California</b>",
            "<b>Yellowstone, Wyoming</b>"
        ],
        mode="text",
        textposition="middle center",
        showlegend=False,
        hoverinfo="none",
        textfont=dict(size=12, color=colors),
```

```
    )
)

# title
fig.add_annotation(
    x=0.5,
    y=1.05,
    text="<b>Biodiversity in Focus: Exploring Species Endangerment and Sightings across Four American National Parks</b>",
    showarrow=False,
    font=dict(size=16, color="black", family="Calibri"),
)

# lock the layout
fig.update_layout(dragmode=False, mapbox=dict(zoom=False))

# save the figure
fig.write_image("TitleFigure1.png", scale=3)

# display the figure
fig.show()
```

National Park Surface Areas

Yellowstone National Park
Approximately 8991 km² (3471 square miles).

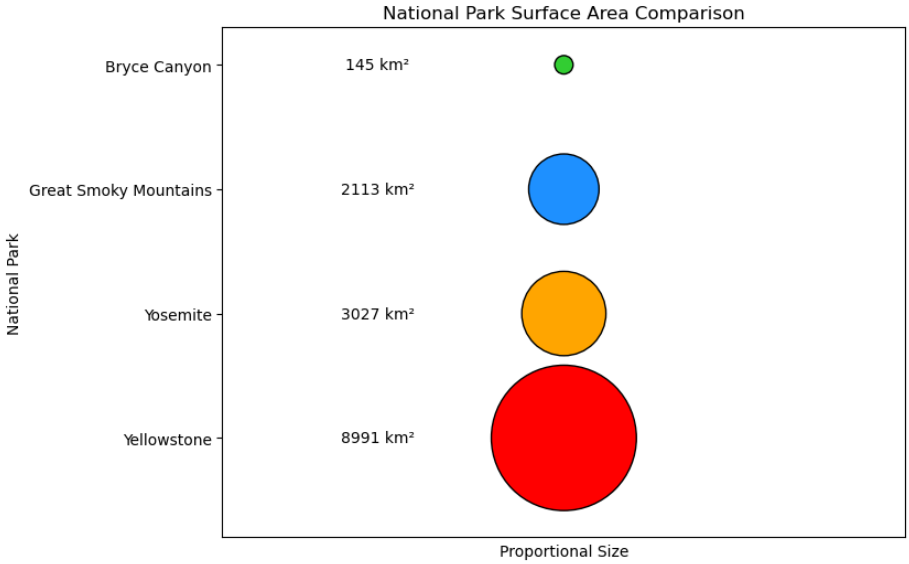
Yosemite National Park
Approximately 3027 km² (1168 square miles).

Great Smoky Mountains National Park
Approximately 2113 km² (815 square miles).

Bryce Canyon National Park
Approximately 145 km² (56 square miles).

Title Figure 2: National Park Surface Area Comparison

In [53]:
```
# adjust the plot size
plt.subplots(figsize=(8, 6))

# park names
parks = ["Yellowstone", "Yosemite", "Great Smoky Mountains", "Bryce Canyon"]

# park areas
areas_km = [8991, 3027, 2113, 145]

# set colors for the bubbles
colors = ["red", "orange", "dodgerblue", "limegreen"]

# create the bubble chart with switched axes
plt.scatter([1] * len(parks), parks, s=areas_km, c=colors, edgecolor="black")

# add labels and title
plt.xlabel("Proportional Size")
plt.ylabel("National Park")
plt.title("National Park Surface Area Comparison")

# set the y-axis limit and tick spacing
plt.ylim(-0.8, 3.3)
plt.yticks(range(len(parks)), parks)

# remove x-axis ticks
plt.xticks([])

# annotate each scatter point with the corresponding areas_km value
for i, area in enumerate(areas_km):
    plt.text(1-0.03, i, f"{area} km²", ha="center", va="center")

# save the figure
plt.savefig("TitleFigure2.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```



Yellowstone National Park is a breathtaking natural wonder located in **Wyoming**.
Renowned for its geothermal features, including the iconic Old Faithful geyser. **Yellowstone is the first national park in the United States and is recognized for its diverse wildlife and stunning landscapes**.

Yosemite National Park is a captivating destination in **California**.
Home to towering granite cliffs, cascading waterfalls, and ancient sequoia groves. Yosemite is celebrated for its awe-inspiring beauty and outdoor recreation opportunities.

Great Smoky Mountains National Park is a cherished treasure nestled between **Tennessee** and North Carolina.
This park is renowned for its mist-covered mountains, rich biodiversity, and vibrant fall foliage, making it one of the most visited national parks in the USA.

Bryce Canyon National Park is an extraordinary geological marvel situated in **Utah**.
Characterized by its distinctive orange-hued rock formations called hoodoos, Bryce Canyon offers unparalleled panoramic vistas and an opportunity to explore the unique beauty of the high desert landscape.

5: Questions and Analysis

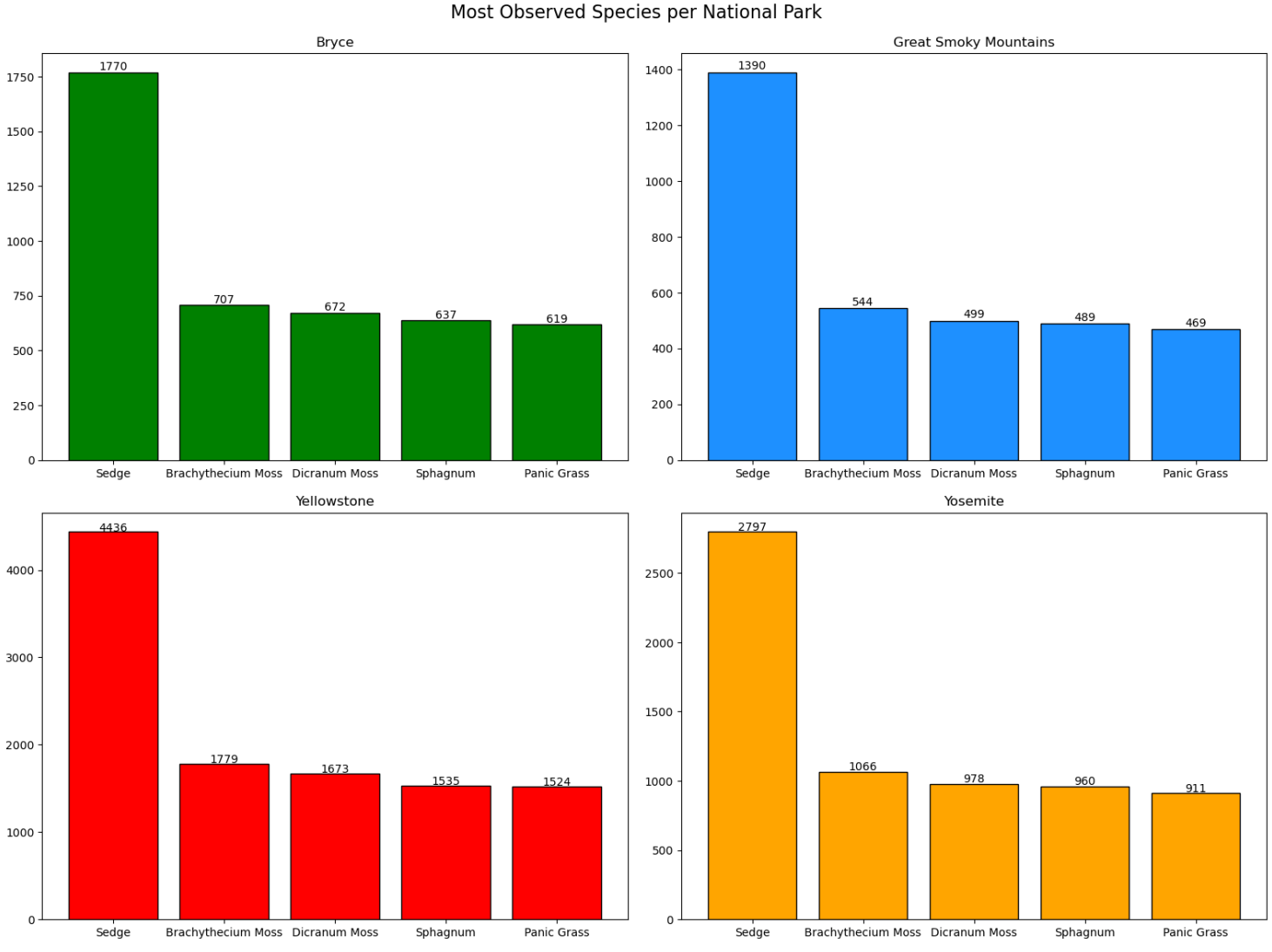
5.1: Which species were spotted the most at each park?

In [54]:
```
# most observed species per category
species_per_park = biodiversity.groupby(["park_name", "category", "common_names"]).observations.sum()

# display the top 5 observed species in each park
for park in species_per_park.index.levels[0]:
    sorted_per_park = species_per_park[park].sort_values(ascending=False)

    print(f"National Park: {park}")
    print(sorted_per_park.head(5), "\n")
```

```
National Park: Bryce
category           common_names
Vascular Plant     Sedge                1770
Nonvascular Plant  Dicranum Moss         707
                   Brachythecium Moss     672
                   Bryum Moss             637
Vascular Plant     Panic Grass            619
Name: observations, dtype: int64

National Park: Great Smoky Mountains
category           common_names
Vascular Plant     Sedge                1390
Nonvascular Plant  Dicranum Moss         544
                   Brachythecium Moss     499
Vascular Plant     Panic Grass            489
Nonvascular Plant  Bryum Moss             469
Name: observations, dtype: int64

National Park: Yellowstone
category           common_names
Vascular Plant     Sedge                4436
Nonvascular Plant  Dicranum Moss        1779
                   Brachythecium Moss   1673
                   Sphagnum             1535
Vascular Plant     Panic Grass          1524
Name: observations, dtype: int64

National Park: Yosemite
category           common_names
Vascular Plant     Sedge                2797
Nonvascular Plant  Brachythecium Moss   1066
                   Dicranum Moss         978
                   Sphagnum              960
Vascular Plant     Panic Grass           911
Name: observations, dtype: int64
```

Figure 1: Most Observed Species per National Park

In [55]:
```
# this observations distribution is another indicator that this dataset is fictional!

# define the parks and subplots layout
parks = species_per_park.index.levels[0]
num_parks = len(parks)

# define colors for each park
colors = ["green", "dodgerblue", "red", "orange"]

# create the figure and subplots
fig, axes = plt.subplots(2, 2, figsize=(16, 12))
axes = axes.flatten()

# iterate over each park
for i, park in enumerate(parks):
    sorted_species = species_per_park.loc[park].sort_values(ascending=False)

    # get the top 5 species for the current park
    top_species = sorted_species.head(5)

    # generate the bar graph
    ax = axes[i]
    ax.bar(top_species.index.get_level_values("common_names"), top_species.values,
           edgecolor="black", color=colors[i])

    # add annotations on top of each bar
    for j, v in enumerate(top_species.values):
        ax.text(j, v + 10, str(v), ha="center")

    ax.set_title(park)

# set the title for the whole figure
fig.suptitle("Most Observed Species per National Park", fontsize=16)

# set x-axis tick positions and labels
for ax in axes:
    ax.set_xticks(range(len(top_species.index)))
    ax.set_xticklabels(top_species.index.get_level_values("common_names"))

# hide empty subplots
if num_parks < 4:
    for i in range(num_parks, 4):
        fig.delaxes(axes[i])

# adjust the spacing between subplots
fig.tight_layout(pad=1.5)

# save the figure
plt.savefig("Figure1.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```



In [56]:
```
# most observed species per category
species_per_category = biodiversity.groupby(["category", "common_names"]).observations.sum()

