myproject

July 15, 2024

1 Final Deep Learning Project (MohammadReza Shabani)

1.0.1 1) Choosing your dataset

I choose Animal Information Dataset

1.0.2 2) Preprocessing and Feature Engineering

[12]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

df = pd.read_csv('/content/Animal Dataset.csv')
df.shape # getting the count of rows and columns in the dataframe
```

[12]:
```
(205, 16)
```

[13]:
```
# checking the data
df.head(10).style.set_properties()
```

[13]:
```
<pandas.io.formats.style.Styler at 0x78d390c9bfa0>
```

[ ]:
```
pd.DataFrame(df.columns,columns=['column name']) # list of columns
```

[ ]:
```
                column name
0                    Animal
1               Height (cm)
2               Weight (kg)
3                     Color
4          Lifespan (years)
5                      Diet
6                   Habitat
7                 Predators
8       Average Speed (km/h)
9           Countries Found
10      Conservation Status
11                   Family
12   Gestation Period (days)
13          Top Speed (km/h)
```

```
14         Social Structure
15      Offspring per Birth
```

1.1 Data cleaning

As we can see in the data numerci columns have '-' in between values we can replace such values with mean of the left and right values of the '-'. First we will reomve all upto values in columns where it is present

```
[14]: columns_with_upto=['Height (cm)','Weight (kg)','Lifespan (years)','Offspring␣
      ↪per Birth']
      for x in columns_with_upto:
          regex='Up.*'
          df[x][df[x].str.match(regex)]=df[x][df[x].str.match(regex)].str.split('␣
      ↪',expand=True)[2]
```

We need to change the datatype of numeric columns as every column type is string datatype.

```
[15]: for x in ['Height (cm)','Weight (kg)','Lifespan (years)','Average Speed (km/
      ↪h)','Gestation Period (days)','Offspring per Birth','Top Speed (km/h)']:
          ab=df[x][df[x].str.contains('-')]
          df[x][df[x].str.contains('-')]=ab.str.split('-').str.get(-1)
```

```
[16]: df['Offspring per Birth'][df['Offspring per Birth'].str.endswith('(usually)')]=3
```

We single out the columns that have multiple data

```
[17]: for x in ['Countries Found','Habitat','Predators']:
          # Split the 'Countries Found' column by commas into separate columns
          split_columns = df[x].str.split(', ', expand=True)

          # Rename the new columns with meaningful names (e.g., Country 1, Country 2)
          split_columns.columns = [f'{x} {i+1}' for i in range(split_columns.
      ↪shape[1])]

          # Concatenate the split columns with the original DataFrame
          df = pd.concat([df, split_columns], axis=1)

          # Drop the original 'Countries Found' column
          df.drop(columns=[x], inplace=True)
```

```
[ ]: df
```

```
[ ]:               Animal Height (cm) Weight (kg)           Color  \
     0           Aardvark         130          65            Grey
     1           Aardwolf          50          14    Yellow-brown
     2   African Elephant         310        6000            Grey
     3       African Lion         110         250             Tan
```

```
4       African Wild Dog          80          36      Multicolored
..                   …         …         …                …
200                  Yak         160        1200      Brown, Black
201  Yellow-Eyed Penguin          65           3     Yellow, White
202            Yeti Crab          15         0.5      White, Hairy
203                Zebra         340         900      Black, White
204          Zebra Shark         330          32  Brown, Yellowish

    Lifespan (years)         Diet Average Speed (km/h) Conservation Status  \
0                 30  Insectivore                   40       Least Concern
1                 12  Insectivore                   30       Least Concern
2                 70    Herbivore                   25          Vulnerable
3                 14    Carnivore                   58          Vulnerable
4                 12    Carnivore                   56          Endangered
..               …          …                  …                 …
200               25    Herbivore                   24       Least Concern
201               20    Carnivore                   25          Endangered
202               20     Omnivore       Not Applicable       Not Evaluated
203               25    Herbivore                   25       Least Concern
204               30    Carnivore                   20          Endangered

              Family Gestation Period (days)  … Offspring per Birth  \
0    Orycteropodidae                     240  …                   1
1          Hyaenidae                      90  …                   5
2       Elephantidae                     660  …                   1
3            Felidae                     105  …                   3
4            Canidae                      70  …                  12
..               …                    …   …                 …
200          Bovidae                     280  …                  50
201     Spheniscidae                      90  …                   1
202         Kiwaidae          Not Applicable  …      Not Applicable
203          Equidae                     365  …                  20
204  Stegostomatidae                      25  …                  25

              Countries Found 1 Countries Found 2 Countries Found 3  \
0                        Africa              None              None
1    Eastern and Southern Africa             None              None
2                        Africa              None              None
3                        Africa              None              None
4             Sub-Saharan Africa             None              None
..                          …               …               …
200                   Himalayas      Central Asia              None
201                 New Zealand              None              None
202               Pacific Ocean              None              None
203                      Africa              None              None
204          Indo-Pacific region             None              None
```

```
    Countries Found 4           Habitat 1   Habitat 2 Habitat 3  \
0                None            Savannas  Grasslands      None
1                None          Grasslands    Savannas      None
2                None            Savannah      Forest      None
3                None          Grasslands    Savannas      None
4                None           Savannahs        None      None
..                …                   …         …       …
200              None           Mountains        None      None
201              None       Coastal Areas        None      None
202              None  Hydrothermal Vents        None      None
203              None          Grasslands        None      None
204              None         Coral Reefs        None      None

        Predators 1 Predators 2
0             Lions      Hyenas
1             Lions    Leopards
2             Lions      Hyenas
3            Hyenas  Crocodiles
4             Lions      Hyenas
..              …           …
200   Snow Leopards      Wolves
201           Seals       Orcas
202  Not Applicable        None
203           Lions      Hyenas
204     Larger Fish        None

[205 rows x 22 columns]
```

[18]:
```
df['Countries Found 4'] = df['Countries Found 4'].replace('None',np.nan)
df['Countries Found 3'] = df['Countries Found 3'].replace('None',np.nan)
df['Countries Found 2'] = df['Countries Found 2'].replace('None',np.nan)
```

[19]:
```
print(df['Countries Found 4'].isna().sum(),df['Countries Found 3'].isna().
 ↪sum(),df['Countries Found 2'].isna().sum())
```

```
203 187 150
```

As we can see most of the countries columns - 'Country found 2','Country found 3,'Country found 4' have too many missing values thus dropping them.

[20]:
```
df.drop(columns=['Countries Found 2','Countries Found 3','Countries Found 4'],␣
 ↪inplace=True)
df = df.rename(columns={'Countries Found 1': 'Countries Found'})
```

[21]:
```
df['Habitat 1'] = df['Habitat 1'].replace('None',np.nan)
df['Habitat 2'] = df['Habitat 2'].replace('None',np.nan)

df['Predators 1'] = df['Predators 1'].replace('None',np.nan)
```

```python
df['Predators 2'] = df['Predators 2'].replace('None',np.nan)
```

[ ]:
```python
df
```

[ ]:
```
                 Animal Height (cm) Weight (kg)              Color  \
0              Aardvark         130          65               Grey
1              Aardwolf          50          14       Yellow-brown
2      African Elephant         310        6000               Grey
3          African Lion         110         250                Tan
4      African Wild Dog          80          36       Multicolored
..                  …           …           …                  …
200                 Yak         160        1200       Brown, Black
201  Yellow-Eyed Penguin         65           3      Yellow, White
202           Yeti Crab          15         0.5       White, Hairy
203               Zebra         340         900       Black, White
204         Zebra Shark         330          32  Brown, Yellowish

    Lifespan (years)         Diet Average Speed (km/h) Conservation Status  \
0                 30  Insectivore                  40       Least Concern
1                 12  Insectivore                  30       Least Concern
2                 70    Herbivore                  25          Vulnerable
3                 14    Carnivore                  58          Vulnerable
4                 12    Carnivore                  56          Endangered
..               …          …                  …                 …
200               25    Herbivore                  24       Least Concern
201               20    Carnivore                  25          Endangered
202               20     Omnivore      Not Applicable       Not Evaluated
203               25    Herbivore                  25       Least Concern
204               30    Carnivore                  20          Endangered

               Family Gestation Period (days) Top Speed (km/h)  \
0     Orycteropodidae                     240               40
1           Hyaenidae                      90               40
2        Elephantidae                     660               40
3             Felidae                     105               80
4             Canidae                      70               56
..                …                     …              …
200           Bovidae                     280               24
201       Spheniscidae                     90               25
202          Kiwaidae          Not Applicable   Not Applicable
203           Equidae                     365               25
204   Stegostomatidae                      25               20

    Social Structure Offspring per Birth              Countries Found  \
0           Solitary                   1                       Africa
1           Solitary                   5  Eastern and Southern Africa
2         Herd-based                   1                       Africa
```

```
3        Group-based                   3                     Africa
4        Group-based                  12         Sub-Saharan Africa
..               …                …                       …
200      Group-based                  50                  Himalayas
201         Solitary                   1                New Zealand
202         Solitary      Not Applicable              Pacific Ocean
203      Group-based                  20                     Africa
204         Solitary                  25        Indo-Pacific region

              Habitat 1    Habitat 2 Habitat 3      Predators 1 Predators 2
0              Savannas  Grasslands      None            Lions      Hyenas
1            Grasslands    Savannas      None            Lions    Leopards
2              Savannah      Forest      None            Lions      Hyenas
3            Grasslands    Savannas      None           Hyenas  Crocodiles
4             Savannahs        None      None            Lions      Hyenas
..                  …           …       …              …         …
200           Mountains        None      None   Snow Leopards      Wolves
201       Coastal Areas        None      None            Seals       Orcas
202  Hydrothermal Vents        None      None  Not Applicable        None
203          Grasslands        None      None            Lions      Hyenas
204         Coral Reefs        None      None     Larger Fish        None

[205 rows x 19 columns]
```

[22]:
```
print(df['Habitat 1'].isna().sum(),df['Habitat 2'].isna().sum(),df['Habitat 3'].
 ↪isna().sum())
print(df['Predators 1'].isna().sum(),df['Predators 2'].isna().sum())
```

```
0 121 204
0 39
```

filling the same values in Habitat 2 from Habitat 1 where there is None.

[23]:
```
df['Habitat 2'] = df['Habitat 2'].fillna(df['Habitat 1'])
```

Dropping Habitat 3 as it has lots of missing values which makes the column not usable

[24]:
```
df.drop(columns=['Habitat 3'], inplace=True)
```

Filling the same values for predator 2 from predator 1 where there is no value

[25]:
```
df['Predators 2'] = df['Predators 2'].fillna(df['Predators 1'])
```

[26]:
```
df.replace('Not Applicable',np.nan,inplace=True)
```

it is better to drop the data with missing values as replacing them with mean mode median values would not give accurate data for that animal.

[27]:
```
df.dropna(inplace=True)
```

[28]:
```
df[df['Height (cm)'].isna()]
```

[28]:
```
Empty DataFrame
Columns: [Animal, Height (cm), Weight (kg), Color, Lifespan (years), Diet,
Average Speed (km/h), Conservation Status, Family, Gestation Period (days), Top
Speed (km/h), Social Structure, Offspring per Birth, Countries Found, Habitat 1,
Habitat 2, Predators 1, Predators 2]
Index: []
```

[29]:
```
df.dropna(subset=['Height (cm)'],inplace=True)
df.dropna(inplace=True)
```

fixing special character values for each columns

[30]:
```
df.drop(df[df['Height (cm)'] == ''].index,inplace= True)
df.replace(',', '', regex=True,inplace=True)
df.replace('Varies',np.nan,inplace=True)
df.dropna(inplace=True)
df['Lifespan (years)']=df['Lifespan (years)'].str.replace(r'\+', '', regex=True)
df['Average Speed (km/h)'][df['Average Speed (km/h)'].str.contains(r'.*water.
↪*', regex=True)]=df['Average Speed (km/h)'][df['Average Speed (km/h)'].str.
↪contains(r'.*water.*', regex=True)].str.split(" ",expand=True)[0]
df.replace(r'\(in burrow\)','',regex=True,inplace=True)
df['Gestation Period (days)'][df['Gestation Period (days)'].str.contains(r'.
↪*days.*', regex=True)]='10'
df['Gestation Period (days)'][df['Gestation Period (days)'].str.contains(r'.
↪*weeks.*', regex=True)]
df['Gestation Period (days)'][df['Gestation Period (days)'].str.contains(r'.
↪*weeks.*', regex=True)]=f'{9 * 7}'
df['Gestation Period (days)'][df['Gestation Period (days)'].str.contains(r'.
↪*months.*', regex=True)]
```

[30]:
```
23     42 months
32     12 months
37     12 months
54     14 months
81     18 months
103     8 months
121    16 months
Name: Gestation Period (days), dtype: object
```

[31]:
```
df['Gestation Period (days)'] = df['Gestation Period (days)'].apply(
    lambda x: f'{int(x.split()[0]) * 30}' if 'months' in x else x
)
df['Offspring per Birth']=df['Offspring per Birth'].str.replace(r')',"")
df['Offspring per Birth']=df['Offspring per Birth'].str.
↪replace(r'Hundreds',"100")
```

[33]:
```
# Initialize an empty list to store the non-convertible values
non_convertible_values = []

