Project - 🐦 Bird Species Observation Analysis

📝 Project Summary:

This project involved analyzing two datasets containing bird species observations across various forest and grassland sites. The dataset included detailed records of bird sightings, covering attributes such as location, observation methods, species names, and environmental conditions. The dataset, in Excel format, comprised multiple sheets, each representing a specific administrative unit, with data linked through a common Admin_Unit_Code. Each sheet provided localized insights for its respective region. By integrating and analyzing these datasets, the project aimed to identify trends in species distribution, assess regional biodiversity, and explore the impact of environmental factors on bird populations.

📌 A comprehensive Exploratory Data Analysis (EDA) was conducted to identify key patterns, such as:

- **Data Cleaning and Preprocessing**: Handled missing values, standardized formats, and ensured data quality for accurate analysis.
- **Exploratory Data Analysis (EDA)**: Discovered trends, seasonal patterns, and distribution of bird sightings using descriptive statistics.
- **Data Visualization**: Created interactive charts and graphs (e.g., species counts, observation trends) for better data understanding.
- **Geographic Analysis**: Mapped bird observations across different regions to identify biodiversity hotspots and migratory patterns.
- **Species Analysis**: Analyzed frequency and diversity of bird species, focusing on rare and endangered sightings.

The project maintained clean, well-commented code, allowing for easy reuse and reproducibility. The final presentation emphasized key insights with actionable recommendations for biodiversity management and environmental planning.

🔍 Problem Statement:

The project aims to analyze the distribution and diversity of bird species across two contrasting ecosystems—forests and grasslands. By studying observational data, the goal is to identify patterns of habitat preference and assess how environmental factors like vegetation, climate, and terrain influence bird populations and behavior. This analysis will help uncover the impact of different habitats on bird diversity and provide valuable insights for biodiversity conservation, habitat management, and understanding the ecological effects of environmental changes on avian species.

Plotly for Data Visualization

In [1]:
```
# Installing plotly
# !pip install plotly - I have commented after the installation
```

Import Libraries

In [2]:
```
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
```

In [3]:
```
# Specify the file path - Here, I'm reading 'Forest data'
file_path = "Bird_Monitoring_Data_FOREST.xlsx"

# Read the Excel file with multiple sheets
forest_data = pd.ExcelFile(file_path)

# Get all sheet names
sheet_names = forest_data.sheet_names

# Read data from all sheets into a dictionary
sheets_dict = {sheet: forest_data.parse(sheet) for sheet in sheet_names}
```

In [4]:
```
print(sheet_names)
# print(sheets_dict)
```

```
['ANTI', 'CATO', 'CHOH', 'GWMP', 'HAFE', 'MANA', 'MONO', 'NACE', 'PRWI', 'ROCR', 'WOTR']
```

In [5]:
```
# Example: Convert sheets_dict to a single DataFrame and stored combined dataframe in forest;
forest = pd.concat(
    [df.assign(Sheet=sheet_name) for sheet_name, df in sheets_dict.items()],
    ignore_index=True
)
```

In [6]:
```
forest.head(3)
```

Out[6]:

	Admin_Unit_Code	Sub_Unit_Code	Site_Name	Plot_Name	Location_Type	Year	Date	Start_Time
0	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:19:00
1	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:19:00
2	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:19:00

3 rows × 30 columns

In [7]:
```
print(forest.shape)
forest.columns
```

```
(8546, 30)
```

Out[7]:
```
Index(['Admin_Unit_Code', 'Sub_Unit_Code', 'Site_Name', 'Plot_Name',
       'Location_Type', 'Year', 'Date', 'Start_Time', 'End_Time', 'Observer',
       'Visit', 'Interval_Length', 'ID_Method', 'Distance', 'Flyover_Observed',
       'Sex', 'Common_Name', 'Scientific_Name', 'AcceptedTSN', 'NPSTaxonCode',
       'AOU_Code', 'PIF_Watchlist_Status', 'Regional_Stewardship_Status',
       'Temperature', 'Humidity', 'Sky', 'Wind', 'Disturbance',
       'Initial_Three_Min_Cnt', 'Sheet'],
      dtype='object')
```