# display the top 5 observed species in each category
for category in species_per_category.index.levels[0]:
    sorted_per_category = species_per_category[category].sort_values(ascending=False)
```

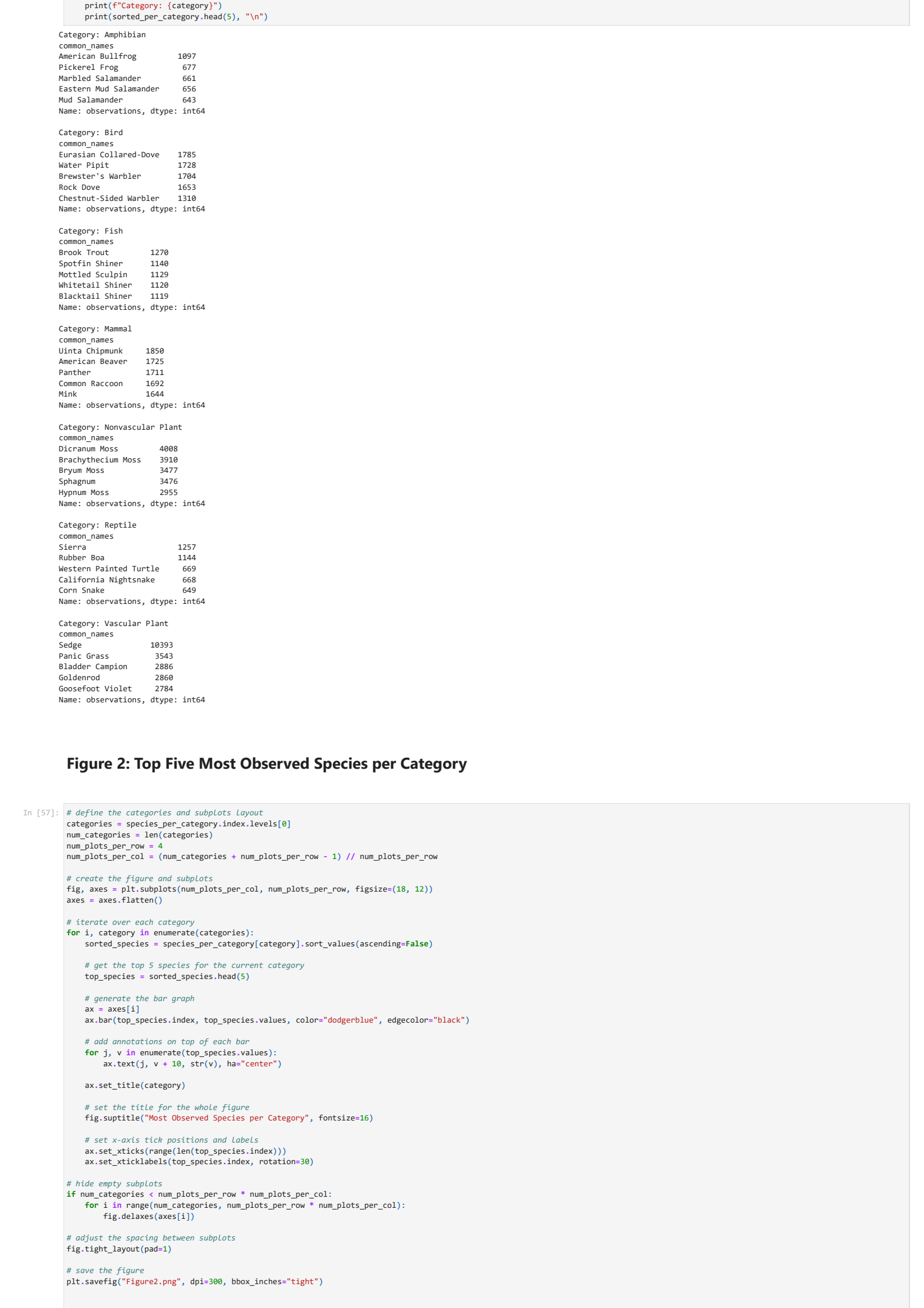
```
    print(f"Category: {category}")
    print(sorted_per_category.head(5), "\n")
```

```
Category: Amphibian
common_names
American Bullfrog         1097
Pickerel Frog              677
Marbled Salamander         661
Eastern Mud Salamander     656
Mud Salamander             643
Name: observations, dtype: int64

Category: Bird
common_names
Eurasian Collared-Dove    1785
Water Pipit               1728
Brewster's Warbler        1704
Rock Dove                 1653
Chestnut-Sided Warbler    1310
Name: observations, dtype: int64

Category: Fish
common_names
Brook Trout         1270
Spotfin Shiner      1140
Mottled Sculpin     1129
Whitetail Shiner    1120
Blacktail Shiner    1119
Name: observations, dtype: int64

Category: Mammal
common_names
Uinta Chipmunk     1850
American Beaver    1725
Panther            1711
Common Raccoon     1692
Mink               1644
Name: observations, dtype: int64

Category: Nonvascular Plant
common_names
Dicranum Moss         4008
Brachythecium Moss    3910
Bryum Moss            3477
Sphagnum              3476
Hypnum Moss           2955
Name: observations, dtype: int64

Category: Reptile
common_names
Sierra                    1257
Rubber Boa                1144
Western Painted Turtle     669
California Nightsnake      668
Corn Snake                 649
Name: observations, dtype: int64

Category: Vascular Plant
common_names
Sedge               10393
Panic Grass          3543
Bladder Campion      2886
Goldenrod            2860
Goosefoot Violet     2784
Name: observations, dtype: int64
```

Figure 2: Top Five Most Observed Species per Category

In [57]:

```
# define the categories and subplots layout
categories = species_per_category.index.levels[0]
num_categories = len(categories)
num_plots_per_row = 4
num_plots_per_col = (num_categories + num_plots_per_row - 1) // num_plots_per_row

# create the figure and subplots
fig, axes = plt.subplots(num_plots_per_col, num_plots_per_row, figsize=(18, 12))
axes = axes.flatten()

# iterate over each category
for i, category in enumerate(categories):
    sorted_species = species_per_category[category].sort_values(ascending=False)

    # get the top 5 species for the current category
    top_species = sorted_species.head(5)

    # generate the bar graph
    ax = axes[i]
    ax.bar(top_species.index, top_species.values, color="dodgerblue", edgecolor="black")

    # add annotations on top of each bar
    for j, v in enumerate(top_species.values):
        ax.text(j, v + 10, str(v), ha="center")

    ax.set_title(category)

    # set the title for the whole figure
    fig.suptitle("Most Observed Species per Category", fontsize=16)

    # set x-axis tick positions and labels
    ax.set_xticks(range(len(top_species.index)))
    ax.set_xticklabels(top_species.index, rotation=30)

# hide empty subplots
if num_categories < num_plots_per_row * num_plots_per_col:
    for i in range(num_categories, num_plots_per_row * num_plots_per_col):
        fig.delaxes(axes[i])

# adjust the spacing between subplots
fig.tight_layout(pad=1)

# save the figure
plt.savefig("Figure2.png", dpi=300, bbox_inches="tight")
```



```
# display the figure
plt.show()
```

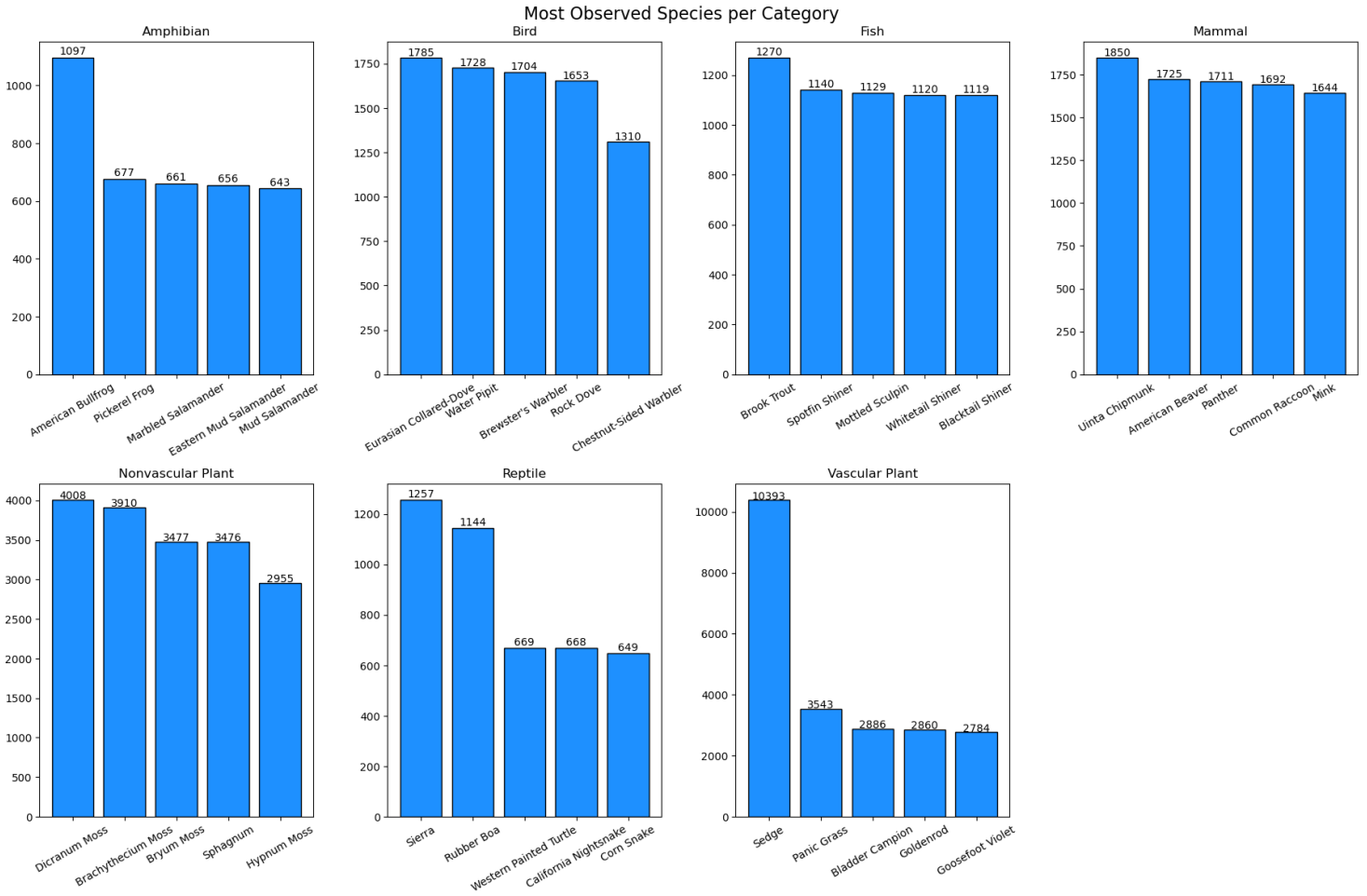


Figure 3: Total Observations per Category

In [58]:
```
# adjust the plot size
plt.subplots(figsize=(8, 6))