# Iterate through the column and attempt to convert to float
for value in df['Offspring per Birth']:
    try:
        float(value)
    except ValueError:
        non_convertible_values.append(value)

# Create a DataFrame containing the non-convertible values
df['Offspring per Birth'][df['Offspring per Birth'].
 ↪isin(non_convertible_values)] = df['Offspring per Birth'][df['Offspring per␣
 ↪Birth'].isin(non_convertible_values)].str.split(" ",expand=True)[0]
```

To correct the data types of different columns

[34]:
```
float_columns = ['Height (cm)','Weight (kg)','Lifespan (years)','Average Speed␣
 ↪(km/h)','Gestation Period (days)','Top Speed (km/h)','Offspring per Birth']
for x in float_columns:
    df[x] = df[x].str.replace('[a-zA-Z]', '', regex=True)
    df[x] = df[x].astype(float)
```

add new column

[35]:
```
df['Height-to-Weight Ratio'] = df['Height (cm)'] / df['Weight (kg)']
```

1.1.1 3) Exploratory Data Analysis

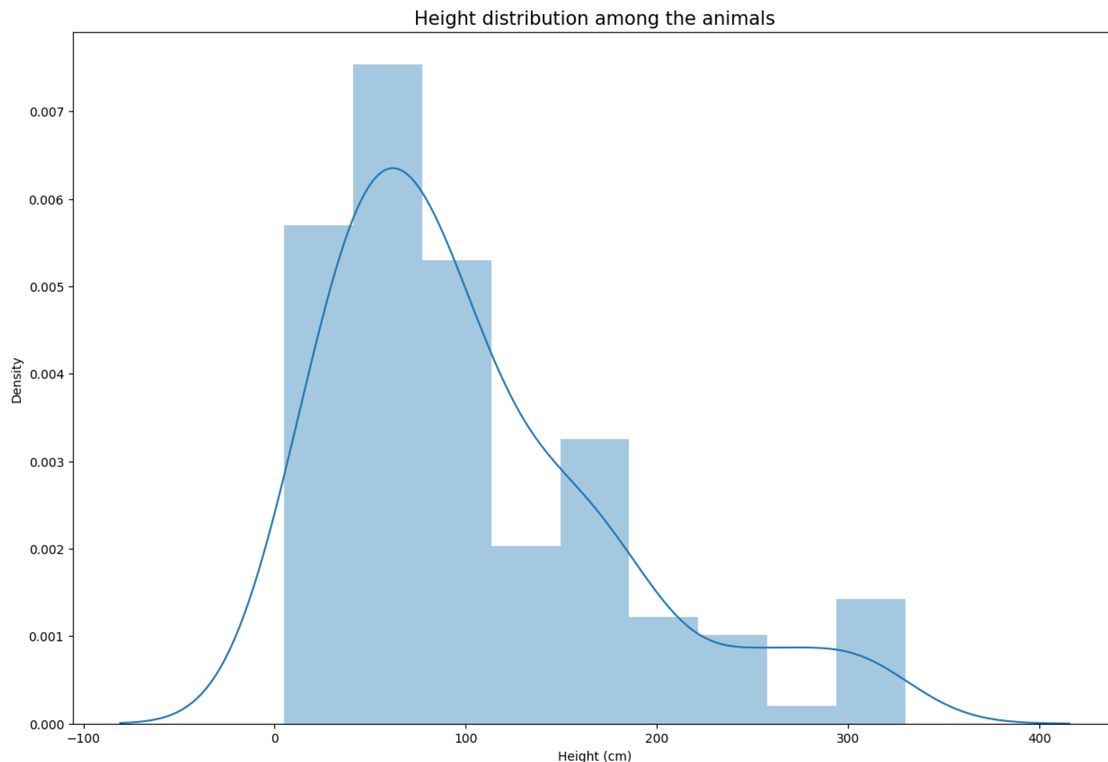
[36]:
```
pd.DataFrame(df.describe()).style\
    .set_properties(**{'background-color': 'lightyellow',
                      'color': 'black'})
```

[36]:
```
<pandas.io.formats.style.Styler at 0x78d390c9b4c0>
```

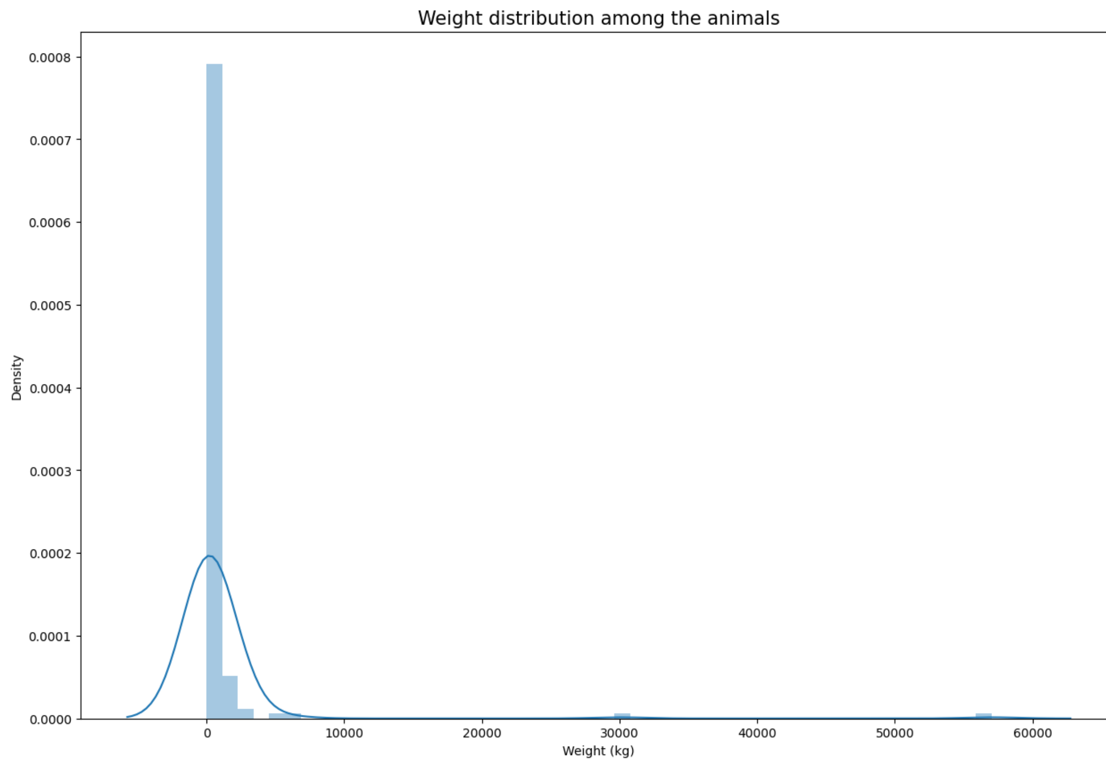
[37]:
```
print("The average height for the animals are {:.1f}cm, 99% of animals have top␣
 ↪speed less than {:.2f}km/hr or less, while the maximum weight for an animal␣
 ↪is {}kg.".format(df['Height (cm)'].mean(), df['Top Speed (km/h)'].quantile(0.
 ↪99), df['Weight (kg)'].max()))
print('99 % of animals have height less than {:.1f}cm'.format(df['Height (cm)'].
 ↪quantile(0.99)))
```

```
The average height for the animals are 197.2cm, 99% of animals have top speed
less than 103.33km/hr or less, while the maximum weight for an animal is
57000.0kg.
99 % of animals have height less than 2175.6cm
```

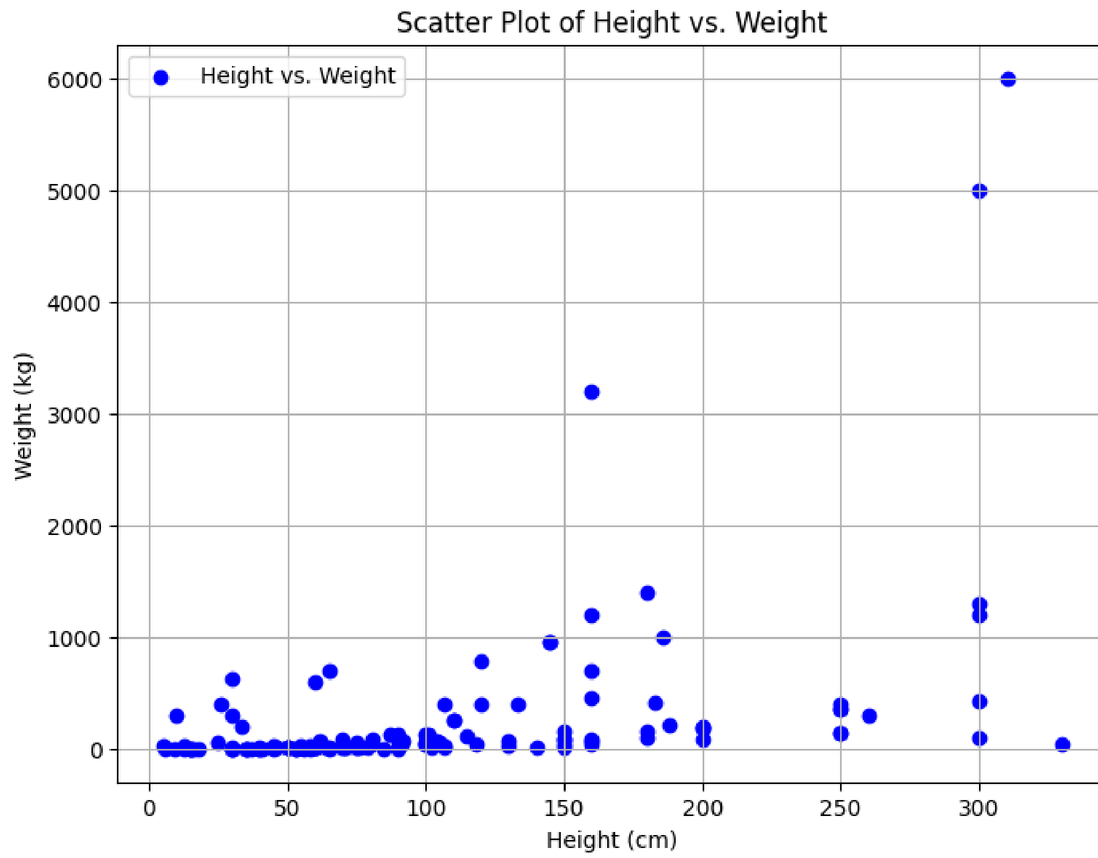
[38]:
```
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