In [8]:
```
# Specify the file path - Here, I'm reading 'Grassland data'
file_path = "Bird_Monitoring_Data_GRASSLAND.xlsx"
```

```python
# Read the Excel file with multiple sheets
grassland_data = pd.ExcelFile(file_path)

# Get all sheet names
sheet_names = grassland_data.sheet_names

# Read data from all sheets into a dictionary
sheets_dict = {sheet: grassland_data.parse(sheet) for sheet in sheet_names}
```

In [9]:
```python
print(sheet_names)
# print(sheets_dict)
```

```
['ANTI', 'CATO', 'CHOH', 'GWMP', 'HAFE', 'MANA', 'MONO', 'NACE', 'PRWI', 'ROCR', 'WOTR']
```

In [10]:
```python
# Example: Convert sheets_dict to a single DataFrame and stored combined dataframe in grasslan
grassland = pd.concat(
    [df.assign(Sheet=sheet_name) for sheet_name, df in sheets_dict.items()],
    ignore_index=True
)
```

```
C:\Users\ravit\AppData\Local\Temp\ipykernel_2000\326147973.py:2: FutureWarning: The behavior o
f DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, thi
s will no longer exclude empty or all-NA columns when determining the result dtypes. To retain
the old behavior, exclude the relevant entries before the concat operation.
  grassland = pd.concat(
```

In [11]:
```python
grassland.head(3)
```

Out[11]:

	Admin_Unit_Code	Sub_Unit_Code	Plot_Name	Location_Type	Year	Date	Start_Time	End_Time
0	ANTI	NaN	ANTI-0054	Grassland	2018	2018-05-22	05:35:00	05:45:00
1	ANTI	NaN	ANTI-0054	Grassland	2018	2018-05-22	05:35:00	05:45:00
2	ANTI	NaN	ANTI-0054	Grassland	2018	2018-05-22	05:35:00	05:45:00

3 rows × 30 columns

In [12]:
```python
print(grassland.columns)
grassland.shape
```

```
Index(['Admin_Unit_Code', 'Sub_Unit_Code', 'Plot_Name', 'Location_Type',
       'Year', 'Date', 'Start_Time', 'End_Time', 'Observer', 'Visit',
       'Interval_Length', 'ID_Method', 'Distance', 'Flyover_Observed', 'Sex',
       'Common_Name', 'Scientific_Name', 'AcceptedTSN', 'TaxonCode',
       'AOU_Code', 'PIF_Watchlist_Status', 'Regional_Stewardship_Status',
       'Temperature', 'Humidity', 'Sky', 'Wind', 'Disturbance',
       'Previously_Obs', 'Initial_Three_Min_Cnt', 'Sheet'],
      dtype='object')
```

Out[12]:
```
(8531, 30)
```

So, I have successfuly loaded both files with multiple sheets !

Now, I'm going to add rows of grassland to end of forest dataframe, returning a new dataframe object data and for doing this I have used `concat()` function.

In [13]:
```python
print(forest.shape)
print(forest.columns)
print('-------')
print(grassland.shape)
print(grassland.columns)
```

```
(8546, 30)
Index(['Admin_Unit_Code', 'Sub_Unit_Code', 'Site_Name', 'Plot_Name',
       'Location_Type', 'Year', 'Date', 'Start_Time', 'End_Time', 'Observer',
       'Visit', 'Interval_Length', 'ID_Method', 'Distance', 'Flyover_Observed',
       'Sex', 'Common_Name', 'Scientific_Name', 'AcceptedTSN', 'NPSTaxonCode',
       'AOU_Code', 'PIF_Watchlist_Status', 'Regional_Stewardship_Status',
       'Temperature', 'Humidity', 'Sky', 'Wind', 'Disturbance',
       'Initial_Three_Min_Cnt', 'Sheet'],
      dtype='object')
-------
(8531, 30)
Index(['Admin_Unit_Code', 'Sub_Unit_Code', 'Plot_Name', 'Location_Type',
       'Year', 'Date', 'Start_Time', 'End_Time', 'Observer', 'Visit',
       'Interval_Length', 'ID_Method', 'Distance', 'Flyover_Observed', 'Sex',
       'Common_Name', 'Scientific_Name', 'AcceptedTSN', 'TaxonCode',
       'AOU_Code', 'PIF_Watchlist_Status', 'Regional_Stewardship_Status',
       'Temperature', 'Humidity', 'Sky', 'Wind', 'Disturbance',
       'Previously_Obs', 'Initial_Three_Min_Cnt', 'Sheet'],
      dtype='object')
```