# sum and sort the observations for each category
total_observations_per_category = biodiversity.groupby("category")["observations"].sum().reset_index().sort_values("observations", ascending=False)

# divide by thousand for readability
total_observations_per_category["observations"] = total_observations_per_category["observations"] / 1000

# determine the color map
cmap = sns.color_palette("coolwarm_r", len(total_observations_per_category))

# barplot
sns.barplot(data=total_observations_per_category, x="category", y="observations", ec="k", palette=cmap)
plt.title("Total Observations per Category")
plt.xlabel("Category")
plt.ylabel("Number of Observations (thousands)")

# add annotation for readability
for index, value in enumerate(total_observations_per_category["observations"]):
    plt.text(index, value, f"{(value/total_observations_per_category.observations.sum() * 100):.0f}%", ha="center", va="bottom")

# format x and y-axis labels
format = ticker.FuncFormatter(lambda x, pos: f"{x:.0f}K")
plt.gca().yaxis.set_major_formatter(format)
plt.xticks(rotation=25)

# save the figure
plt.savefig("Figure3.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```

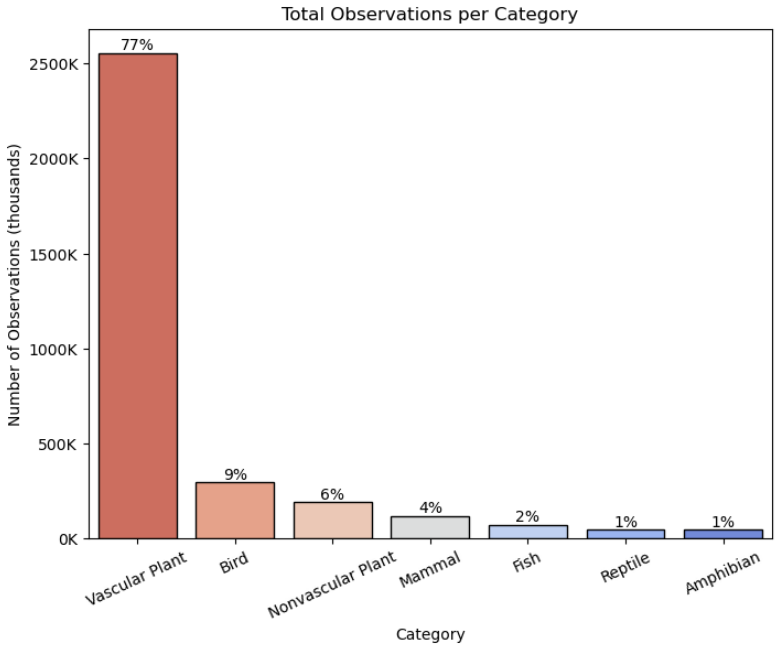


Figure 4: Total Observations per National Park

In [59]:
```
# adjust the plot size
plt.subplots(figsize=(8, 6))

# sum and sort the observations for each park
total_observations_per_park = biodiversity.groupby("park_name")["observations"].sum().reset_index().sort_values("observations", ascending=False)

# divide by thousand for readability
total_observations_per_park["observations"] = total_observations_per_park["observations"] / 1000

# set custom colors for each bar
colors = ["red", "orange", "dodgerblue", "limegreen"]

# barplot with custom colors
sns.barplot(data=total_observations_per_park, x="park_name", y="observations", ec="k", palette=colors)
plt.title("Total Observations per National Park")
plt.xlabel("National Park")
plt.ylabel("Number of Observations (thousands)")

# add annotation for easier graph reading
for index, value in enumerate(total_observations_per_park["observations"]):
    plt.text(index, value, f"{(value/total_observations_per_park.observations.sum() * 100):.0f}%", ha="center", va="bottom")

# format y-axis labels
format = ticker.FuncFormatter(lambda x, pos: f"{x:.0f}K")
plt.gca().yaxis.set_major_formatter(format)

# save the figure
plt.savefig("Figure4.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```

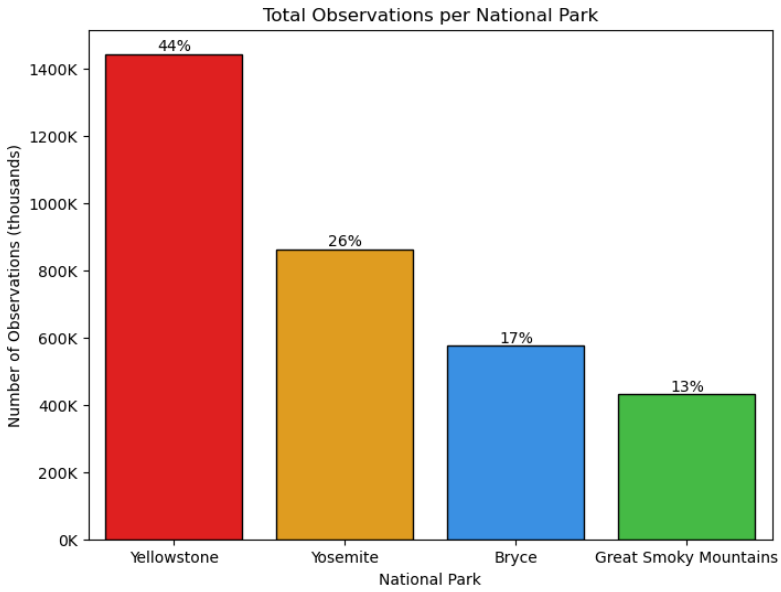


Figure 5: Observations per Category per National Park

In [60]:
```
# group the dataframe by park names and category and calculate the sum of observations
grouped_data = biodiversity.groupby(["park_name", "category"])["observations"].sum().reset_index()
```

```
# pivot the dataframe to have park names as rows, category as columns, and observations as values
pivoted_data = grouped_data.pivot(index="park_name", columns="category", values="observations")

# specify the desired order of parks for sorting
park_order = ["Yellowstone", "Yosemite", "Bryce", "Great Smoky Mountains"]

# sort the pivoted_data DataFrame based on the specified park order
pivoted_data = pivoted_data.loc[park_order]

# sort the columns within each park in descending order
pivoted_data = pivoted_data[pivoted_data.sum().sort_values(ascending=False).index]

# create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# plot the first subplot
ax1 = axes[0]
pivoted_data.plot(kind="bar", stacked=False, edgecolor="black", width=0.8, ax=ax1)

# format x and y-axis
ax1.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: "{:.0f}K".format(x * 1e-3)))
ax1.set_xticklabels(pivoted_data.index, rotation=0)

# add axis labels, title, and legend
ax1.set_xlabel("National Park")
ax1.set_ylabel("Number of Observations (thousands)")
ax1.set_title("Observations per Category per National Park")
ax1.legend(title="Category", frameon=False, fontsize=8)

# plot the second subplot with logarithmic y-scale
ax2 = axes[1]
pivoted_data.plot(kind="bar", stacked=False, edgecolor="black", width=0.8, ax=ax2)

# format x and y-axis with logarithmic scale
ax2.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos: "{:.0f}K".format(x * 1e-3)))
ax2.set_xticklabels(pivoted_data.index, rotation=0)
# logarithmic scale allows for a better understanding of the relationship between the categories in each park.
# It is important to address that this distorts the bigger picture.
# For example, Yellowstone has the majority of observations across all categories, especially for vascular plants.
ax2.set_yscale("log")

# add axis labels, title, and legend
ax2.set_xlabel("National Park")
ax2.set_ylabel("Number of Observations")
ax2.set_title("Observations per Category per National Park (Logarithmic Scale)")
ax2.legend(title="Category", frameon=False, fontsize=8, loc=(0.78,0.706))