plt.figure(figsize=(15,10))
data = df.copy()
data=data[data['Height (cm)'] < data['Height (cm)'].quantile(0.90)]
sns.distplot(data['Height (cm)'].sort_values())
plt.title("Height distribution among the animals",fontsize=15)
plt.show()
```



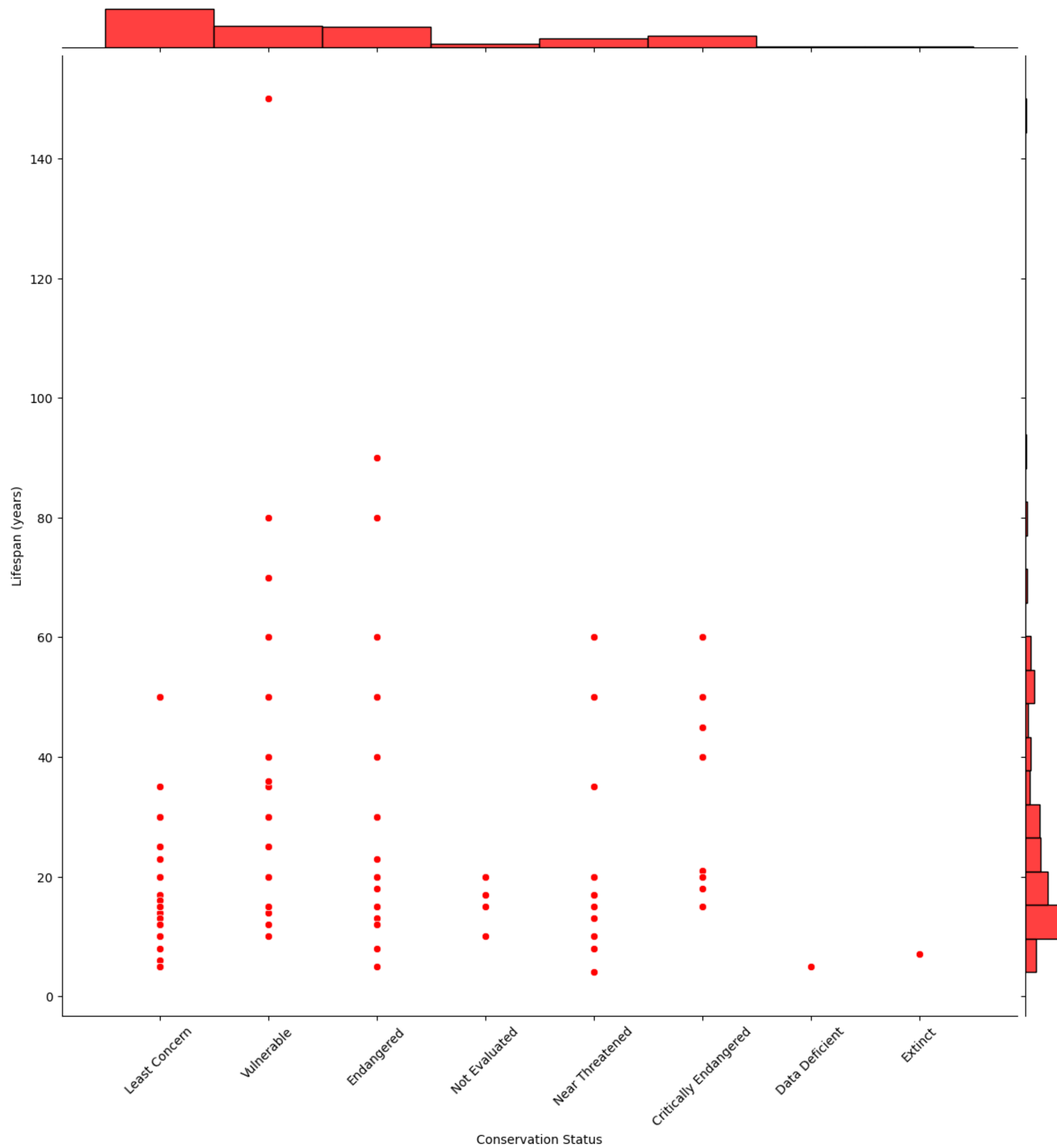
[ ]:
```
plt.figure(figsize=(15,10))
data2=df.copy()
data2=data2[data2['Weight (kg)'] < data2['Weight (kg)'].quantile(0.90)]
sns.distplot(df['Weight (kg)'].sort_values())
plt.title("Weight distribution among the animals",fontsize=15)
plt.show()
```



[ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(data['Height (cm)'], data['Weight (kg)'], c='blue', marker='o',␣
↪label='Height vs. Weight')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.title('Scatter Plot of Height vs. Weight')
plt.legend()
plt.grid(True)
plt.show()
```



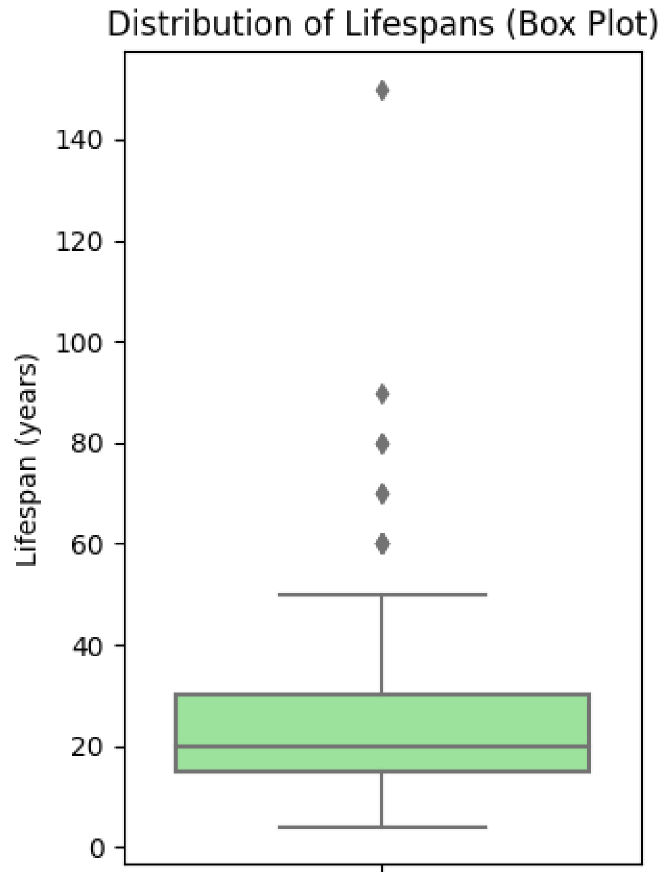
[ ]:
```
sns.jointplot(x="Conservation Status", y="Lifespan (years)", data=df,␣
  ↪height=12, ratio=20, color="r")
plt.xticks(rotation=45)
plt.show()
```



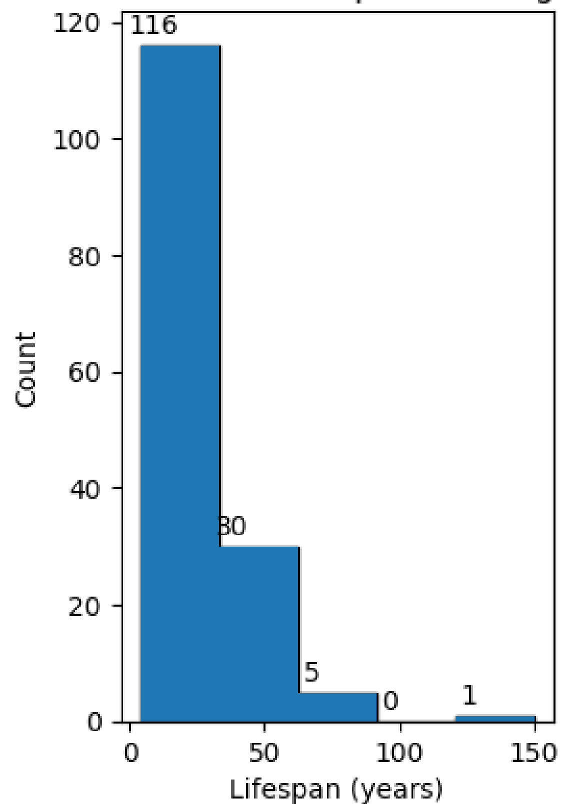
[ ]:
```
plt.figure(figsize=(16, 10))
df['Color'].value_counts().plot(kind='bar', color='skyblue')
plt.xlabel('Color')
plt.ylabel('Count')
plt.title('Distribution of Colors in Animals')
plt.xticks(rotation=40)
plt.show()
```

[ ]:

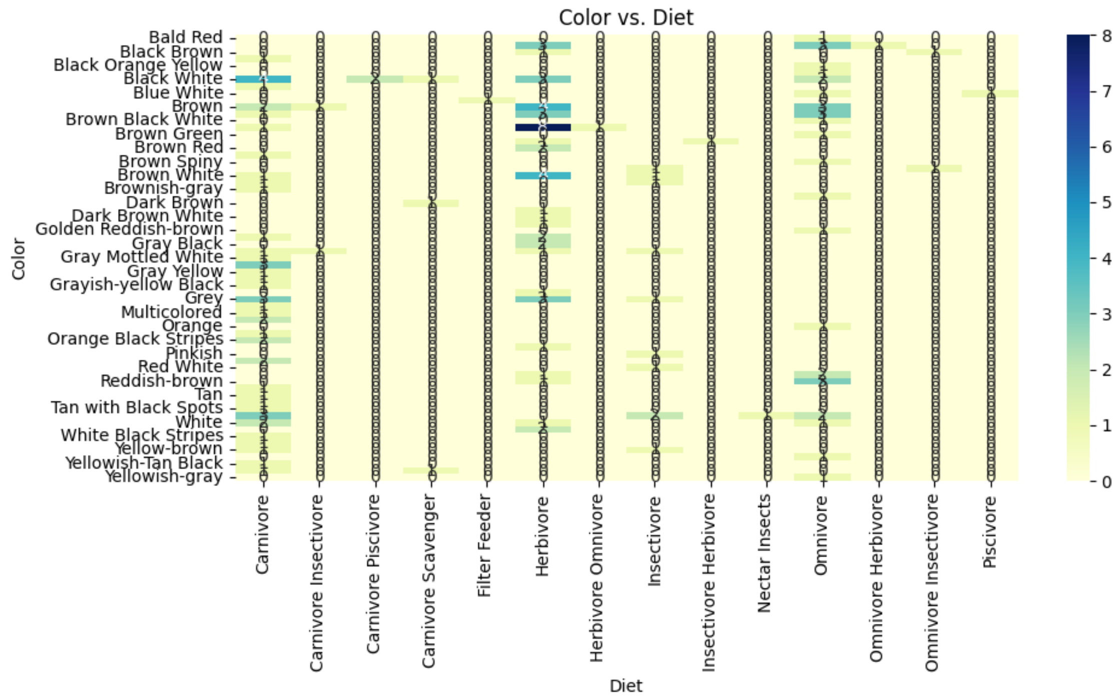
```
plt.subplot(1, 2, 2)
sns.boxplot(data=df, y='Lifespan (years)', color='lightgreen')
plt.ylabel('Lifespan (years)')
plt.title('Distribution of Lifespans (Box Plot)')

# Adjust the layout
plt.tight_layout()
plt.show()
```



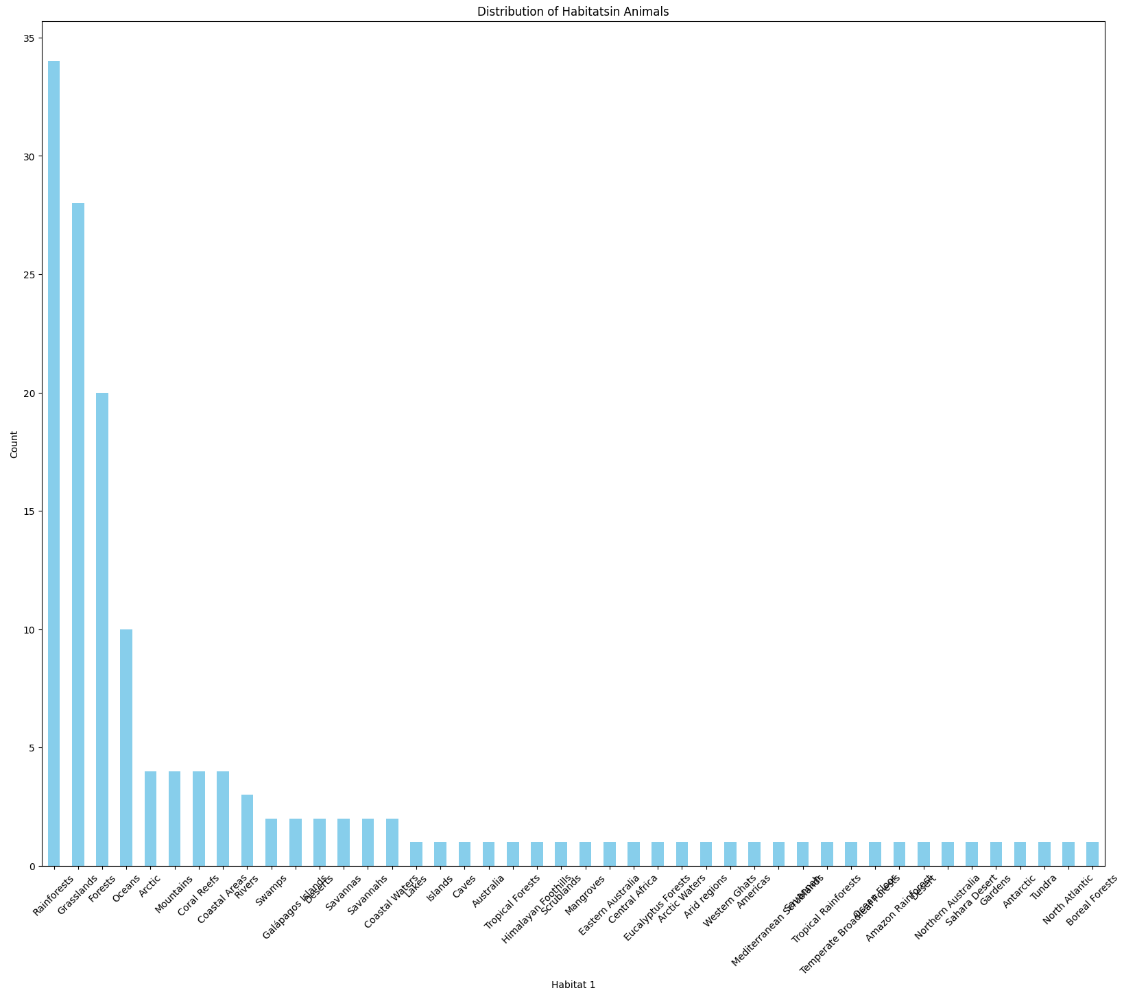

```
[ ]: plt.subplot(1, 2, 2)
     plt.hist(df['Lifespan (years)'], bins=5, color='skyblue', edgecolor='black')
     plt.xlabel('Lifespan (years)')
     plt.ylabel('Count')
     plt.title('Distribution of Lifespans (Histogram)')
     # Get the histogram counts and bin edges
     counts, bin_edges, _ = plt.hist(df['Lifespan (years)'], bins=5)

     # Annotate each bar with the count
     for count, x in zip(counts, bin_edges):
         plt.text(x + 5, count + 1, str(int(count)), ha='center', va='bottom')
```