In [14]:
```python
# Appending/Concatinating
bird = pd.concat([forest, grassland], ignore_index=True)
bird
```

Out[14]:

	Admin_Unit_Code	Sub_Unit_Code	Site_Name	Plot_Name	Location_Type	Year	Date	Start_Ti
0	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:1
1	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:1
2	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:1
3	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:1
4	ANTI	NaN	ANTI 1	ANTI-0036	Forest	2018	2018-05-22	06:1
...	...	...	...	...	...	...	...	
17072	MONO	NaN	NaN	MONO-0089	Grassland	2018	2018-05-10	06:3
17073	MONO	NaN	NaN	MONO-0089	Grassland	2018	2018-05-10	06:3
17074	MONO	NaN	NaN	MONO-0089	Grassland	2018	2018-05-10	06:3
17075	MONO	NaN	NaN	MONO-0089	Grassland	2018	2018-05-10	06:3
17076	MONO	NaN	NaN	MONO-0089	Grassland	2018	2018-05-10	06:3

17077 rows × 32 columns

In [15]:
```
print(bird.shape)
print(bird.columns)
```

```
(17077, 32)
Index(['Admin_Unit_Code', 'Sub_Unit_Code', 'Site_Name', 'Plot_Name',
       'Location_Type', 'Year', 'Date', 'Start_Time', 'End_Time', 'Observer',
       'Visit', 'Interval_Length', 'ID_Method', 'Distance', 'Flyover_Observed',
       'Sex', 'Common_Name', 'Scientific_Name', 'AcceptedTSN', 'NPSTaxonCode',
       'AOU_Code', 'PIF_Watchlist_Status', 'Regional_Stewardship_Status',
       'Temperature', 'Humidity', 'Sky', 'Wind', 'Disturbance',
       'Initial_Three_Min_Cnt', 'Sheet', 'TaxonCode', 'Previously_Obs'],
      dtype='object')
```

Here, we can observed after the concatination no. of columns has increased by 2. This is because `Site_Name` column is not available in the `grassland` dataframe so this column appended till the end with `NaN` value and at the same `Previously_Obs` column is not available in the `forest` dataframe so it's appended with `NaN` value.

Next -

Data Cleaning and Processing

Droping & Renaming Columns

I'm going to drop unnecessary columns which are not important while analysis
Rename Column - I want to change column name from `Location_Type` to `habitat_type`

In [16]:
```
# Droping Column
bird = bird.drop(columns=['Plot_Name'])
bird = bird.drop(columns=['Sub_Unit_Code'])
bird = bird.drop(columns=['Site_Name'])
bird = bird.drop(columns=['Common_Name'])
bird = bird.drop(columns=['AcceptedTSN'])
bird = bird.drop(columns=['NPSTaxonCode'])
bird = bird.drop(columns=['Initial_Three_Min_Cnt'])
bird = bird.drop(columns=['Sheet'])
bird = bird.drop(columns=['TaxonCode'])
bird = bird.drop(columns=['Previously_Obs'])

# Renaming column
bird = bird.rename(columns={'Location_Type':'Habitat_type'})
```

In [17]:
```
bird
```

Out[17]:

	Admin_Unit_Code	Habitat_type	Year	Date	Start_Time	End_Time	Observer	Visit	Interval_L
0	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2
1	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2
2	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 -
3	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 -
4	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 -
...	...	...	...	...	...	...	...	...	
17072	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	2.5 -
17073	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	2.5 -
17074	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	2.5 -
17075	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	2.5 -
17076	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	2.5 -