# adjust the spacing between subplots
plt.tight_layout(pad=2.5)

# save the figure
plt.savefig("Figure5.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```

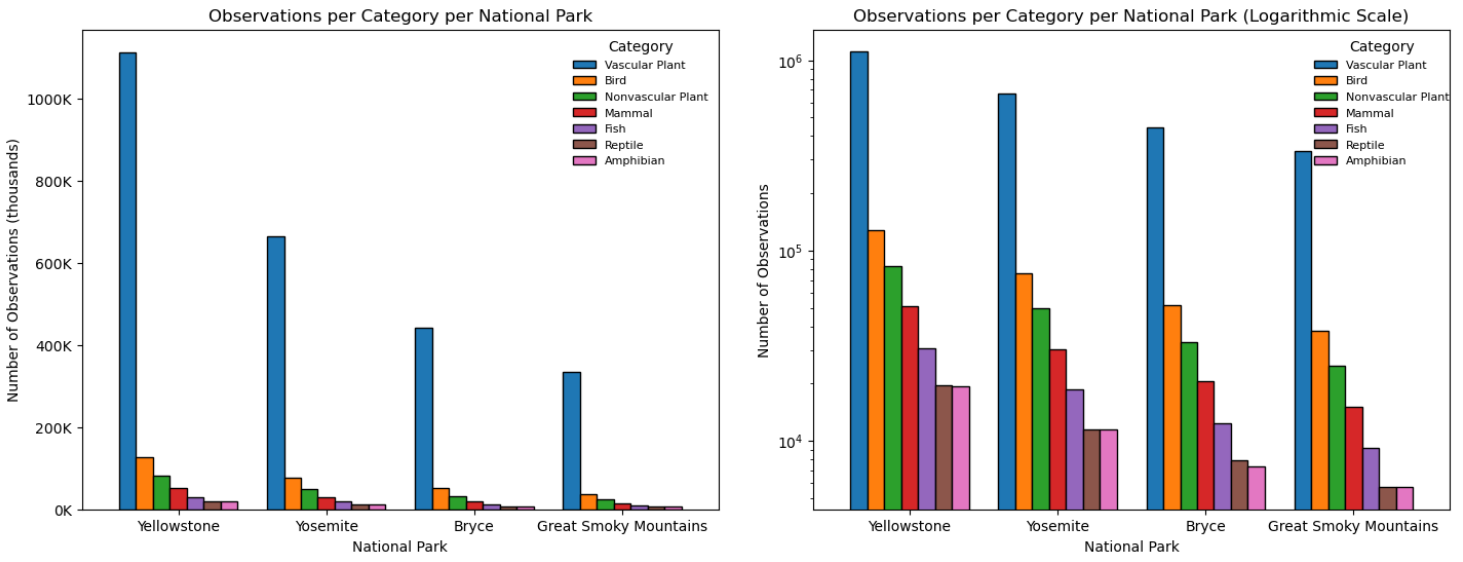


Figure 6: Species Distribution by Category

In [61]:
```
# adjust the plot size
plt.subplots(figsize=(8, 6))

# count species by category
species_category = pd.DataFrame(biodiversity.category.value_counts())

# calculate the percentage of each category
species_category["percentage"] = (species_category["count"] / species_category["count"].sum() * 100)

# x axis is for categories
x = species_category.index

# determine the color map
cmap = sns.light_palette("blue", len(species_category), reverse=True)

# seaborn bar plot
sns.barplot(data=species_category, x=x, y="percentage", ec="k", palette=cmap)
plt.title("Species Distribution by Category")
plt.xlabel("Category")
plt.ylabel("Percent / %")
plt.xticks(rotation=25)

# add annotation for easier graph reading
for index, value in enumerate(species_category["percentage"]):
    plt.text(index, value, f"{value:.2f}%", ha="center", va="bottom")
```

```
# save the figure
plt.savefig("Figure6.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```

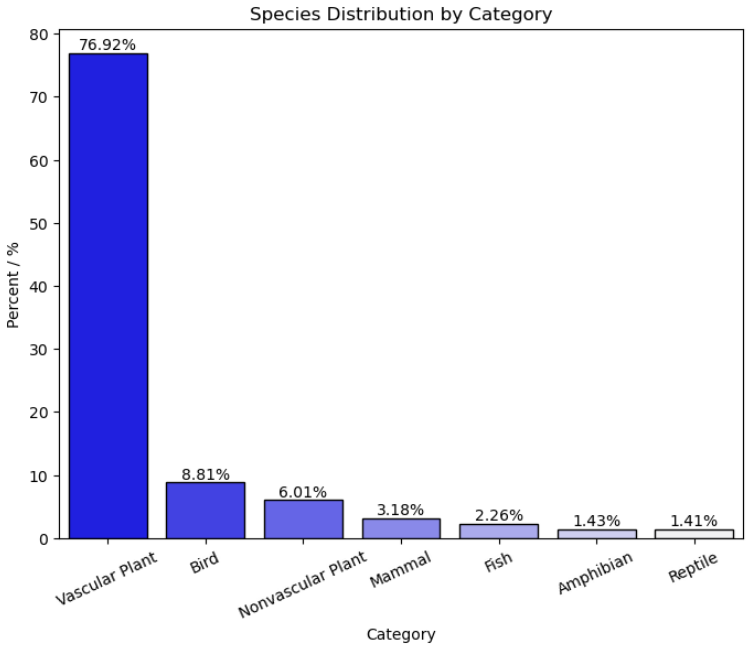


Table 1: Species Distribution by Category: Count and Percentage

In [62]:
```
display(species_category.style.set_caption("Species Distribution by Category: Count and Percentage"))
```

Species Distribution by Category: Count and Percentage

category	count	percentage
Vascular Plant	17048	76.917524
Bird	1952	8.807075
Nonvascular Plant	1332	6.009746
Mammal	704	3.176322
Fish	500	2.255910
Amphibian	316	1.425735
Reptile	312	1.407688

5.2: What is the Distribution of Conservation Status for Animals?

Figure 7: Overall Conservation Status Distribution (Logarithmic scale)

In [63]:
```
# adjust the plot size
plt.subplots(figsize=(8, 6))

# count species by category
conservation_status = pd.DataFrame(biodiversity.conservation_status.value_counts())

# calculate and display the percentage of each category
conservation_status["percentage"] = (conservation_status["count"] / conservation_status["count"].sum() * 100)
display(conservation_status.style.set_caption("Overall Conservation Status Distribution: Count and Percentage"))

# x axis is for conservation status
x = biodiversity.conservation_status.unique()

# determine the color map
cmap = sns.light_palette("magenta", len(conservation_status))

# seaborn bar plot
sns.barplot(data=conservation_status, x=x, y="percentage", ec="k", palette=cmap)
plt.title("Overall Conservation Status Distribution (Logarithmic scale)")
plt.xlabel("Conservation Status")
plt.ylabel("Percent / %")

# add annotation for easier graph reading
for index, value in enumerate(conservation_status["percentage"]):
    plt.text(index, value, f"{value:.2f}%", ha="center", va="bottom")

# logarithmic scale for better readability
plt.yscale("log")

# save the figure
plt.savefig("Figure7.png", dpi=300, bbox_inches="tight")
```

```
# display the figure
plt.show()
```

Overall Conservation Status Distribution:
Count and Percentage

conservation_status	count	percentage
Least concern	21452	96.787583
Species of Concern	604	2.725140
Endangered	60	0.270709
Threatened	36	0.162426
In Recovery	12	0.054142

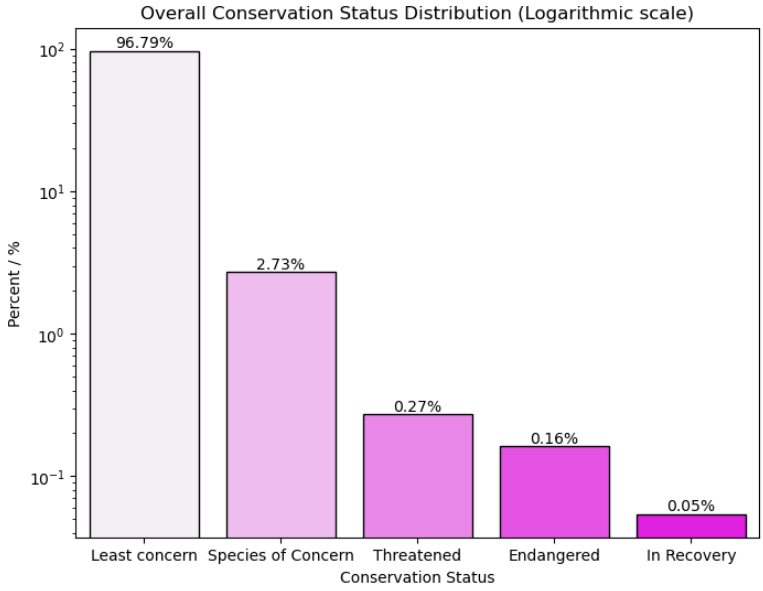


Table 2: Conservation Status Distribution per Park

In [64]:
```
# counting conservation status per park
conservation_status_per_park = pd.DataFrame(biodiversity.groupby(["park_name", "conservation_status"]).size().unstack(fill_value=0))

# there is no need to plot this beacuse the amount of each conservation status is exactly the same for each national park;
# This is a fine indication that this is a fictional dataset!
display(conservation_status_per_park.style.set_caption("Conservation Status Distribution per Park"))
```

Conservation Status Distribution per Park

conservation_status	Endangered	In Recovery	Least concern	Species of Concern	Threatened
park_name					
Bryce	15	3	5363	151	9
Great Smoky Mountains	15	3	5363	151	9
Yellowstone	15	3	5363	151	9
Yosemite	15	3	5363	151	9

Figure 8: Conservation Status Distribution per Category (Logarithmic scale)

In [65]:
```
# create subplots with desired figure size
fig, ax = plt.subplots(figsize=(8, 6))

# count the conservation status for each category
conservation_status_per_category = biodiversity.groupby(["category", "conservation_status"]).size().unstack(fill_value=0)

# sort each category by conservation status occurrence from highest to lowest
conservation_status_per_category = conservation_status_per_category.reindex(conservation_status_per_category.sum(axis=1).sort_values(ascending=False).index)

# sort each conservation status within each category
conservation_status_per_category = conservation_status_per_category[conservation_status_per_category.sum().sort_values(ascending=False).index]

# plotting the bar chart
conservation_status_per_category.plot(kind="barh", stacked=False, edgecolor="black", width=0.8, ax=ax)

# customizing the plot
ax.set_title("Conservation Status Distribution per Category (Logarithmic scale)")
ax.set_xlabel("Count")
ax.set_ylabel("Category")
# logarithmic scale for better readability
ax.set_xscale("log")
ax.legend(frameon=False)
# adjusting the height to increase the space between each bar group
ax.set_ylim([-0.55, len(conservation_status_per_category)-0.9])
ax.invert_yaxis()
plt.legend(frameon=False, title="Conservation Status")

# save the figure
plt.savefig("Figure8.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```

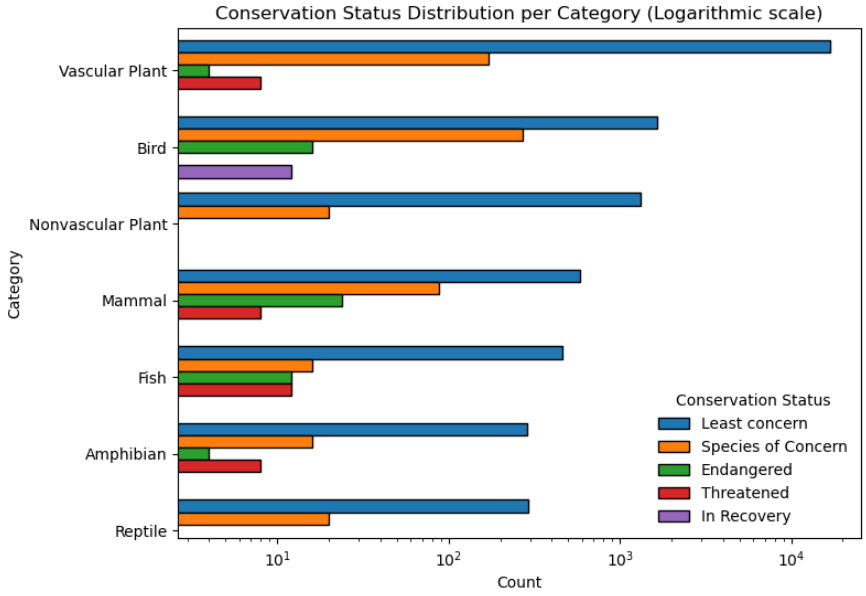


Table 3: Conservation Status Distribution Per Category

In [66]:
```
# add a row called "total" with the sum of each conservation status column
conservation_status_per_category.loc["Total"] = conservation_status_per_category.sum()

# add a column called "total" with the total sum for each category
conservation_status_per_category["Total"] = conservation_status_per_category.sum(axis=1)

# display the table
display(conservation_status_per_category.style.set_caption("Conservation Status Distribution Per Category"))
```

Conservation Status Distribution Per Category

conservation_status category	Least concern	Species of Concern	Endangered	Threatened	In Recovery	Total
Vascular Plant	16864	172	4	8	0	17048
Bird	1652	272	16	0	12	1952
Nonvascular Plant	1312	20	0	0	0	1332
Mammal	584	88	24	8	0	704
Fish	460	16	12	12	0	500
Amphibian	288	16	4	8	0	316
Reptile	292	20	0	0	0	312
Total	21452	604	60	36	12	22164

5.3: Are certain types of species more likely to be endangered?