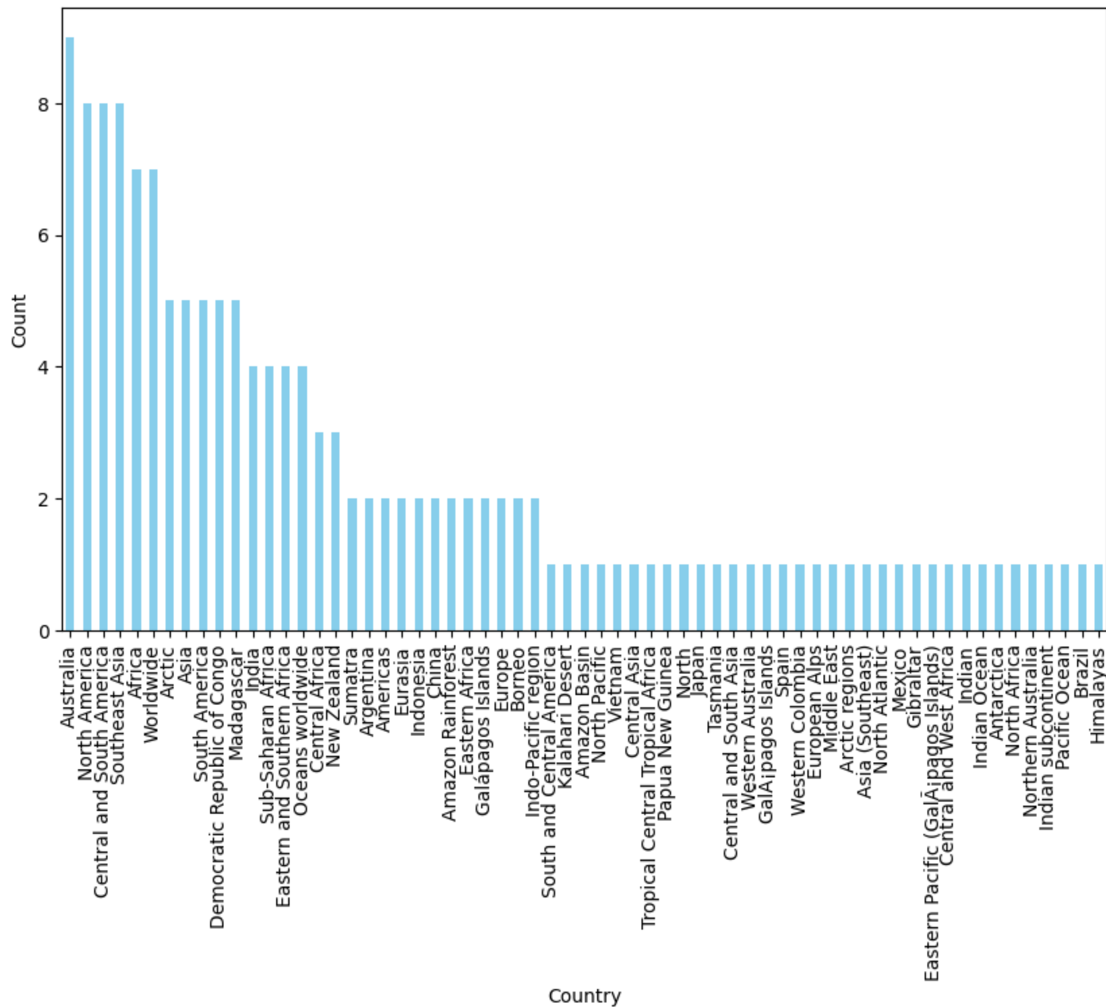
[ ]:
```
# Create a cross-tabulation of Color vs. Diet
color_diet = pd.crosstab(df['Color'], df['Diet'])

# Create a cross-tabulation of Color vs. Habitat
color_habitat = pd.crosstab(df['Color'], df['Habitat 1'])

# Set the figure size
plt.figure(figsize=(18, 6))

# Create a heatmap for Color vs. Diet
plt.subplot(1, 2, 1)
sns.heatmap(color_diet, annot=True, fmt='d', cmap='YlGnBu')
plt.title('Color vs. Diet')

plt.tight_layout()
```



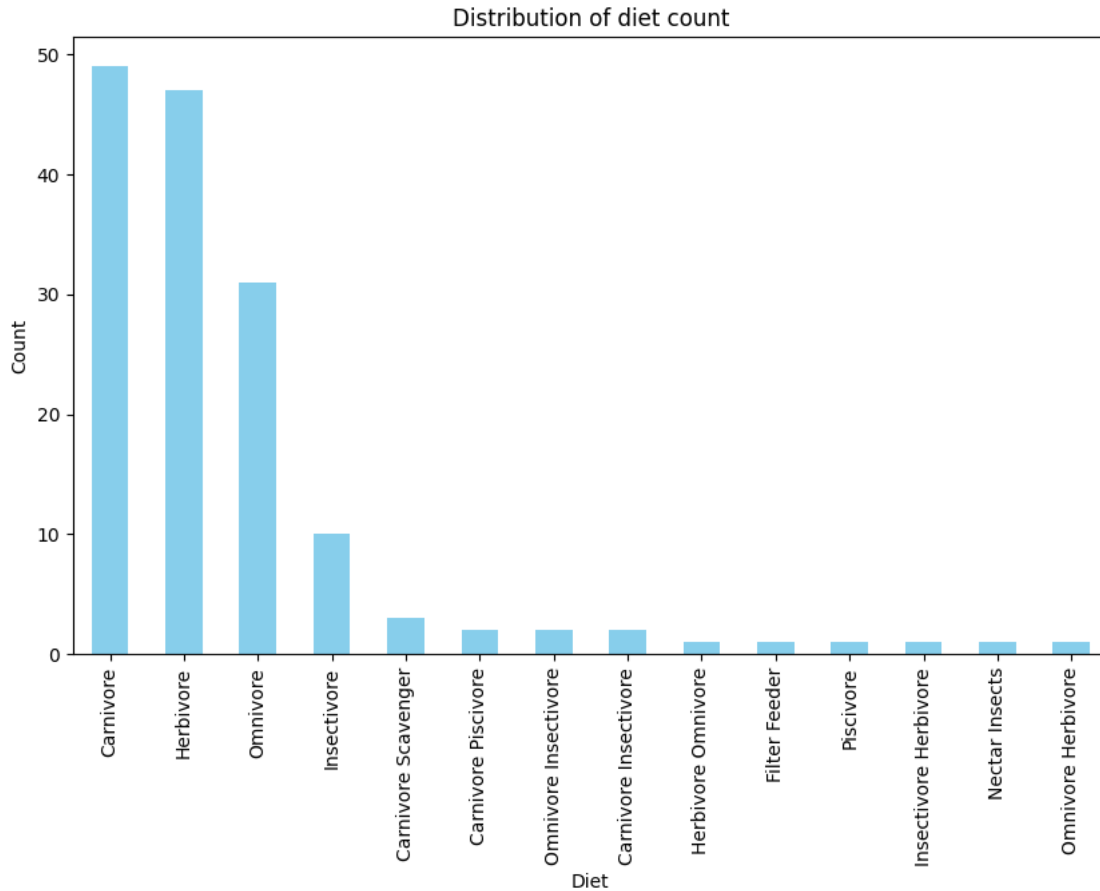

```
[ ]: plt.figure(figsize=(20, 16))
df['Habitat 1'].value_counts().plot(kind='bar', color='skyblue')
plt.xlabel('Habitat 1')
plt.ylabel('Count')
plt.title('Distribution of Habitatsin Animals')
plt.xticks(rotation=45)
plt.show()
```



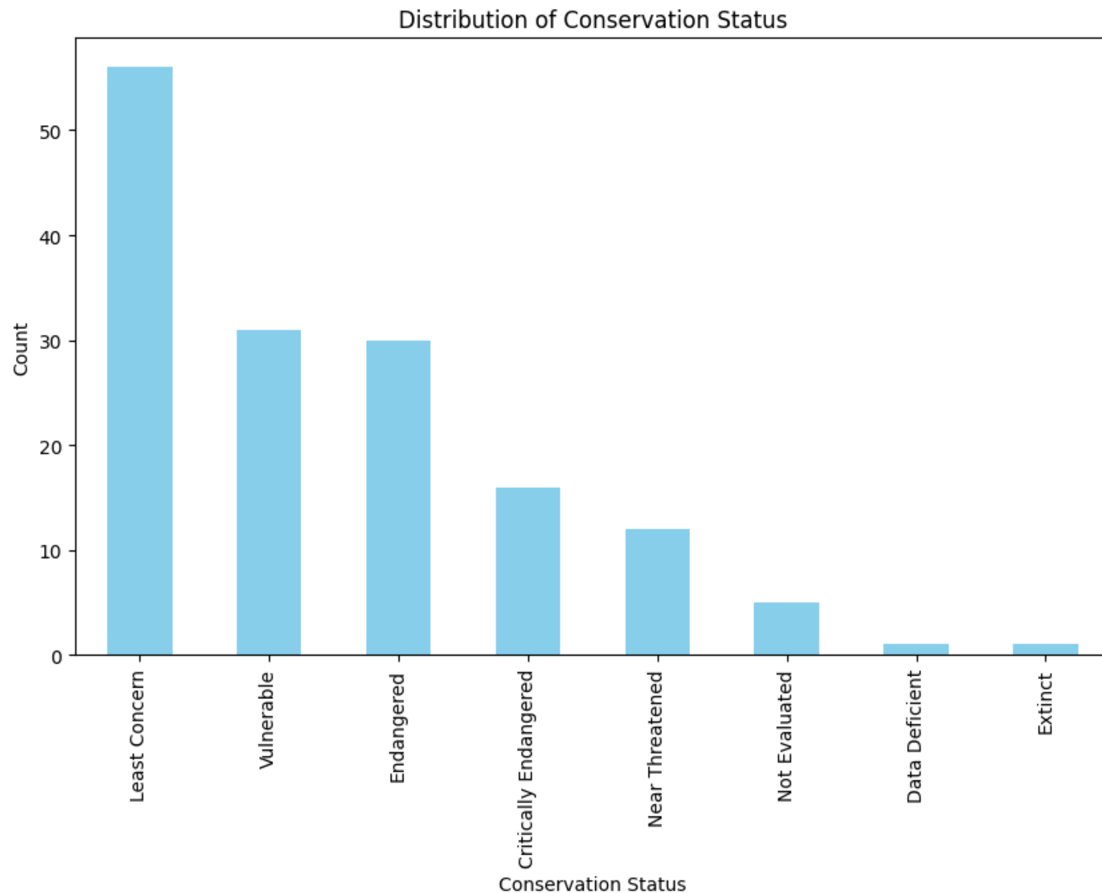
[ ]:
```python
# Count the occurrences of each country
country_counts = df['Countries Found'].value_counts()

# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot to visualize the distribution of countries
country_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Country')
plt.ylabel('Count')
plt.title('Distribution of Countries Where Animals are Found')
plt.show()
```