17077 rows × 22 columns

In [18]:
```python
print(bird.shape)
```

```
(17077, 22)
```

In [19]:
```python
bird.columns
```

Out[19]:
```
Index(['Admin_Unit_Code', 'Habitat_type', 'Year', 'Date', 'Start_Time',
       'End_Time', 'Observer', 'Visit', 'Interval_Length', 'ID_Method',
       'Distance', 'Flyover_Observed', 'Sex', 'Scientific_Name', 'AOU_Code',
       'PIF_Watchlist_Status', 'Regional_Stewardship_Status', 'Temperature',
       'Humidity', 'Sky', 'Wind', 'Disturbance'],
      dtype='object')
```

In [20]:
```python
bird.info()
# info() - info() function provide a concise summary of dataframe. it is useful to understand
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17077 entries, 0 to 17076
Data columns (total 22 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Admin_Unit_Code              17077 non-null  object
 1   Habitat_type                 17077 non-null  object
 2   Year                         17077 non-null  object
 3   Date                         17077 non-null  datetime64[ns]
 4   Start_Time                   17077 non-null  object
 5   End_Time                     17077 non-null  object
 6   Observer                     17077 non-null  object
 7   Visit                        17077 non-null  object
 8   Interval_Length              17077 non-null  object
 9   ID_Method                    17075 non-null  object
 10  Distance                     15591 non-null  object
 11  Flyover_Observed             17077 non-null  object
 12  Sex                          11894 non-null  object
 13  Scientific_Name              17077 non-null  object
 14  AOU_Code                     17077 non-null  object
 15  PIF_Watchlist_Status         17077 non-null  object
 16  Regional_Stewardship_Status  17077 non-null  object
 17  Temperature                  17077 non-null  float64
 18  Humidity                     17077 non-null  float64
 19  Sky                          17077 non-null  object
 20  Wind                         17077 non-null  object
 21  Disturbance                  17077 non-null  object
dtypes: datetime64[ns](1), float64(2), object(19)
memory usage: 2.9+ MB
```

Correct Formatting/New Column creation -

In [21]:
```python
bird['Month'] = bird['Date'].dt.month                # Here, I have created new 'month' column
bird['Humidity'] = round(bird['Humidity'],3)         # I have rounded the value of humidity an
bird['Temperature'] = round(bird['Temperature'],3)

# Here, I'm creating new 'season' column and define season by month as per indian climate; th
bird['Season'] = bird['Month'].apply(lambda x: (
    'Winter' if x in [12, 1, 2] else
    'Summer' if x in [3, 4, 5] else
    'Monsoon' if x in [6, 7, 8] else
    'Post-Monsoon'))
```

In [22]:
```python
bird.head()
```

Out[22]:

	Admin_Unit_Code	Habitat_type	Year	Date	Start_Time	End_Time	Observer	Visit	Interval_Lengt
0	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2.5 mi
1	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2.5 mi
2	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 - 5 mi
3	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 - 5 mi
4	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 - 5 mi

5 rows × 24 columns

In [23]:

```
bird.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17077 entries, 0 to 17076
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Admin_Unit_Code             17077 non-null  object
 1   Habitat_type                17077 non-null  object
 2   Year                        17077 non-null  object
 3   Date                        17077 non-null  datetime64[ns]
 4   Start_Time                  17077 non-null  object
 5   End_Time                    17077 non-null  object
 6   Observer                    17077 non-null  object
 7   Visit                       17077 non-null  object
 8   Interval_Length             17077 non-null  object
 9   ID_Method                   17075 non-null  object
 10  Distance                    15591 non-null  object
 11  Flyover_Observed            17077 non-null  object
 12  Sex                         11894 non-null  object
 13  Scientific_Name             17077 non-null  object
 14  AOU_Code                    17077 non-null  object
 15  PIF_Watchlist_Status        17077 non-null  object
 16  Regional_Stewardship_Status 17077 non-null  object
 17  Temperature                 17077 non-null  float64
 18  Humidity                    17077 non-null  float64
 19  Sky                         17077 non-null  object
 20  Wind                        17077 non-null  object
 21  Disturbance                 17077 non-null  object
 22  Month                       17077 non-null  int32
 23  Season                      17077 non-null  object
dtypes: datetime64[ns](1), float64(2), int32(1), object(20)
memory usage: 3.1+ MB
```