Table 4: Conservation Status Distribution Per Category in Percentages

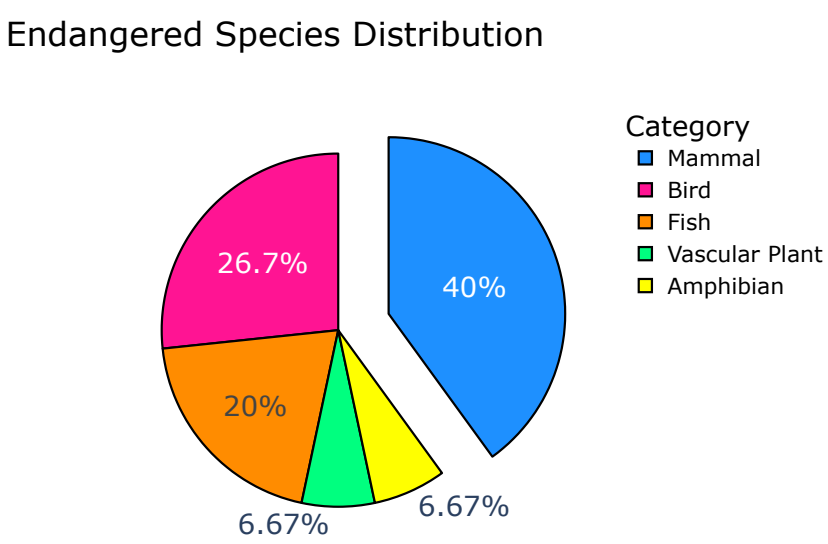
In [67]:
```
# table to calculate the percentage of conservation status per category
endangerment_table = conservation_status_per_category.copy().drop(labels="Total", axis=1)

# calculate the percentage for each row
endangerment_table_percent = endangerment_table.copy()
for column in endangerment_table.columns:
        endangerment_table_percent[column] = endangerment_table_percent[column] / endangerment_table_percent[column]["Total"] * 100

display(endangerment_table_percent.style.set_caption("Conservation Status Distribution per Category in Percentages").format("{:.2f}"))
```

Conservation Status Distribution per Category in Percentages

conservation_status category	Least concern	Species of Concern	Endangered	Threatened	In Recovery
Vascular Plant	78.61	28.48	6.67	22.22	0.00
Bird	7.70	45.03	26.67	0.00	100.00
Nonvascular Plant	6.12	3.31	0.00	0.00	0.00
Mammal	2.72	14.57	40.00	22.22	0.00
Fish	2.14	2.65	20.00	33.33	0.00
Amphibian	1.34	2.65	6.67	22.22	0.00
Reptile	1.36	3.31	0.00	0.00	0.00
Total	100.00	100.00	100.00	100.00	100.00

Figure 9: Endangered Species Distribution

In [68]:
```
# dataframe containing only endangered species
endangered_species = biodiversity.loc[biodiversity["conservation_status"] == "Endangered"]

# endangered species count per category
endangered_category_distribution = endangered_species.category.value_counts()

# pie chart colors
colors = ["DodgerBlue", "DeepPink", "DarkOrange", "SpringGreen", "Yellow"]

# pie chart with modifications
fig = go.Figure(data=[go.Pie(
    labels=endangered_category_distribution.index,
    values=endangered_category_distribution,
    pull=[0.3],
    insidetextorientation="horizontal")]
)

# pie chart design
fig.update_layout(width=800, height=600, title={"text": "Endangered Species Distribution", "font": {"size": 30, "color": "black"}})
fig.update_traces(hoverinfo="label+value",
                  textinfo="percent",
                  textfont_size=25,
                  marker=dict(colors=colors, line=dict(color="#000000", width=2))
)

# pie chart design
fig.update_layout(
    title={
        "text": "Endangered Species Distribution",
        "font": {"size": 30}},
    legend=dict(
        title="Category",
        title_font=dict(size=25, color="black"),
        font=dict(size=20, color="black"),
    )
)

# save the figure
fig.write_image("Figure9.png", scale=2)

# display the figure
fig.show()
```



In [69]:
```
display(endangered_category_distribution)
```

```
category
Mammal            24
Bird              16
Fish              12
Vascular Plant     4
Amphibian          4
Name: count, dtype: int64
```

In [70]:
```
endangered_mammals = biodiversity[(biodiversity["conservation_status"] == "Endangered") & (biodiversity["category"] == "Mammal")].sort_values("observations", ascending=False)
```

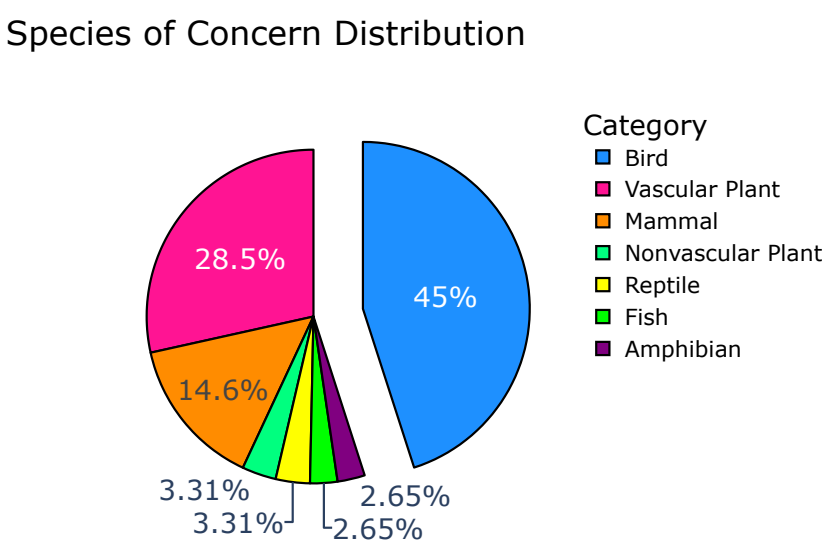
In [71]:
```
display(endangered_mammals.head(10).style.set_caption("Top 10 Endangered Mammal Species"))
```

Top 10 Endangered Mammal Species

	park_name	scientific_name	category	common_names	conservation_status	observations
11886	Yellowstone	Canis lupus	Mammal	Gray Wolf	Endangered	330
17427	Yosemite	Canis lupus	Mammal	Gray Wolf	Endangered	196
804	Bryce	Canis lupus	Mammal	Gray Wolf	Endangered	130
14406	Yellowstone	Myotis sodalis	Mammal	Indiana Bat	Endangered	68
14401	Yellowstone	Myotis grisescens	Mammal	Gray Myotis	Endangered	68
13374	Yellowstone	Glaucomys sabrinus coloratus	Mammal	Carolina Northern Flying Squirrel	Endangered	67
14574	Yellowstone	Ovis canadensis sierrae	Mammal	Sierra Nevada Bighorn Sheep	Endangered	67
11887	Yellowstone	Canis rufus	Mammal	Red Wolf	Endangered	60
6345	Great Smoky Mountains	Canis lupus	Mammal	Gray Wolf	Endangered	59
19942	Yosemite	Myotis grisescens	Mammal	Gray Myotis	Endangered	39

Figure 10: Species of Concern Distribution

In [72]:
```
# dataframe containing only species of concern
species_of_concern = biodiversity.loc[biodiversity["conservation_status"] == "Species of Concern"]

# endangered species count per category
species_of_concern_distribution_by_category = species_of_concern.category.value_counts()

# pie chart colors
colors = ["DodgerBlue", "DeepPink", "DarkOrange", "SpringGreen", "Yellow", "Lime", "Purple"]

# pie chart with modifications
fig = go.Figure(data=[go.Pie(
    labels=species_of_concern_distribution_by_category.index,
    values=species_of_concern_distribution_by_category,
    pull=[0.3],
    insidetextorientation="horizontal")]
)

# pie chart design
fig.update_layout(width=800, height=600, title={"text": "Endangered Species Distribution", "font": {"size": 30, "color": "black"}})
fig.update_traces(hoverinfo="label+value",
                  textinfo="percent",
                  textfont_size=25,
                  marker=dict(colors=colors, line=dict(color="#000000", width=2))
)

# pie chart design
fig.update_layout(
    title={
        "text": "Species of Concern Distribution",
        "font": {"size": 30}},
    legend=dict(
        title="Category",
        title_font=dict(size=25, color="black"),
        font=dict(size=20, color="black"),
    )
)

# save the figure
fig.write_image("Figure10.png", scale=2)

# display the figure
fig.show()
```



In [73]:
```
display(species_of_concern_distribution_by_category)
```

```
category
Bird                272
Vascular Plant      172
Mammal               88
Nonvascular Plant    20
Reptile              20
Fish                 16
Amphibian            16
Name: count, dtype: int64
```

In [74]:
```
species_of_concern_birds = biodiversity[(biodiversity["conservation_status"] == "Species of Concern") & (biodiversity["category"] == "Bird")].sort_values("observations", ascending=Fals
```

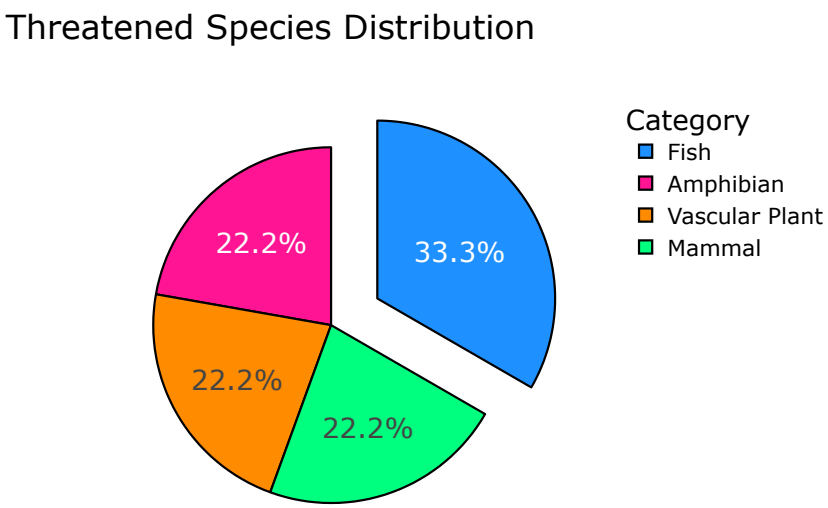
In [75]:
```
display(species_of_concern_birds.head(10).style.set_caption("Top 10 Species of Concern Bird Species"))
```

Top 10 Species of Concern Bird Species

	park_name	scientific_name	category	common_names	conservation_status	observations
14608	Yellowstone	Pandion haliaetus	Bird	Osprey	Species of Concern	466
14487	Yellowstone	Nycticorax nycticorax	Bird	Black-Crowned Night-Heron	Species of Concern	439
15456	Yellowstone	Riparia riparia	Bird	Bank Swallow	Species of Concern	425
13307	Yellowstone	Gavia immer	Bird	Common Loon	Species of Concern	408
18848	Yosemite	Gavia immer	Bird	Common Loon	Species of Concern	319
20149	Yosemite	Pandion haliaetus	Bird	Osprey	Species of Concern	282
20997	Yosemite	Riparia riparia	Bird	Bank Swallow	Species of Concern	270
20028	Yosemite	Nycticorax nycticorax	Bird	Black-Crowned Night-Heron	Species of Concern	269
13953	Yellowstone	Limosa fedoa	Bird	Marbled Godwit	Species of Concern	266
13192	Yellowstone	Falco columbarius	Bird	Merlin	Species of Concern	258

Figure 11: Threatened Species Distribution

In [76]:
```
# dataframe containing only species of concern
threatened_species = biodiversity.loc[biodiversity["conservation_status"] == "Threatened"]

# endangered species count per category
threatened_species_by_category = threatened_species.category.value_counts()

# pie chart colors
colors = ["DodgerBlue", "DeepPink", "DarkOrange", "SpringGreen", "Yellow", "Lime", "Purple"]

# pie chart with modifications
fig = go.Figure(data=[go.Pie(
    labels=threatened_species_by_category.index,
    values=threatened_species_by_category,
    pull=[0.3],
    insidetextorientation="horizontal")]
)

# pie chart design
fig.update_layout(width=800, height=600, title={"text": "Endangered Species Distribution", "font": {"size": 30, "color": "black"}})
fig.update_traces(hoverinfo="label+value",
                  textinfo="percent",
                  textfont_size=25,
                  marker=dict(colors=colors, line=dict(color="#000000", width=2))
)

# pie chart design
fig.update_layout(
    title={
        "text": "Threatened Species Distribution",
        "font": {"size": 30}},
    legend=dict(
        title="Category",
        title_font=dict(size=25, color="black"),
        font=dict(size=20, color="black"),
    )
)

# save the figure
fig.write_image("Figure11.png", scale=2)

# display the figure
fig.show()
```