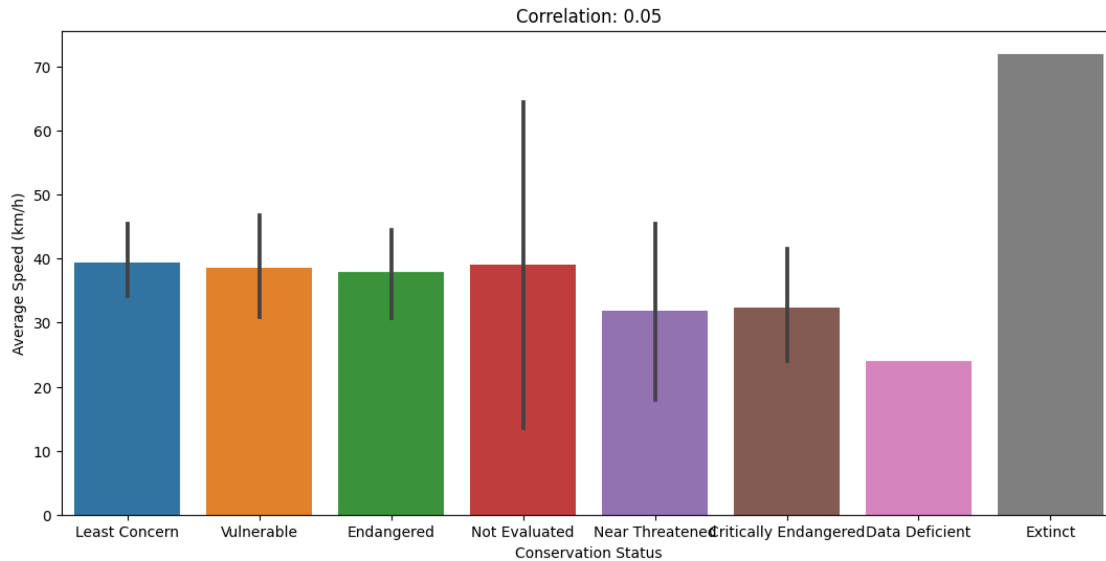
[ ]:
```
# Count the occurrences of each diet
country_counts = df['Diet'].value_counts()

# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot to visualize the distribution of Diet
country_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Diet')
plt.ylabel('Count')
plt.title('Distribution of diet count')
plt.show()
```



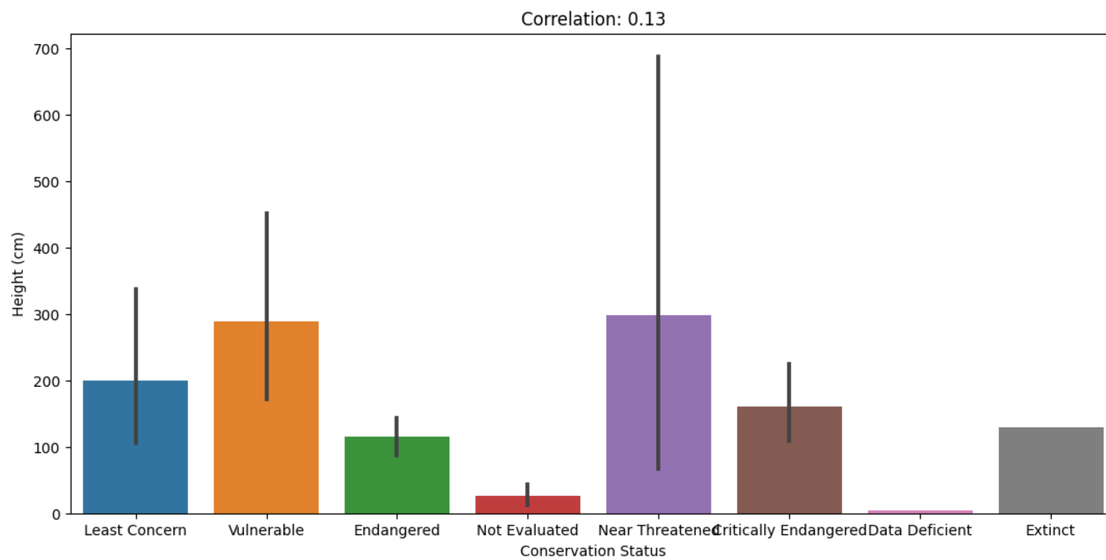
[ ]:
```
# Count the occurrences of each diet
country_counts = df['Conservation Status'].value_counts()

# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot to visualize the distribution of Diet
country_counts.plot(kind='bar', color='skyblue')
plt.xlabel('Conservation Status')
plt.ylabel('Count')
plt.title('Distribution of Conservation Status')
plt.show()
```



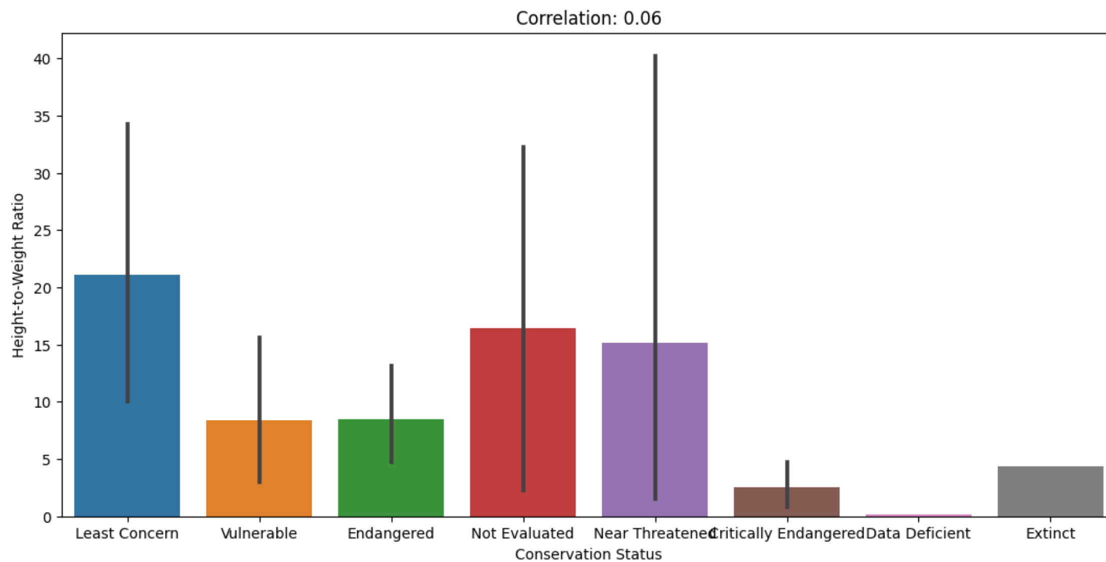
[ ]:
```python
# Perform correlation analysis between Conservation Status and Average Speed
correlation = df['Conservation Status'].astype('category').cat.codes.
    ↪corr(df['Average Speed (km/h)'])

# Create a bar plot to visualize the relationship
plt.figure(figsize=(13, 6))
sns.barplot(x='Conservation Status', y='Average Speed (km/h)', data=df)
plt.title(f'Correlation: {correlation:.2f}')
plt.show()
```



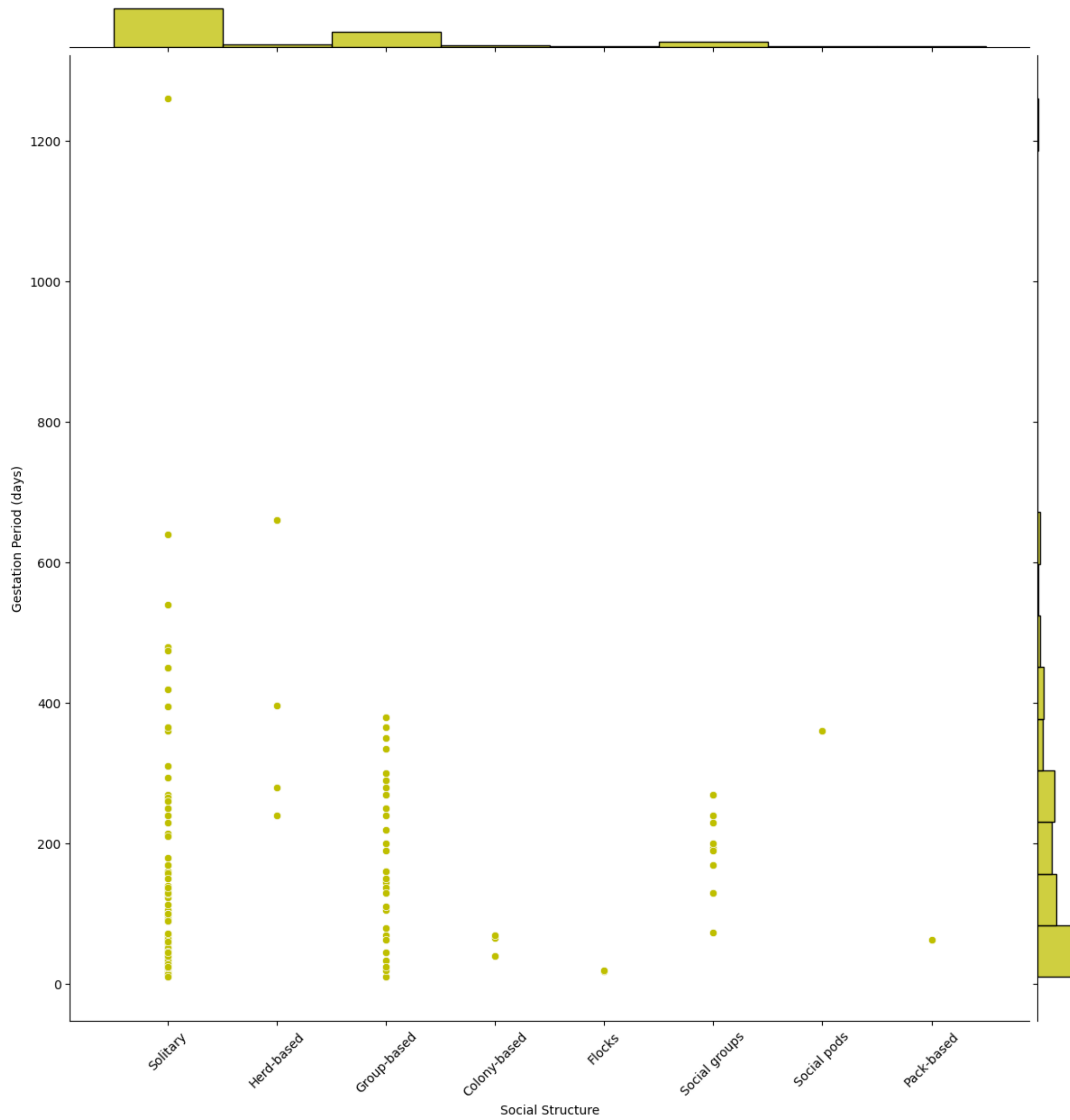
[ ]:
```
# Perform correlation analysis between Conservation Status and Height
correlation = df['Conservation Status'].astype('category').cat.codes.
 ↪corr(df['Height (cm)'])

# Create a bar plot to visualize the relationship
plt.figure(figsize=(13, 6))
sns.barplot(x='Conservation Status', y='Height (cm)', data=df)
plt.title(f'Correlation: {correlation:.2f}')
plt.show()
```



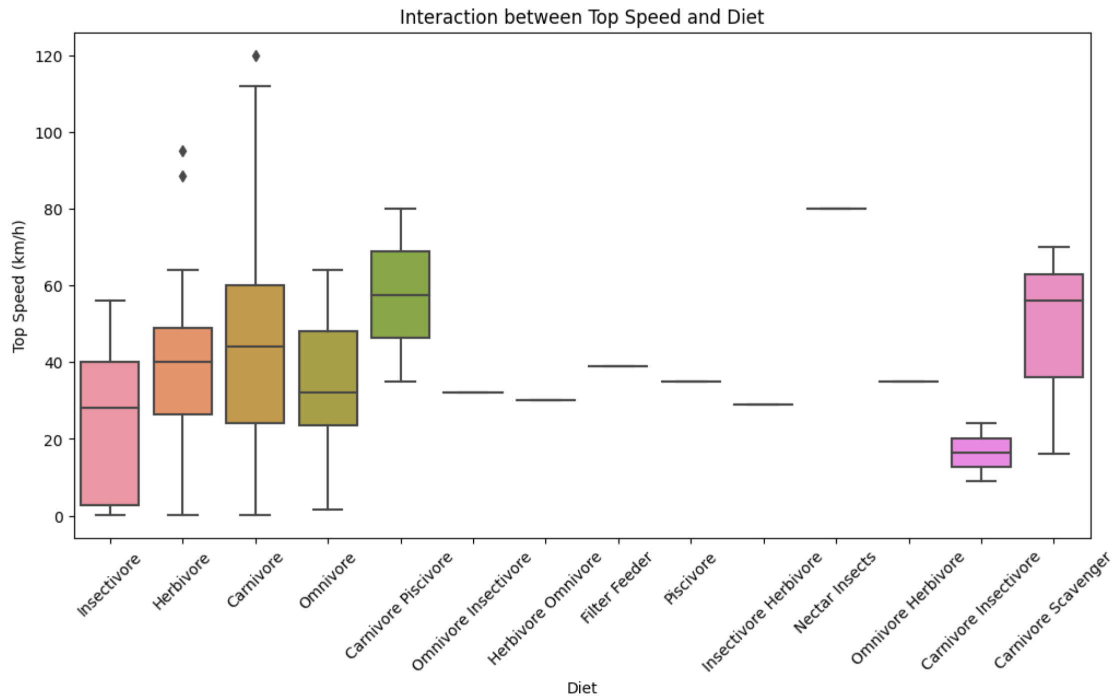
[ ]:
```
# Perform correlation analysis between Conservation Status and Weight
correlation = df['Conservation Status'].astype('category').cat.codes.
 ↪corr(df['Height-to-Weight Ratio'])

# Create a bar plot to visualize the relationship
plt.figure(figsize=(13, 6))
sns.barplot(x='Conservation Status', y='Height-to-Weight Ratio', data=df)
plt.title(f'Correlation: {correlation:.2f}')
plt.show()
```