Check/Drop duplicates -

In [24]:
```
# Checking duplicates
bird.duplicated().sum()
```

Out[24]:
```
1713
```

In [25]:
```
# Droping duplicates
bird.drop_duplicates(inplace=True)
# The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it
```

In [26]:
```
bird.shape
# Here, we can see successfully droped duplicate values
```

Out[26]:
```
(15364, 24)
```

Check/Handle Null vaules

In [27]:
```
# Checking null values
bird.isnull().sum()
# Below we can see only 3 columns have null values
```

Out[27]:
```
Admin_Unit_Code                 0
Habitat_type                    0
Year                            0
Date                            0
Start_Time                      0
End_Time                        0
Observer                        0
Visit                           0
Interval_Length                 0
ID_Method                       2
Distance                      689
Flyover_Observed                0
Sex                          5177
Scientific_Name                 0
AOU_Code                        0
PIF_Watchlist_Status            0
Regional_Stewardship_Status     0
Temperature                     0
Humidity                        0
Sky                             0
Wind                            0
Disturbance                     0
Month                           0
Season                          0
dtype: int64
```

In [28]:
```
# Handle null values by droping or filling
# 1. filling null values with 'NA'
bird["Sex"] = bird["Sex"].fillna("NA")
```

In [29]:
```
# 2. Drop rows with null values
bird = bird.dropna(subset=['ID_Method', 'Distance'])
```

In [30]:
```
bird.isnull().sum()
```

Out[30]:
```
Admin_Unit_Code                0
Habitat_type                   0
Year                           0
Date                           0
Start_Time                     0
End_Time                       0
Observer                       0
Visit                          0
Interval_Length                0
ID_Method                      0
Distance                       0
Flyover_Observed               0
Sex                            0
Scientific_Name                0
AOU_Code                       0
PIF_Watchlist_Status           0
Regional_Stewardship_Status    0
Temperature                    0
Humidity                       0
Sky                            0
Wind                           0
Disturbance                    0
Month                          0
Season                         0
dtype: int64
```

In [31]:
```
bird.head()
```

Out[31]:

	Admin_Unit_Code	Habitat_type	Year	Date	Start_Time	End_Time	Observer	Visit	Interval_Lengt
0	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2.5 mi
1	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2.5 mi
2	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 - 5 mi
3	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 - 5 mi
4	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 - 5 mi

5 rows × 24 columns

`Wind` column analysis

So, contains categorical descriptions like:

- "Calm (< 1 mph) smoke rises vertically"
- "Light air (1-3 mph) leaves rustle"
- "Light breeze (4-7 mph) small branches move"

Let's Understand the distribution of wind conditions -

Clean or Simplify Wind Categories

In [32]:

```
print(bird['Wind'].unique())    # To see the category of wind and speed of wind
```

```
['Calm (< 1 mph) smoke rises vertically'
 'Light air movement (1-3 mph) smoke drifts'
 'Light breeze (4-7 mph) wind felt on face'
 'Gentle breeze (8-12 mph), leaves in motion']
```

In [33]:

```
def simplify_wind(wind):
    if pd.isna(wind):
        return 'Unknown'
    elif 'Calm' in wind:
        return 'Calm'
    elif 'Light air' in wind:
        return 'Light Air'
    elif 'Light breeze' in wind:
        return 'Light Breeze'
    elif 'Gentle breeze' in wind:
```

```
        return 'Gentle Breeze'
    else:
        return 'Other'

bird['Wind_Category'] = bird['Wind'].apply(simplify_wind)
```

```
C:\Users\ravit\AppData\Local\Temp\ipykernel_2000\4170358401.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  bird['Wind_Category'] = bird['Wind'].apply(simplify_wind)
```