In [77]:
```
display(threatened_species_by_category)
```

```
category
Fish              12
Amphibian          8
Vascular Plant     8
Mammal             8
Name: count, dtype: int64
```

In [78]:
```
threatened_fish = biodiversity[(biodiversity["conservation_status"] == "Threatened") & (biodiversity["category"] == "Fish")].sort_values("observations", ascending=False)
```

In [79]:
```
display(threatened_fish.head(10).style.set_caption("Top 10 Threatened Fish Species"))
```

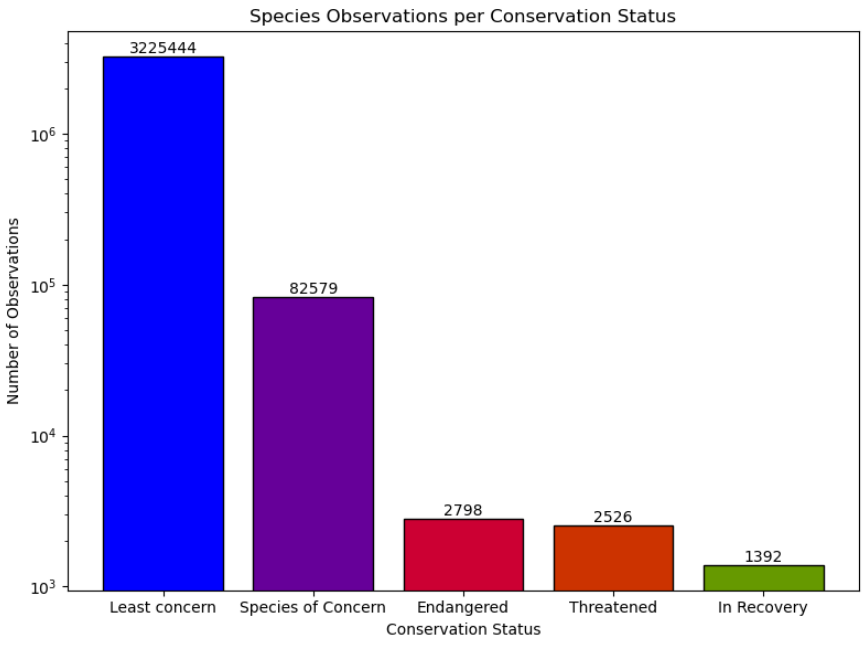
Top 10 Threatened Fish Species

	park_name	scientific_name	category	common_names	conservation_status	observations
14477	Yellowstone	Noturus flavipinnis	Fish	Yellowfin Madtom	Threatened	126
14510	Yellowstone	Oncorhynchus clarkii henshawi	Fish	Lahontan Cutthroat Trout	Threatened	111
13043	Yellowstone	Erimonax monachus	Fish	Spotfin Chub	Threatened	109
20051	Yosemite	Oncorhynchus clarkii henshawi	Fish	Lahontan Cutthroat Trout	Threatened	85
20018	Yosemite	Noturus flavipinnis	Fish	Yellowfin Madtom	Threatened	72
18584	Yosemite	Erimonax monachus	Fish	Spotfin Chub	Threatened	69
3428	Bryce	Oncorhynchus clarkii henshawi	Fish	Lahontan Cutthroat Trout	Threatened	57
3395	Bryce	Noturus flavipinnis	Fish	Yellowfin Madtom	Threatened	55
8969	Great Smoky Mountains	Oncorhynchus clarkii henshawi	Fish	Lahontan Cutthroat Trout	Threatened	48
1961	Bryce	Erimonax monachus	Fish	Spotfin Chub	Threatened	46

5.4: Are the differences between species and their conservation status significant?

Figure 12: Species Observations per Conservation Status

In [80]:
```
# sum the number of observations for each conservation status
observations_per_conservation_status = biodiversity.groupby("conservation_status")["observations"].sum().sort_values(ascending=False)

# color mapped bar chart
plt.figure(figsize=(8, 6))
colors = plt.cm.brg_r(np.linspace(1, 0.2, len(observations_per_conservation_status)))
plt.bar(observations_per_conservation_status.index, observations_per_conservation_status.values, edgecolor="black", color=colors)
plt.xlabel("Conservation Status")
plt.ylabel("Number of Observations")
plt.title("Species Observations per Conservation Status")
plt.tight_layout()
plt.yscale("log")

# add annotations for readability
for index, value in enumerate(observations_per_conservation_status):
    plt.text(index, value, f"{value:.0f}", ha="center", va="bottom")

# save the figure
plt.savefig("Figure12.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```



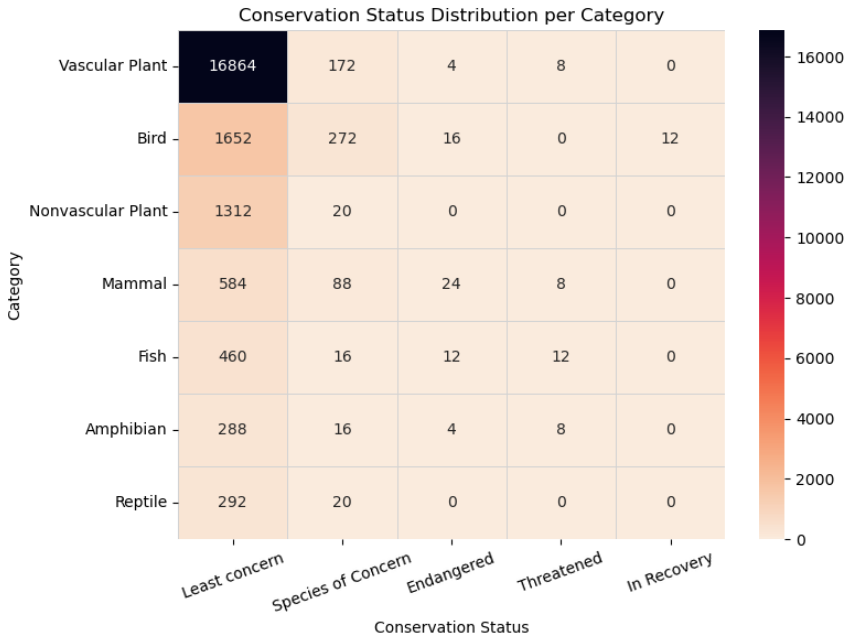
In [81]:
```
observations_per_conservation_status
```

Out[81]:
```
conservation_status
Least concern         3225444
Species of Concern      82579
Endangered               2798
Threatened               2526
In Recovery              1392
Name: observations, dtype: int64
```

Figure 13: Conservation Status Distribution per Category

In [82]:
```
# count the conservation status for each category
conservation_status_per_category = biodiversity.groupby(["category", "conservation_status"])["scientific_name"].size().unstack(fill_value=0)
```

```
# sort each category by conservation status occurrence from highest to lowest
conservation_status_per_category = conservation_status_per_category.reindex(conservation_status_per_category.sum(axis=1).sort_values(ascending=False).index)

# sort each conservation status within each category
conservation_status_per_category = conservation_status_per_category[conservation_status_per_category.sum().sort_values(ascending=False).index]

# create a heatmap
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(conservation_status_per_category, annot=True, fmt="d", cmap="rocket_r", cbar=True, ax=ax, linewidths=0.5, linecolor="lightgray")

# customizing the plot
ax.set_title("Conservation Status Distribution per Category")
ax.set_xlabel("Conservation Status")
ax.set_ylabel("Category")
ax.set_xticklabels(ax.get_xticklabels(), rotation=20)

# save the figure
plt.savefig("Figure13.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```



5.4.1: Shannon-Weaver and Simpson's Index for each National Park

In [83]:
```
# calculate the Shannon-Weaver index for each National Park
shannon_indices = biodiversity.groupby("park_name")["observations"].apply(lambda x: entropy(x.value_counts(normalize=True)))

# calculate the Simpson's index for each National Park
simpsons_indices = biodiversity.groupby("park_name")["observations"].apply(lambda x: 1 - np.sum((x.value_counts() / len(x)) ** 2))
```

Table 5: Shannon-Weaver and Simpson's Index for each National Park

In [84]:
```
# print the calculated indices
print("Shannon-Weaver Index:")
print(round(shannon_indices, 4))

print("\nSimpson's Index:")
print(round(simpsons_indices, 4))
```

```
Shannon-Weaver Index:
park_name
Bryce                   4.6192
Great Smoky Mountains   4.5724
Yellowstone             4.6472
Yosemite                4.6353
Name: observations, dtype: float64

Simpson's Index:
park_name
Bryce                   0.9873
Great Smoky Mountains   0.9871
Yellowstone             0.9875
Yosemite                0.9873
Name: observations, dtype: float64
```

Both the **Shannon-Weaver** and **Simpson's indices** are used to **measure biodiversity**, but they provide slightly different perspectives on the diversity within each park.

The **Shannon-Weaver index takes into account both species richness (the number of species present) and evenness (how evenly distributed the individuals are among species)**. A higher Shannon-Weaver index value indicates higher biodiversity. In this case, **all four parks have relatively high Shannon-Weaver index values**. This suggests that **all the parks exhibit considerable species richness and evenness in terms of the observed species**.

On the other hand, the **Simpson's index focuses more on species dominance**. It measures the probability that two individuals randomly selected from the population belong to the same species. The Simpson's index ranges from 0 to 1, with higher values indicating lower diversity and higher dominance of a few species. In this case, **all four parks have high Simpson's index values, indicating a relatively low dominance of any specific species and a high level of diversity**.

Overall, these results suggest that **all four parks have relatively high biodiversity, with a diverse array of species present and a relatively even distribution of individuals among species**.

5.4.2: Chi-squared test and Fisher's, Barnard's and Boschloo's exact tests

- **Chi-squared (χ^2) Test**:
  (https://en.wikipedia.org/wiki/Chi-squared_test)
  The chi-square test is used to determine the association between two categorical variables. It compares the observed frequencies in a contingency table with the expected frequencies under the assumption of independence. The test statistic, χ^2, is calculated based on the differences between observed and expected frequencies. The p-value is then derived from the chi-square distribution with appropriate degrees of freedom. A low p-value suggests that there is a significant association between the variables.

- **Fisher's Exact Test**:
  (https://en.wikipedia.org/wiki/Fisher%27s_exact_test)
  Fisher's exact test is also used to analyze the association between categorical variables. It is commonly used when the sample size is small, or when any expected cell frequency in a contingency table is less than 5. Fisher's exact test calculates the probability of obtaining the observed distribution and all other possible distributions that are more extreme, assuming independence. The p-value represents the sum of these probabilities. A low p-value indicates a significant association between the variables.

- **Barnard's Exact Test**:
  (https://en.wikipedia.org/wiki/Barnard%27s_test)
  Barnard's exact test is similar to Fisher's exact test and is used for analyzing the association between categorical variables. It calculates the probability of observing a given table, assuming independence, and compares it to the probability of observing the observed table under a specific alternative hypothesis. The p-value is computed by summing the probabilities of tables that are at least as extreme as the observed table. Again, a low p-value indicates a significant association.