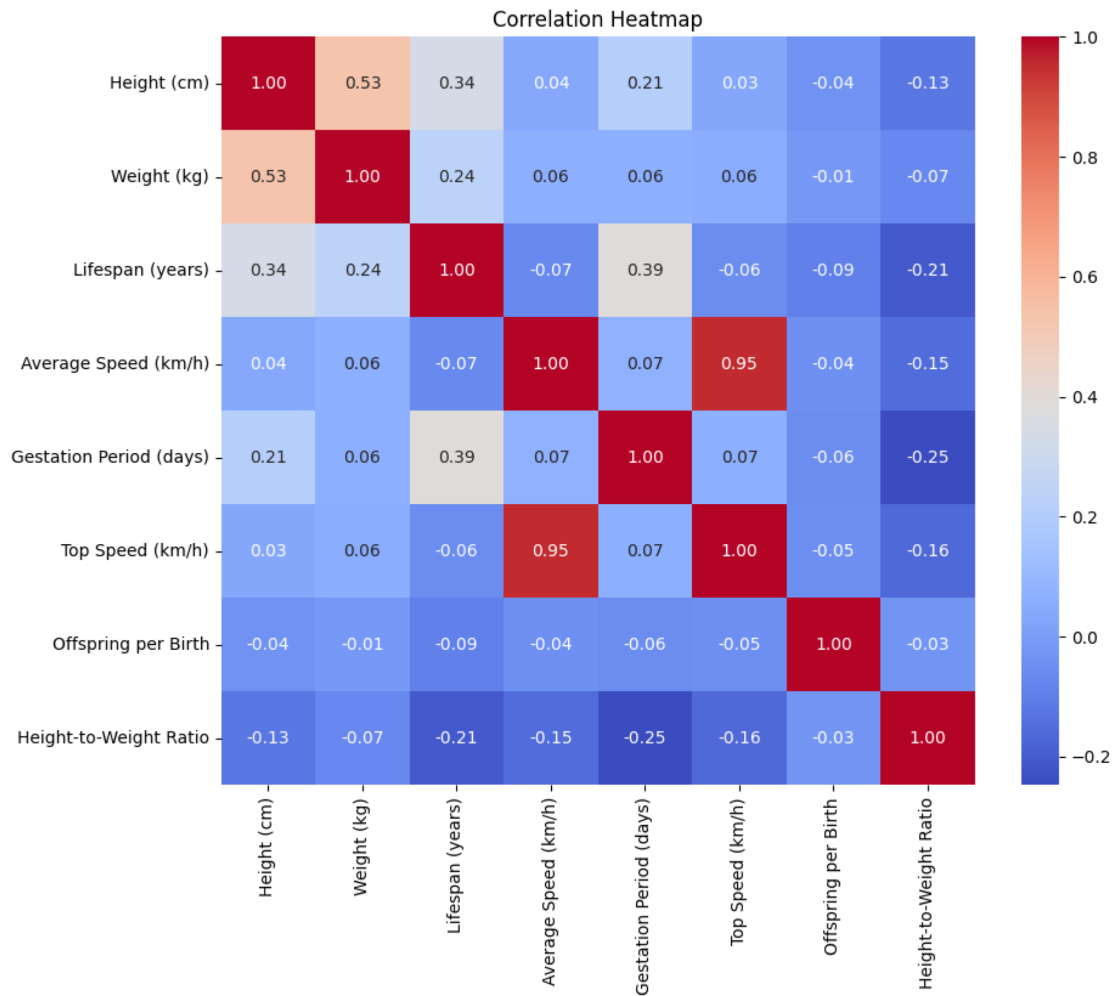
[ ]:
```
sns.jointplot(y="Gestation Period (days)", x="Social Structure", data=df,␣
 ↪height=12, ratio=20, color="y")
plt.xticks(rotation=45)
plt.show()
```



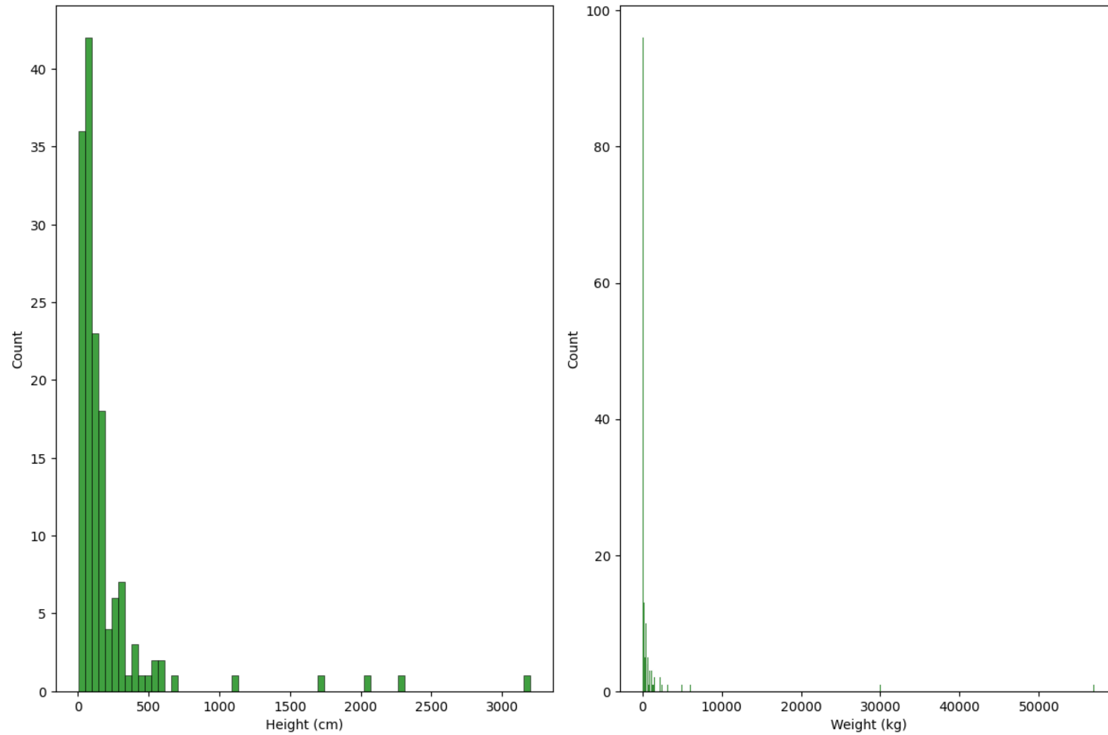

```
[ ]: plt.figure(figsize=(12, 6))
     sns.boxplot(x='Diet', y='Top Speed (km/h)', data=df)
     plt.title('Interaction between Top Speed and Diet')
     plt.xlabel('Diet')
     plt.ylabel('Top Speed (km/h)')
     plt.xticks(rotation=45)
     plt.show()
```



[ ]:
```
numeric_cols = [x for x in df.columns if df[x].dtype in ['float64', 'int64']]
correlation_matrix = df[numeric_cols].corr()

# Create a correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



[ ]:
```
fig ,axes = plt.subplots(nrows=1,ncols=2,figsize=(12,8))

sns.histplot(df['Height (cm)'],ax=axes[0],color='green')
sns.histplot(df['Weight (kg)'],ax=axes[1],color='green')


plt.tight_layout()
plt.show()
```



1.1.2 5) Applying basic DL Models

[39]:
```
pd.unique(df['Diet'])

# Carnivore Scavenger =
# Carnivore Insectivore =
# Omnivore Herbivore =
# Nectar Insects =
# Insectivore Herbivore =
# Piscivore =
# Filter Feeder =
# Herbivore Omnivore =
# Omnivore Insectivore =
# Carnivore Piscivore =
# Omnivore =
# Carnivore =
# Herbivore =
# Insectivore =

# map

# Omnivore =  Omnivore Insectivore, Filter Feeder, Omnivore Herbivore
```

```
# Carnivore => Insectivore, Carnivore Piscivore, Piscivore, Insectivore␣
↪Herbivore, Carnivore Insectivore, Carnivore Scavenger
# Herbivore = Herbivore Omnivore, Nectar Insects

df['Diet'].replace('Omnivore Insectivore', 'Omnivore', inplace=True)
df['Diet'].replace('Filter Feeder', 'Omnivore', inplace=True)
df['Diet'].replace('Omnivore Herbivore', 'Omnivore', inplace=True)

df['Diet'].replace('Insectivore', 'Carnivore', inplace=True)
df['Diet'].replace('Carnivore Piscivore', 'Carnivore', inplace=True)
df['Diet'].replace('Piscivore', 'Carnivore', inplace=True)
df['Diet'].replace('Insectivore Herbivore', 'Carnivore', inplace=True)
df['Diet'].replace('Carnivore Insectivore', 'Carnivore', inplace=True)
df['Diet'].replace('Carnivore Scavenger', 'Carnivore', inplace=True)

df['Diet'].replace('Herbivore Omnivore', 'Herbivore', inplace=True)
df['Diet'].replace('Nectar Insects', 'Herbivore', inplace=True)

pd.unique(df['Diet'])
```

[39]:
```
array(['Carnivore', 'Herbivore', 'Omnivore'], dtype=object)
```

[40]:
```
df
```

[40]:
```
                 Animal  Height (cm)  Weight (kg)             Color  \
0              Aardvark        130.0         65.0              Grey
1              Aardwolf         50.0         14.0      Yellow-brown
2      African Elephant        310.0       6000.0              Grey
3          African Lion        110.0        250.0               Tan
4      African Wild Dog         80.0         36.0      Multicolored
..                  …            …            …                 …
197              Wombat        118.0         35.0        Brown Gray
200                 Yak        160.0       1200.0       Brown Black
201  Yellow-Eyed Penguin        65.0          3.0      Yellow White
203               Zebra        340.0        900.0       Black White
204         Zebra Shark        330.0         32.0   Brown Yellowish

     Lifespan (years)       Diet  Average Speed (km/h) Conservation Status  \
0                30.0  Carnivore                  40.0       Least Concern
1                12.0  Carnivore                  30.0       Least Concern
2                70.0  Herbivore                  25.0          Vulnerable
3                14.0  Carnivore                  58.0          Vulnerable
4                12.0  Carnivore                  56.0          Endangered
..                …          …                   …                   …
197              10.0  Herbivore                  20.0       Least Concern
200              25.0  Herbivore                  24.0       Least Concern
201              20.0  Carnivore                  25.0          Endangered
```

```
203             25.0  Herbivore                  25.0       Least Concern
204             30.0  Carnivore                  20.0          Endangered

              Family  Gestation Period (days)  Top Speed (km/h)  \
0    Orycteropodidae                    240.0              40.0
1          Hyaenidae                     90.0              40.0
2       Elephantidae                    660.0              40.0
3            Felidae                    105.0              80.0
4            Canidae                     70.0              56.0
..               …                        …                 …
197       Vombatidae                    138.0              20.0
200          Bovidae                    280.0              24.0
201     Spheniscidae                     90.0              25.0
203          Equidae                    365.0              25.0
204  Stegostomatidae                     25.0              20.0

    Social Structure  Offspring per Birth             Countries Found  \
0           Solitary                  1.0                      Africa
1           Solitary                  5.0  Eastern and Southern Africa
2         Herd-based                  1.0                      Africa
3        Group-based                  NaN                      Africa
4        Group-based                 12.0          Sub-Saharan Africa
..               …                  …                           …
197         Solitary                  1.0                   Australia
200      Group-based                 50.0                    Himalayas
201         Solitary                  1.0                 New Zealand
203      Group-based                 20.0                      Africa
204         Solitary                 25.0         Indo-Pacific region