In [34]:
```
bird['Wind_Category'].unique()
```

Out[34]:
```
array(['Calm', 'Light Air', 'Light Breeze', 'Gentle Breeze'], dtype=object)
```

wind intensity has a progressive effect on bird activity:

In [35]:
```
wind_order = {
    'Calm': '<1 mph',
    'Light Air': '1-3 mph',
    'Light Breeze': '4-7 mph',
    'Gentle Breeze': '8-12 mph'
}

bird['Wind_Speed'] = bird['Wind_Category'].map(wind_order)
```

```
C:\Users\ravit\AppData\Local\Temp\ipykernel_2000\1134770776.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  bird['Wind_Speed'] = bird['Wind_Category'].map(wind_order)
```

Above I have done `wind` column analysis and I have created two new columns `Wind_Category` and `Wind_Speed`.

In [36]:
```
bird
```

Out[36]:

	Admin_Unit_Code	Habitat_type	Year	Date	Start_Time	End_Time	Observer	Visit	Interval_L
0	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2
1	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2
2	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 -
3	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 -
4	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	2.5 -
...	...	...	...	...	...	...	...	...	
17056	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	7.5 -
17057	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	7.5 -
17060	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	0-2
17062	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	0-2
17063	MONO	Grassland	2018	2018-05-10	06:35:00	06:46:00	Brian Swimelar	1	0-2

14674 rows × 26 columns

In [37]:
```
bird.reset_index(drop=True, inplace=True)
bird.head(2)
# Here, reseting the index because after removing duplicates records/rows indexing had deteri
```

Out[37]:

	Admin_Unit_Code	Habitat_type	Year	Date	Start_Time	End_Time	Observer	Visit	Interval_Lengt
0	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2.5 mi
1	ANTI	Forest	2018	2018-05-22	06:19:00	06:29:00	Elizabeth Oswald	1	0-2.5 mi

2 rows × 26 columns

In [38]:
```
bird.shape
```

Out[38]:
```
(14674, 26)
```

In [39]:
```
bird.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14674 entries, 0 to 14673
Data columns (total 26 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Admin_Unit_Code               14674 non-null  object
 1   Habitat_type                  14674 non-null  object
 2   Year                          14674 non-null  object
 3   Date                          14674 non-null  datetime64[ns]
 4   Start_Time                    14674 non-null  object
 5   End_Time                      14674 non-null  object
 6   Observer                      14674 non-null  object
 7   Visit                         14674 non-null  object
 8   Interval_Length               14674 non-null  object
 9   ID_Method                     14674 non-null  object
 10  Distance                      14674 non-null  object
 11  Flyover_Observed              14674 non-null  object
 12  Sex                           14674 non-null  object
 13  Scientific_Name               14674 non-null  object
 14  AOU_Code                      14674 non-null  object
 15  PIF_Watchlist_Status          14674 non-null  object
 16  Regional_Stewardship_Status   14674 non-null  object
 17  Temperature                   14674 non-null  float64
 18  Humidity                      14674 non-null  float64
 19  Sky                           14674 non-null  object
 20  Wind                          14674 non-null  object
 21  Disturbance                   14674 non-null  object
 22  Month                         14674 non-null  int32
 23  Season                        14674 non-null  object
 24  Wind_Category                 14674 non-null  object
 25  Wind_Speed                    14674 non-null  object
dtypes: datetime64[ns](1), float64(2), int32(1), object(22)
memory usage: 2.9+ MB
```

Here, I have saved cleaned data for future analysis

```
In [40]: # bird.to_excel("bird_observation_excel_data.xlsx", index=False)
         # bird.to_csv("bird_observation_csv_data.csv", index=False)
```

Now data is ready for visualization ---

📊 Data Visualization - Exploratory Data Analysis

1. Habitat type distribution -

```
In [41]: fig = px.bar(
             bird,
             x="Habitat_type",
             color="Habitat_type",
             facet_col='Sex',
             opacity=1,
             color_discrete_sequence=px.colors.qualitative.Bold  # strong, solid colors
         )

         fig.update_traces(marker=dict(line=dict(width=0)))  # remove white outlines
         fig.update_layout(bargap=0.1)  # optional: adjust spacing
         fig.show()
```