- **Boschloo's Exact Test**:
  (https://en.wikipedia.org/wiki/Boschloo%27s_test)
  Boschloo's exact test is an alternative to Fisher's exact test that can be used when there are ordered categories or ordinal data. It tests for the association between two ordinal variables. The test calculates the probability of observing the data under the null hypothesis of no association, and the p-value is derived accordingly. A low p-value suggests a significant association between the ordinal variables.

Boschloo's exact test is not considered in this analysis due to its use for ordinal categories.

The **operational difference between Barnard's exact test and Fisher's exact test is how they handle nuisance parameter(s) in calculating the p-value**. Fisher's exact test avoids estimating nuisance parameters by conditioning on both margins, while Barnard's test considers all legitimate possible values and chooses the value that maximizes the p-value.

The **theoretical difference is that Barnard's test uses the double-binomially distributed distribution, while Fisher's test uses the hypergeometric distribution for conditioning**.

Both tests are valid, bounding the type I error rate at the alpha level. However, Barnard's test can be more powerful than Fisher's test by considering more extreme tables and not conditioning on the second margin, which Fisher's test ignores.

It is considered that **Fisher's exact test** is **a more uniformly powerful alternative** to the **Chi-squared test**, while **Barnard's** and **Boschloo's exact tests** are **uniformly more powerful** alternatives to **Fischer's exact test**.

To analyze the statistics and p-values of these tests, usually **the obtained p-value is compared to a significance level (such as 0.05; 5%)**. If the **p-value is smaller** than the **significance level**, **the null hypothesis is rejected** and it is concluded that there is evidence of an association between the variables. Conversely, if the **p-value is larger** than the **significance level**, **the null hypothesis stands** and **it can be concluded that there is insufficient evidence to suggest an association**.

In statistics, the **Bonferroni correction** is **a method to counteract the multiple comparisons problem** (https://en.wikipedia.org/wiki/Bonferroni_correction).

Statistical hypothesis testing is based on rejecting the null hypothesis if the likelihood of the observed data under the null hypotheses is low.
If multiple hypotheses are tested, the probability of observing a rare event increases, and therefore, the likelihood of incorrectly rejecting a null hypothesis (i.e., making a Type I error) increases.

For these reasons, **p-value is corrected according to the Bonferroni correction**.

Number of 2x2 contingency tables that can be constructed without including same-named ones and inverse pairs (if we have A and B, B and A is not needed) involves selecting combinations of categories, we can calculate the number of unique combinations using the formula for combinations:

nCr = n! / (r!(n-r)!)

Where n is the total number of categories (**7 in this case**) and r is the number of categories to be selected (**2 for a 2x2 table**).

Applying this formula, we get:

nCr = 7! / (2!(7-2)!) = 7! / (2!5!) = (7x6) / (2x1) = **21**

Null Hypothesis: There are no significant differences between the protected and unprotected species.

Alternative Hypothesis: There are significant differences between the protected and unprotected species.

To analyze the given results, we need to **compare the p-values** with the predetermined **significance level of 0.00238**. The significance level, also known as **alpha (α)**, **represents the threshold below which we reject the null hypothesis**.

In [85]:
```
# a dictionary containing counts of protected (every conservation status except "Least concern") and not protected species for each category
protected_count = {}

# counting them and storing them into the dictionary
for category in set(biodiversity.category):
    category_conservation_status = biodiversity.conservation_status[biodiversity.category == category]
    total_count = len(category_conservation_status)
    not_protected = (category_conservation_status == "Least concern").sum()
    protected = total_count - not_protected
    protected_count[category] = {"not_protected": not_protected,
                                 "protected": protected,
                                 "total": total_count}
```

Table 6: Category Protection Breakdown: Counts, Totals, and Percentages

In [86]:
```
protected_data = pd.DataFrame(protected_count).T
protected_data["protected_percentage"] = protected_data["protected"] / protected_data["total"] * 100
protected_data = protected_data.sort_values("protected_percentage", ascending=False)
display(protected_data.style.set_caption("Category Protection Breakdown: Counts, Totals, and Percentages"))
```

Category Protection Breakdown: Counts, Totals, and Percentages

	not_protected	protected	total	protected_percentage
Mammal	584	120	704	17.045455
Bird	1652	300	1952	15.368852
Amphibian	288	28	316	8.860759
Fish	460	40	500	8.000000
Reptile	292	20	312	6.410256
Nonvascular Plant	1312	20	1332	1.501502
Vascular Plant	16864	184	17048	1.079305

Table 7: Chi-squared test and Fisher's and Barnard's exact test results

In [87]:
```
# iterate through "protected_data" and run "calculate_exact_tests()" for each 2x2 contingency table (not for AA; if there is AB, exclude BA)

# create empty lists to store categories under Null and Alternative hypotheses
null_hypothesis_results = []
alternative_hypothesis_results = []

iteration = 0

# calculate the total number of iterations
total_iterations = (len(protected_data) * (len(protected_data) - 1)) // 2

# set the desired p-value threshold; 5% can be considered as the universal standard
# Divided by "total_iterations" to include the Bonferroni correction
# Expected output is 21 calculations; Checks out!
p_value = 0.05 / total_iterations

# loops that ensure the right amount of combinations for the contingency tables
for i in range(len(protected_data)):
    not_protected_1 = protected_data.iloc[i]["not_protected"]
    protected_1 = protected_data.iloc[i]["protected"]
    category_1 = protected_data.index[i]

    for j in range(i+1, len(protected_data)):
        not_protected_2 = protected_data.iloc[j]["not_protected"]
        protected_2 = protected_data.iloc[j]["protected"]
        category_2 = protected_data.index[j]

        result = calculate_exact_tests([[not_protected_1, protected_1], [not_protected_2, protected_2]])

        # compare the p-value to the given threshold and store everything
        # force comparison with the most rigorous test (Barnards)
        if result["p-value"][2] < p_value:
            alternative_hypothesis_results.append({
                "Category 1": category_1,
                "Category 2": category_2,
                "Chi2 statistic": result["Statistic"][0],
                "Chi2 p-value": result["p-value"][0],
                "Fisher statistic": result["Statistic"][1],
                "Fisher p-value": result["p-value"][1],
                "Barnard statistic": result["Statistic"][2],
                "Barnard p-value": result["p-value"][2],
                "Hypothesis": "Alternative"
            })
        else:
            null_hypothesis_results.append({
                "Category 1": category_1,
                "Category 2": category_2,
                "Chi2 statistic": result["Statistic"][0],
                "Chi2 p-value": result["p-value"][0],
                "Fisher statistic": result["Statistic"][1],
                "Fisher p-value": result["p-value"][1],
                "Barnard statistic": result["Statistic"][2],
                "Barnard p-value": result["p-value"][2],
                "Hypothesis": "Null"
            })

        iteration += 1
        print(f"Iteration {iteration}/{total_iterations} completed!")

# create DataFrames to categorize the results
null_hypothesis_df = pd.DataFrame(null_hypothesis_results)
alternative_hypothesis_results = pd.DataFrame(alternative_hypothesis_results)

# merge null and alternative hypothesis DataFrames
exact_tests_results = pd.concat([null_hypothesis_df, alternative_hypothesis_results], ignore_index=True)

print("Calculations successfully completed!")
```

```
Iteration 1/21 completed!
Iteration 2/21 completed!
Iteration 3/21 completed!
Iteration 4/21 completed!
Iteration 5/21 completed!
Iteration 6/21 completed!
Iteration 7/21 completed!
Iteration 8/21 completed!
Iteration 9/21 completed!
Iteration 10/21 completed!
Iteration 11/21 completed!
Iteration 12/21 completed!
Iteration 13/21 completed!
Iteration 14/21 completed!
Iteration 15/21 completed!
Iteration 16/21 completed!
Iteration 17/21 completed!
Iteration 18/21 completed!
Iteration 19/21 completed!
Iteration 20/21 completed!
Iteration 21/21 completed!
Calculations successfully completed!
```

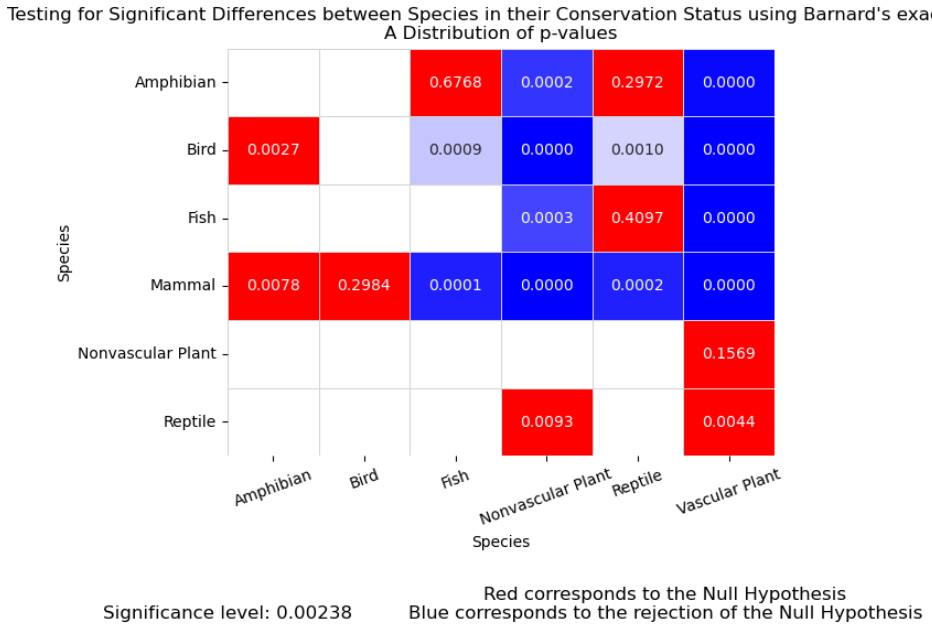
In [88]:
```
display(exact_tests_results.style.set_caption(f"Chi-squared test and Fisher's and Barnard's exact test results (Significance level: {round(p_value, 6)})"))
```

Chi-squared test and Fisher's and Barnard's exact test results (Significance level: 0.002381)