         Habitat 1      Habitat 2    Predators 1       Predators 2  \
0         Savannas     Grasslands          Lions            Hyenas
1       Grasslands       Savannas          Lions          Leopards
2         Savannah         Forest          Lions            Hyenas
3       Grasslands       Savannas         Hyenas        Crocodiles
4        Savannahs      Savannahs          Lions            Hyenas
..             …              …              …                 …
197        Forests     Grasslands        Dingoes  Tasmanian Devils
200      Mountains      Mountains  Snow Leopards            Wolves
201  Coastal Areas  Coastal Areas          Seals             Orcas
203     Grasslands     Grasslands          Lions            Hyenas
204    Coral Reefs    Coral Reefs    Larger Fish       Larger Fish

     Height-to-Weight Ratio
0                  2.000000
1                  3.571429
2                  0.051667
3                  0.440000
```

```
4                   2.222222
..                       …
197                 3.371429
200                 0.133333
201                21.666667
203                 0.377778
204                10.312500

[152 rows x 19 columns]
```

[41]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df = df.dropna()
for i in ['Color', 'Conservation Status', 'Family', 'Social Structure',␣
↪'Countries Found', 'Habitat 1', 'Habitat 2', 'Predators 1', 'Predators 2']:
    df[i] = df[i].astype('category')
    df[i + '_Codes'] = df[i].cat.codes

x_data = df.drop(['Offspring per Birth','Diet', 'Animal', 'Color',␣
↪'Conservation Status', 'Family', 'Social Structure', 'Countries Found',␣
↪'Habitat 1', 'Habitat 2', 'Predators 1', 'Predators 2'],axis=1)
y_data = df['Diet']
```

[42]:
```
x_data
```

[42]:
```
     Height (cm)  Weight (kg)  Lifespan (years)  Average Speed (km/h)  \
0          130.0         65.0              30.0                  40.0
1           50.0         14.0              12.0                  30.0
2          310.0       6000.0              70.0                  25.0
4           80.0         36.0              12.0                  56.0
5          101.0        120.0              20.0                  64.0
..           …           …               …                    …
197        118.0         35.0              10.0                  20.0
200        160.0       1200.0              25.0                  24.0
201         65.0          3.0              20.0                  25.0
203        340.0        900.0              25.0                  25.0
204        330.0         32.0              30.0                  20.0

     Gestation Period (days)  Top Speed (km/h)  Height-to-Weight Ratio  \
0                      240.0              40.0                2.000000
1                       90.0              40.0                3.571429
2                      660.0              40.0                0.051667
```

```
4                        70.0            56.0                 2.222222
5                       150.0            64.0                 0.841667
..                        …               …                    …
197                     138.0            20.0                 3.371429
200                     280.0            24.0                 0.133333
201                      90.0            25.0                21.666667
203                     365.0            25.0                 0.377778
204                      25.0            20.0                10.312500

     Color_Codes  Conservation Status_Codes  Family_Codes  \
0             38                          4            54
1             58                          4            40
2             38                          7            27
4             40                          2            14
5             10                          4            11
..           …                         …            …
197           13                          4            84
200           11                          4            11
201           57                          2            67
203            6                          4            29
204           21                          2            68

     Social Structure_Codes  Countries Found_Codes  Habitat 1_Codes  \
0                         7                      0               36
1                         7                     22               20
2                         3                      0               34
4                         2                     55               35
5                         2                     25               26
..                      …                      …                …
197                       7                     10               17
200                       2                     29               26
201                       7                     41               10
203                       2                      0               20
204                       7                     34               12

     Habitat 2_Codes  Predators 1_Codes  Predators 2_Codes
0                 22                 21                 23
1                 39                 21                 30
2                 19                 21                 23
4                 38                 21                 23
5                  0                 33                 16
..              …                  …                  …
197               22                  7                 44
200               30                 29                 48
201               12                 26                 34
203               22                 21                 23
204               14                 17                 26
```

[150 rows x 16 columns]

[43]:
```
MinMaxScaler = preprocessing.MinMaxScaler()
X_data_minmax = MinMaxScaler.fit_transform(x_data)
data = pd.DataFrame(X_data_minmax,columns=x_data.columns)
data.head()
# df.head()
```

[43]:
```
   Height (cm)  Weight (kg)  Lifespan (years)  Average Speed (km/h)  \
0     0.039124     0.001139          0.178082              0.331997
1     0.014085     0.000245          0.054795              0.248497
2     0.095462     0.105262          0.452055              0.206747
3     0.023474     0.000631          0.054795              0.465598
4     0.030047     0.002104          0.109589              0.532398

   Gestation Period (days)  Top Speed (km/h)  Height-to-Weight Ratio  \
0                    0.184          0.331997                0.007868
1                    0.064          0.331997                0.014154
2                    0.520          0.331997                0.000073
3                    0.048          0.465598                0.008757
4                    0.112          0.532398                0.003234

   Color_Codes  Conservation Status_Codes  Family_Codes  \
0     0.612903                   0.571429      0.642857
1     0.935484                   0.571429      0.476190
2     0.612903                   1.000000      0.321429
3     0.645161                   0.285714      0.166667
4     0.161290                   0.571429      0.130952

   Social Structure_Codes  Countries Found_Codes  Habitat 1_Codes  \
0                1.000000               0.000000         0.837209
1                1.000000               0.354839         0.465116
2                0.428571               0.000000         0.790698
3                0.285714               0.887097         0.813953
4                0.285714               0.403226         0.604651

   Habitat 2_Codes  Predators 1_Codes  Predators 2_Codes
0             0.44           0.636364           0.479167
1             0.78           0.636364           0.625000
2             0.38           0.636364           0.479167
3             0.76           0.636364           0.479167
4             0.00           1.000000           0.333333
```

[44]:
```
pd.unique(df['Diet'])
df['Diet'] = df['Diet'].map({'Herbivore' :0, 'Carnivore' :1, 'Omnivore' :2}).
 ↪astype(int) #mapping numbers
```

```
pd.unique(df['Diet'])
```

[44]:
```
array([1, 0, 2])
```

[45]:
```
df
```

[45]:
```
                  Animal  Height (cm)  Weight (kg)            Color  \
0               Aardvark        130.0         65.0             Grey
1               Aardwolf         50.0         14.0     Yellow-brown
2       African Elephant        310.0       6000.0             Grey
4       African Wild Dog         80.0         36.0     Multicolored
5            Alpine Ibex        101.0        120.0            Brown
..                   …            …           …              …
197               Wombat        118.0         35.0       Brown Gray
200                  Yak        160.0       1200.0      Brown Black
201   Yellow-Eyed Penguin         65.0          3.0     Yellow White
203                Zebra        340.0        900.0      Black White
204          Zebra Shark        330.0         32.0  Brown Yellowish

     Lifespan (years)  Diet  Average Speed (km/h) Conservation Status  \
0                30.0     1                  40.0       Least Concern
1                12.0     1                  30.0       Least Concern
2                70.0     0                  25.0          Vulnerable
4                12.0     1                  56.0          Endangered
5                20.0     0                  64.0       Least Concern
..                 …   …                   …                 …
197              10.0     0                  20.0       Least Concern
200              25.0     0                  24.0       Least Concern
201              20.0     1                  25.0          Endangered
203              25.0     0                  25.0       Least Concern
204              30.0     1                  20.0          Endangered

              Family  Gestation Period (days)  …  Height-to-Weight Ratio  \
0     Orycteropodidae                    240.0  …                2.000000
1           Hyaenidae                     90.0  …                3.571429
2        Elephantidae                    660.0  …                0.051667
4             Canidae                     70.0  …                2.222222
5             Bovidae                    150.0  …                0.841667
..                 …                    …  …                     …
197        Vombatidae                    138.0  …                3.371429
200           Bovidae                    280.0  …                0.133333
201      Spheniscidae                     90.0  …               21.666667
203           Equidae                    365.0  …                0.377778
204  Stegostomatidae                     25.0  …               10.312500

    Color_Codes  Conservation Status_Codes Family_Codes  \
0            38                          4           54
```

```
1            58                        4          40
2            38                        7          27
4            40                        2          14
5            10                        4          11
..          ...                      ...         ...
197          13                        4          84
200          11                        4          11
201          57                        2          67
203           6                        4          29
204          21                        2          68

    Social Structure_Codes  Countries Found_Codes  Habitat 1_Codes  \
0                        7                      0               36
1                        7                     22               20
2                        3                      0               34
4                        2                     55               35
5                        2                     25               26
..                     ...                    ...              ...
197                      7                     10               17
200                      2                     29               26
201                      7                     41               10
203                      2                      0               20
204                      7                     34               12

    Habitat 2_Codes  Predators 1_Codes  Predators 2_Codes
0                22                 21                 23
1                39                 21                 30
2                19                 21                 23
4                38                 21                 23
5                 0                 33                 16
..              ...                ...                ...
197              22                  7                 44
200              30                 29                 48
201              12                 26                 34
203              22                 21                 23
204              14                 17                 26

[150 rows x 28 columns]
```

[46]:
```
X_train, X_test, y_train, y_test = train_test_split(data, y_data,test_size=0.2,␣
 ↪random_state = 1)
knn_clf=KNeighborsClassifier()
knn_clf.fit(X_train,y_train)
ypred=knn_clf.predict(X_test) #These are the predicted output values
```

[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
```

```
result = confusion_matrix(y_test, ypred)
print('Confusion Matrix:')
print(result)
result1 = classification_report(y_test, ypred)
print('Classification Report:')
print (result1)
result2 = accuracy_score(y_test,ypred)
print('Accuracy:',result2)
```

```
Confusion Matrix:
[[ 4  3  2]
 [ 4 10  2]
 [ 3  2  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.36      0.44      0.40         9
           1       0.67      0.62      0.65        16
           2       0.00      0.00      0.00         5

    accuracy                           0.47        30
   macro avg       0.34      0.36      0.35        30
weighted avg       0.46      0.47      0.46        30

Accuracy: 0.4666666666666667
```

1.1.3 6) Creating the Graph

[ ]:
```
!pip install torch_geometric
```

```
Collecting torch_geometric
  Using cached torch_geometric-2.5.3-py3-none-any.whl (1.1 MB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (4.66.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (1.11.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (2023.6.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (3.1.4)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (3.9.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (2.31.0)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-
```

```
packages (from torch_geometric) (3.1.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (1.2.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (5.9.5)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->torch_geometric) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->torch_geometric) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (4.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch_geometric) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->torch_geometric) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->torch_geometric) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->torch_geometric) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->torch_geometric)
(2024.7.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn->torch_geometric) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->torch_geometric)
(3.5.0)
Installing collected packages: torch_geometric
Successfully installed torch_geometric-2.5.3
Requirement already satisfied: torch_geometric in
/usr/local/lib/python3.10/dist-packages (2.5.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (4.66.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (1.11.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (2023.6.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch_geometric) (3.1.4)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-
```

```
packages (from torch_geometric) (3.9.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (2.31.0)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (3.1.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (1.2.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (5.9.5)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->torch_geometric) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-
packages (from aiohttp->torch_geometric) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->torch_geometric) (4.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch_geometric) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->torch_geometric) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->torch_geometric) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->torch_geometric) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->torch_geometric)
(2024.7.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn->torch_geometric) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->torch_geometric)
(3.5.0)
```

[79]:

```
# Clean the data by dropping rows with NaN values
df = df.dropna()

# Convert categorical variables to numerical codes
for i in ['Color', 'Conservation Status', 'Family', 'Social Structure',␣
 ↪'Countries Found', 'Habitat 1', 'Habitat 2', 'Predators 1', 'Predators 2']:
    df[i] = df[i].astype('category')
    df[i + '_Codes'] = df[i].cat.codes
```

```
# Prepare x_data by dropping specified columns
x_data = df.drop(['Offspring per Birth', 'Diet', 'Animal', 'Color',␣
↪'Conservation Status', 'Family', 'Social Structure', 'Countries Found',␣
↪'Habitat 1', 'Habitat 2', 'Predators 1', 'Predators 2'], axis=1)

# Prepare y_data as the 'Diet' column
y_data = df['Diet']
```

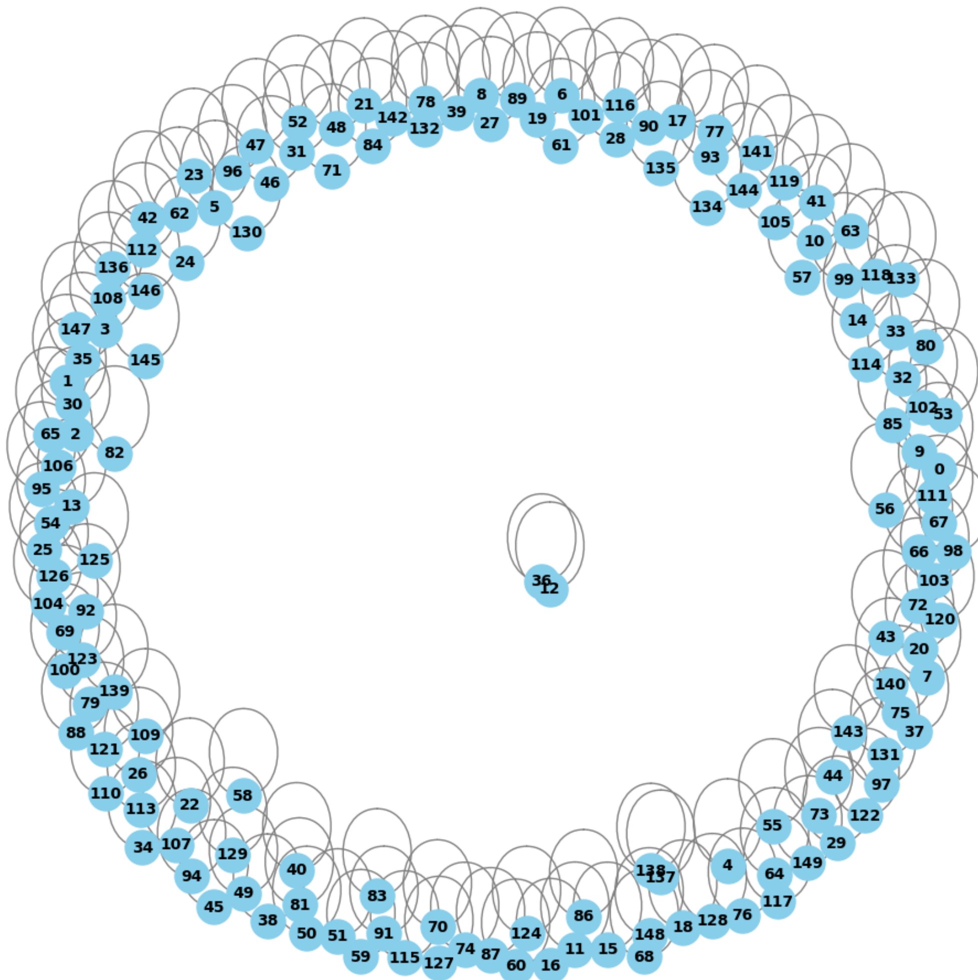
Data Processing and Graph Creation with PyTorch Geometric

[80]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances
import torch
from torch_geometric.data import Data
import numpy as np

# Standardize the node features
scaler = StandardScaler()
node_features = scaler.fit_transform(x_data)

# Define a similarity threshold to create edges
threshold = 0.5  # This can be adjusted based on the dataset
distances = euclidean_distances(node_features)
edges = np.where(distances < threshold)
edge_index = torch.tensor(edges, dtype=torch.long)

# Create the adjacency matrix
adj_t = torch.tensor(distances < threshold, dtype=torch.float32)

# Assuming there is a column in the dataset for labels
labels = y_data  # Replace 'label_column' with the actual label column name
y = torch.tensor(labels.values, dtype=torch.long)

# Split the data for training and testing (assuming 80-20 split)
num_nodes = len(df)
train_size = int(0.8 * num_nodes)
indices = np.random.permutation(num_nodes)
train_indices = indices[:train_size]
test_indices = indices[train_size:]
split_idx = {'train': torch.tensor(train_indices, dtype=torch.long),
             'test': torch.tensor(test_indices, dtype=torch.long)}

# Create the PyTorch Geometric Data object without unsqueezing y
data = Data(x=torch.tensor(node_features, dtype=torch.float32),
            edge_index=edge_index,
            adj_t=adj_t,
```

```
            y=y,
            split_idx=split_idx,
            num_classes=3)  # Replace 3 with the actual number of classes

# Save the data object for further use
torch.save(data, 'graph_data.pt')
```

Graph Visualization with NetworkX and Torch Geometric

[81]:
```
import torch
import networkx as nx
import matplotlib.pyplot as plt
from torch_geometric.data import Data

# Load the graph data
data = torch.load('graph_data.pt')

# Convert edge_index to NetworkX format
edges = data.edge_index.numpy().T  # Transpose to get edge pairs
G = nx.Graph()
G.add_edges_from(edges)

# Draw the graph
plt.figure(figsize=(10, 10))
pos = nx.spring_layout(G)  # Layout for visualization
nx.draw(G, pos, with_labels=True, node_size=500, node_color='skyblue',␣
 ↪font_size=10, font_color='black', font_weight='bold', edge_color='gray')

plt.title('Graph Visualization')
plt.show()
```



1.1.4 7) Applying GNNs for Node prediction

GNN Model for Classifying Animal Diets

[78]:
```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

# Define the Graph Neural Network (GNN) model
class GNNModel(torch.nn.Module):
    def __init__(self, num_node_features, num_classes):
        super(GNNModel, self).__init__()
        # Initialize two graph convolution layers
```

```
        self.conv1 = GCNConv(num_node_features, 16)  # First convolutional layer
        self.conv2 = GCNConv(16, num_classes)        # Second convolutional␣
 ↪layer

    def forward(self, data):
        # Get node features and edge indices from the data
        x, edge_index = data.x, data.edge_index
        # Apply the first convolutional layer and ReLU activation
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        # Apply the second convolutional layer
        x = self.conv2(x, edge_index)
        # Apply log softmax to get log probabilities
        return F.log_softmax(x, dim=1)

# Load the graph data from a file
data = torch.load('graph_data.pt')

# Create boolean masks for training and testing sets
train_mask = torch.zeros(data.num_nodes, dtype=torch.bool)
test_mask = torch.zeros(data.num_nodes, dtype=torch.bool)

# Assign True to training and testing indices based on the split
train_mask[data.split_idx['train']] = True
test_mask[data.split_idx['test']] = True

# Add the masks to the data object
data.train_mask = train_mask
data.test_mask = test_mask

# Use GPU if available, otherwise fallback to CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Initialize the model and move it to the chosen device
model = GNNModel(data.num_node_features, data.num_classes).to(device)
# Move the data to the chosen device
data = data.to(device)

# Initialize the optimizer with model parameters, learning rate, and weight␣
 ↪decay
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

# Training function
def train():
    model.train()  # Set the model to training mode
    optimizer.zero_grad()  # Reset gradients
    out = model(data)  # Forward pass
```

```python
    loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])  # Compute␣
↪loss
    loss.backward()  # Backward pass
    optimizer.step()  # Update weights
    return loss.item()

# Testing function
def test():
    model.eval()  # Set the model to evaluation mode
    out = model(data)  # Forward pass
    pred = out.argmax(dim=1)  # Get predictions
    correct = pred[data.test_mask] == data.y[data.test_mask]  # Compare␣
↪predictions with true labels
    accuracy = int(correct.sum()) / int(data.test_mask.sum())  # Calculate␣
↪accuracy
    return accuracy

# Training loop
for epoch in range(200):
    loss = train()  # Train the model for one epoch
    if epoch % 10 == 0:  # Every 10 epochs, evaluate the model
        accuracy = test()
        print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}, Test Accuracy: {accuracy:.
↪4f}')

# Final test accuracy
final_accuracy = test()
print(f'Final Test Accuracy: {final_accuracy:.4f}')
```

```
Epoch: 000, Loss: 1.2444, Test Accuracy: 0.5667
Epoch: 010, Loss: 0.8213, Test Accuracy: 0.6333
Epoch: 020, Loss: 0.7087, Test Accuracy: 0.5667
Epoch: 030, Loss: 0.6178, Test Accuracy: 0.6000
Epoch: 040, Loss: 0.5309, Test Accuracy: 0.6000
Epoch: 050, Loss: 0.4459, Test Accuracy: 0.6000
Epoch: 060, Loss: 0.3620, Test Accuracy: 0.6000
Epoch: 070, Loss: 0.2903, Test Accuracy: 0.6000
Epoch: 080, Loss: 0.2285, Test Accuracy: 0.5333
Epoch: 090, Loss: 0.1780, Test Accuracy: 0.5333
Epoch: 100, Loss: 0.1392, Test Accuracy: 0.5333
Epoch: 110, Loss: 0.1094, Test Accuracy: 0.5333
Epoch: 120, Loss: 0.0872, Test Accuracy: 0.5333
Epoch: 130, Loss: 0.0708, Test Accuracy: 0.5333
Epoch: 140, Loss: 0.0589, Test Accuracy: 0.5333
Epoch: 150, Loss: 0.0501, Test Accuracy: 0.5000
Epoch: 160, Loss: 0.0433, Test Accuracy: 0.5000
Epoch: 170, Loss: 0.0381, Test Accuracy: 0.5333
```

```
Epoch: 180, Loss: 0.0338, Test Accuracy: 0.5333
Epoch: 190, Loss: 0.0299, Test Accuracy: 0.5333
Final Test Accuracy: 0.5667
```

1.1.5 Explanation of Accuracies

Basic Model Accuracy (46%): - Reason for Low Accuracy: - Imbalanced Dataset: The dataset might have an uneven distribution of the three diet classes (Carnivores, Omnivores, Herbivores), leading to the model being biased towards the majority class. - **Insufficient Feature Separation:** The features might not be distinct enough for the basic model to differentiate between the classes effectively. - **Model Complexity:** The basic model might be too simplistic to capture complex patterns in the data.

GNN Model Accuracy (53%): - Reason for Slightly Higher Accuracy: - Graph Structure: GNNs can leverage the relationships and dependencies between data points, which might be present implicitly in the dataset. - **Enhanced Learning:** GNNs can capture more intricate patterns due to their ability to update node features iteratively during training.

1.1.6 Are GNNs Suitable for Any Dataset?

Not Always Suitable: GNNs are particularly effective for datasets with an inherent graph structure, such as social networks, citation networks, or molecular structures. For purely tabular data, GNNs may not provide significant advantages unless meaningful relationships between data points can be defined.

1.1.7 Is the Procedure of Converting a Tabular Dataset to a Graph Dataset Viable?

Conversion Feasibility: Converting a tabular dataset to a graph dataset can be viable if meaningful relationships between rows (data points) are defined. For instance, similarity in features, geographical proximity, or domain-specific connections can be used to create edges. **Challenges:** Defining meaningful edges and node features from tabular data requires domain expertise and may not always be straightforward.

1.1.8 Do You See Overfitting?

No

1.1.9 Can You Improve the Results by Any Means?

Techniques to Improve Results: 1. **Feature Engineering:** Creating new features or transforming existing ones to better capture the underlying patterns. 2. **Balancing the Dataset:** Using techniques like oversampling, undersampling, or class weighting to handle imbalanced data. 3. **Hyperparameter Tuning:** Experimenting with different model architectures, learning rates, and other hyperparameters. 4. **Ensembling:** Combining multiple models to leverage their strengths and improve overall performance.

1.1.10 Does the Model Need to Update Node Features Alongside Weights?

Node Feature Updates: In GNNs, node features are updated iteratively during the message-passing process. This is crucial for capturing the evolving state of each node based on its neighbors.

Weight Updates: As with any neural network, weights in a GNN are updated during training to minimize the loss function.

1.1.11 Draw a Table Comparing the Results with the Results from Basic Models

Model	Accuracy (%)
Basic Model	46
GNN Model	53

1.1.12 Mention and Explain Your Idea to Improve the Basic GNN Models

Ideas to Improve Basic GNN Models:

1. **Enhanced Feature Engineering:** Create additional features that better represent the data, such as interaction terms, polynomial features, or domain-specific transformations.
2. **Graph Construction Optimization:** Improve the way the graph is constructed from the tabular data. Ensure that the edges and node features are meaningful and relevant to the classification task.
3. **Regularization Techniques:** Use dropout, L2 regularization, and other techniques to prevent overfitting.
4. **Data Augmentation:** Augment the training data by creating synthetic samples or using techniques like SMOTE.
5. **Hyperparameter Optimization:** Use grid search, random search, or Bayesian optimization to find the best hyperparameters for the GNN model.
6. **Advanced GNN Architectures:** Experiment with more advanced GNN architectures like GraphSAGE, GAT (Graph Attention Networks), or R-GCN (Relational GCN) to better capture the complex relationships in the data.
7. **Transfer Learning:** Use pre-trained GNN models on similar datasets and fine-tune them on your specific dataset.