	Category 1	Category 2	Chi2 statistic	Chi2 p-value	Fisher statistic	Fisher p-value	Barnard statistic	Barnard p-value	Hypothesis
0	Mammal	Bird	0.970184	0.324634	0.883777	0.305849	-1.045225	0.298403	Null
1	Mammal	Amphibian	11.127833	0.000850	0.473148	0.000505	-3.431969	0.007760	Null
2	Bird	Amphibian	8.793312	0.003023	0.535370	0.001834	-3.051553	0.002696	Null
3	Amphibian	Fish	0.092023	0.761621	0.894410	0.697297	-0.433361	0.676771	Null
4	Amphibian	Reptile	1.010914	0.314684	0.704501	0.293558	-1.155637	0.297246	Null
5	Fish	Reptile	0.496226	0.481163	0.787671	0.490707	-0.842331	0.409732	Null
6	Reptile	Nonvascular Plant	23.632806	0.000001	0.222561	0.000007	-5.065467	0.009251	Null
7	Reptile	Vascular Plant	70.459002	0.000000	0.159298	0.000000	-8.659054	0.004388	Null
8	Nonvascular Plant	Vascular Plant	1.640202	0.200298	0.715750	0.172131	-1.416483	0.156892	Null
9	Mammal	Fish	19.981578	0.000008	0.423188	0.000004	-4.556220	0.000146	Alternative
10	Mammal	Reptile	19.696214	0.000009	0.333333	0.000003	-4.536699	0.000152	Alternative
11	Mammal	Nonvascular Plant	171.365605	0.000000	0.074187	0.000000	-13.182738	0.000000	Alternative
12	Mammal	Vascular Plant	1014.476132	0.000000	0.053099	0.000000	-31.999062	0.000000	Alternative
13	Bird	Fish	17.484891	0.000029	0.478841	0.000009	-4.254011	0.000929	Alternative
14	Bird	Reptile	17.058216	0.000036	0.377169	0.000007	-4.217667	0.000993	Alternative
15	Bird	Nonvascular Plant	171.546322	0.000000	0.083943	0.000000	-13.157489	0.000000	Alternative
16	Bird	Vascular Plant	1434.875551	0.000000	0.060082	0.000000	-37.955579	0.000000	Alternative
17	Amphibian	Nonvascular Plant	46.348633	0.000000	0.156794	0.000000	-6.994033	0.000248	Alternative
18	Amphibian	Vascular Plant	149.383640	0.000000	0.112226	0.000000	-12.480747	0.000041	Alternative
19	Fish	Nonvascular Plant	46.433369	0.000000	0.175305	0.000000	-6.961541	0.000320	Alternative
20	Fish	Vascular Plant	179.167196	0.000000	0.125474	0.000000	-13.587424	0.000000	Alternative

Figure 14: Testing for Significant Differences between Species in their Conservation Status using Barnard's exact test: A Distribution of p-values

In [89]:
```
# select data for a heatmap
heatmap_data = exact_tests_results[["Category 1", "Category 2", "Barnard p-value"]].copy()

# pivot the data to create the heatmap_data DataFrame
heatmap_data = heatmap_data.pivot(index="Category 1", columns="Category 2", values="Barnard p-value")

# create the heatmap
heatmap = sns.heatmap(heatmap_data, annot=True, fmt=".4f", cmap="bwr", cbar=False, vmin=0, vmax=p_value, linewidths=0.5, linecolor="lightgray")

# add annotation below the graph for the hypotheses
plt.annotate("Red corresponds to the Null Hypothesis\nBlue corresponds to the rejection of the Null Hypothesis",
             xy=(0.8, -0.4),
             xycoords="axes fraction",
             ha="center",
             fontsize=12)

# add annotation below the graph for significance level
plt.annotate("Significance level: 0.00238",
             xy=(0, -0.4),
             xycoords="axes fraction",
             ha="center",
             fontsize=12)

# set the title, x-label, and y-label
plt.title("Testing for Significant Differences between Species in their Conservation Status using Barnard's exact test:\nA Distribution of p-values")
plt.xlabel("Species")
plt.ylabel("Species")

# rotate the x-axis tick labels
heatmap.set_xticklabels(heatmap.get_xticklabels(), rotation=20)

# save the figure
plt.savefig("Figure14.png", dpi=300, bbox_inches="tight")

# display the figure
plt.show()
```



Conclusions

The **analysis of biodiversity** has been **conducted in four USA national parks**:

> 1. **Yellowstone National Park**
> 2. **Yosemite National Park**
> 3. **Great Smoky Mountains National Park**
> 4. **Bryce National Park**

Although the **surface areas of the areas in question vary significantly** (**Surface Areas Comparison**), there is a **noticeable pattern in the number and distribution of observed species**, as depicted in **Figure 1**. These findings reinforce the suspicions that emerged during the data cleaning process, indicating that **this dataset is fictional**. **Additional evidence** can be found in **Table 2**. It is highly improbable for four national parks with distinct surface areas and geographical locations to exhibit such a perfect distribution of species, both collectively and individually, in terms of their conservation status. While such patterns may be observed in places like zoos, even there, they are unlikely to occur. Nonetheless, we can still conduct analysis and gain insights from this dataset, which **comprises information on 5541 unique species across seven categories**:

> 1. **Mammal**
> 2. **Bird**
> 3. **Reptile**
> 4. **Amphibian**
> 5. **Fish**
> 6. **Vascular Plant**
> 7. **Nonvascular Plant**

The **most observed species in each park is a Vascular Plant called Sedge** (https://en.wikipedia.org/wiki/Cyperaceae). Five most observed species for every national park can be seen in **Figure 1**.

Figure 2 displays the **top five most observed species for each of the seven categories**, with the count of observations displayed atop each bar. **Observations serve as one of the primary analysis factors in this dataset**. **Vascular plants hold an overwhelming lead with 77% of the total observations**, as depicted in **Figure 3**. This distribution aligns with expectations, as plants exhibit the widest range, remain stationary, are easily identifiable, and possess a high species abundance.

Even with the perfect distribution of species within parks, a touch of realism emerges when examining the total observations per park, as evidenced by the data provided (**Figure 4**). **Yellowstone National Park stands out with 44% of all observations, coinciding with its status as the largest park in terms of surface area**. However, there is a notable discrepancy between Bryce and Great Smoky Mountains, which cannot be accounted for due to the lack of information.

The **count and relative proportions of observations for each category in each national park** can be observed in **Figure 5**. The **distribution of observations follows an established pattern**, with **Vascular Plants** and **Yellowstone National Park** having the **highest number of observations**. Due to the significant difference in total observations per category, a **logarithmic scale is applied to enhance the readability of proportions between categories**. To gain insights into the distribution of species within each category, one can refer to **Figure 6**, and consult **Table 1** for a more comprehensive analysis.

The dataset also comprises **five distinct conservation status groups** for each species, which serve **to indicate the degree to which a species is threatened or protected**:

> 1. **Species of Concern**
> 2. **Endangered**
> 3. **Threatened**
> 4. **In Recovery**
> 5. **Least Concern**

Figure 7 provides an **overview of the conservation status distribution**, revealing that the **"Least Concern" category encompasses more than 96% of the total species within the dataset**. For a more **visual type understanding of conservation status distributions**, **Figure 8** presents a **detailed breakdown for each category**, employing a logarithmic scale yet again to ensure clarity and ease of interpretation. For a **comprehensive numerical analysis**, refer to **Table 3**, which contains the same information regarding conservation status distributions. Additionally, **Figure 13** provides **a heatmap visualization corresponding to the data presented in **Table 3****. **Table 4** presents tangible results in the form of **percentages, offering another clear representation of the data**. Species of concern, endangered species, and species that are threatened are then analyzed in more detail.

Figures 9, 10 and 11 present a summary of the conservation statuses for **endangered, species of concern, and threatened species**, respectively. Among the realm of **endangered species**, **mammals occupy a prominent position as one of the most vulnerable groups**. A significant proportion of **species of concern** belongs to the **avian class**. However, it is the **fish** species that predominate as the **majority of the threatened species**. In addition to each figure, there is a table highlighting the **top ten species observed for their respective conservation status**. Figure 12 displays the **total number of observations for each conservation status**, and it aligns with the previous conclusions. As most species fall under the category of "Least Concerned", they are also the most observed ones. **This pattern continues in a predictable way and in conjunction with previous data**.

A **diversity index** is a quantitative measure that reflects how many different types (such as species) there are in a dataset (a community), and that can simultaneously take into account the phylogenetic relations among the individuals distributed among those types, such as **richness**, **divergence** or **evenness**. **These indices are statistical representations of biodiversity in different aspects**.

Table 5 presents the results for the **Shannon-Weaver and Simpson's indices** calculated for each national park. The findings indicate that **all of the parks demonstrate substantial species richness and evenness in terms of the observed species**. Additionally, there is **a relatively low dominance of any particular species, indicating a balanced distribution of individuals among different species within the parks**. These results collectively signify **a high level of diversity among the species present in all four parks**. Thus, it can be inferred that these **parks exhibit notable biodiversity, characterized by a wide variety of species and a relatively equitable distribution of individuals among them**.

There are several ways to determine associations between two categorical variables. In order to assess **the relationship between protected and unproteced species categories**, **Chi-squared test**, along with **Fisher's, Barnard's, and Boschlo's exact tests**, have been considered. The rationales for considering each test are discussed in section 5.4.2.

To ensure **accurate assessment of the results**, the **significance level**, also known as **alpha (α)**, **needs to be corrected**. Therefore, **the p-value is adjusted using the Bonferroni correction** (5.4.2). Subsequently, **Python and its modules** are utilized to perform the aforementioned tests on several **2x2 contingency tables** derived from Table 6. **Boschloo's exact test is not considered in this analysis due to its use for ordinal categories**.

Table 7 contains the **results for each category pair, including the corresponding statistic and p-value for each test**. Additionally, there is a "**Hypothesis**" column indicating whether the **Null Hypothesis is accepted or rejected**. In case of rejection, it states "Alternative". The **significance level was compared to the most rigorous test**, which is the **Barnard's exact test**. To provide a more **visual representation of these numbers**, Figure 14 illustrates a heatmap showcasing all sensible combinations of protected and unproteced species categories.

What can be gathered from these results is that **mammals and birds** are the **most endangered categories** because they account for the highest percentage of all protected species, followed by **fish and amphibians**. Some protected and unprotected categories show **statistically significant differences**, while others do not.

In order to gain **a more comprehensive understanding of the results** obtained from the tests conducted on significant differences between species in their conservation status using Barnard's exact test, **it is essential to obtain additional data and perform more thorough analysis**. **The distribution of p-values obtained thus far indicates the presence of some potential distinctions**, but to draw reliable conclusions and make informed decisions, further data collection and refined statistical examination are imperative. By incorporating a broader range of data and implementing robust analytical techniques, we can obtain a deeper comprehension of the variations in conservation status among species.

In order to **safeguard biodiversity**, it is **imperative to allocate future resources towards the conservation and protection of endangered species**. Among these, special emphasis should be placed on the **Gray Wolf species**, which faces significant threats to its survival.

Furthermore, it is essential to investigate the factors contributing to the substantial number of bird species falling under the "Species of Concern" conservation status, as well as the fish species classified as "Threatened". Several **potential risk factors** may be influencing these conservation statuses, including:

> - **Inadequate waste management**
> - **Detrimental impact of wildfires**
> - **Unsustainable hunting practices**
> - **Rapid pace of modernization**

To address these issues effectively, a comprehensive approach is required. **Adequate funding should be allocated towards habitat preservation, restoration, and management, with a focus on protecting critical ecosystems for these species**. Collaborative efforts involving governmental organizations, conservation agencies, and local communities must be encouraged to develop and implement conservation strategies.

Moreover, **public awareness campaigns should be initiated to educate communities** about the importance of conserving these species and the ecosystems they inhabit. Promoting sustainable practices, such as responsible waste management and wildlife-friendly land use, can help mitigate the threats faced by these species.

Additionally, **robust legislation and enforcement mechanisms** should be put in place to **deter illegal activities**, including poaching and the trade of endangered species. Strengthening existing laws and regulations will ensure that the protection of these species remains a top priority.

In conclusion, directing future resources towards the preservation of endangered species, particularly the Gray Wolf, is of paramount importance. Simultaneously, a comprehensive assessment of the factors contributing to the increased number of bird species under the "Species of Concern" category and fish species in the "Threatened" conservation status is crucial. **By addressing potential risk factors, implementing conservation measures, and raising public awareness, we can work towards safeguarding these species and their habitats for the benefit of present and future generations**.