Insights

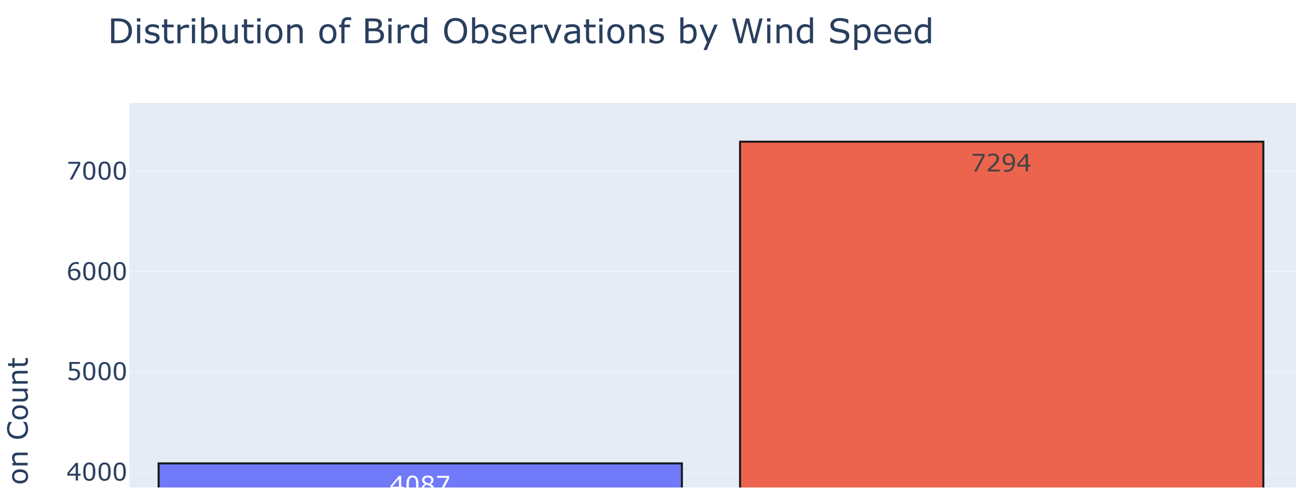
The faceted bar chart reveals the distribution of bird observations across habitat types (Forest vs. Grassland) for each sex category (Male, Female, Undetermined, NA).

- Undetermined and NA sexes are recorded in both habitats, with a slightly higher count in Forest.
- Males appear more frequently in Grassland, while Females are scarce and mostly observed in Grassland.

- The dominance of Undetermined and NA categories suggests that sex identification was often not determined during observations, potentially limiting sex-based ecological analysis.

2. Bird Observe vs Wind speed -

In [42]:
```
fig = px.histogram(
    bird,
    x="Wind_Speed",
    color="Wind_Speed",  # color bars
    opacity=0.9,
    text_auto=True,      # show counts on bars
    hover_data=["Wind_Speed"]  # additional columns can be added here
)

fig.update_traces(
    marker_line_width=1,
    marker_line_color="black"
)

fig.update_layout(
    title="Distribution of Bird Observations by Wind Speed",
    xaxis_title="Wind Speed",
    yaxis_title="Observation Count",
    bargap=0.1
)

fig.show()
```



Insights

The histogram shows how bird observations are distributed across different Wind_Speed categories.

- Most observations occur at low wind speeds (e.g., <1 mph, 1-3 mph), suggesting calmer conditions are more common during surveys or more favorable for bird detection.
- Observations drop noticeably as wind speed increases, possibly due to reduced bird activity or observer detection challenges in windy conditions.

3. Birds Count by Observation site -

In [43]:
```
fig = px.bar(
    bird,
    x="Admin_Unit_Code",
    color="Admin_Unit_Code",
    opacity=1,
    color_discrete_sequence=px.colors.qualitative.Bold  # strong, solid colors
)

fig.update_traces(marker=dict(line=dict(width=0)))  # remove white outlines
fig.update_layout(bargap=0.1)  # optional: adjust spacing
fig.show()
```

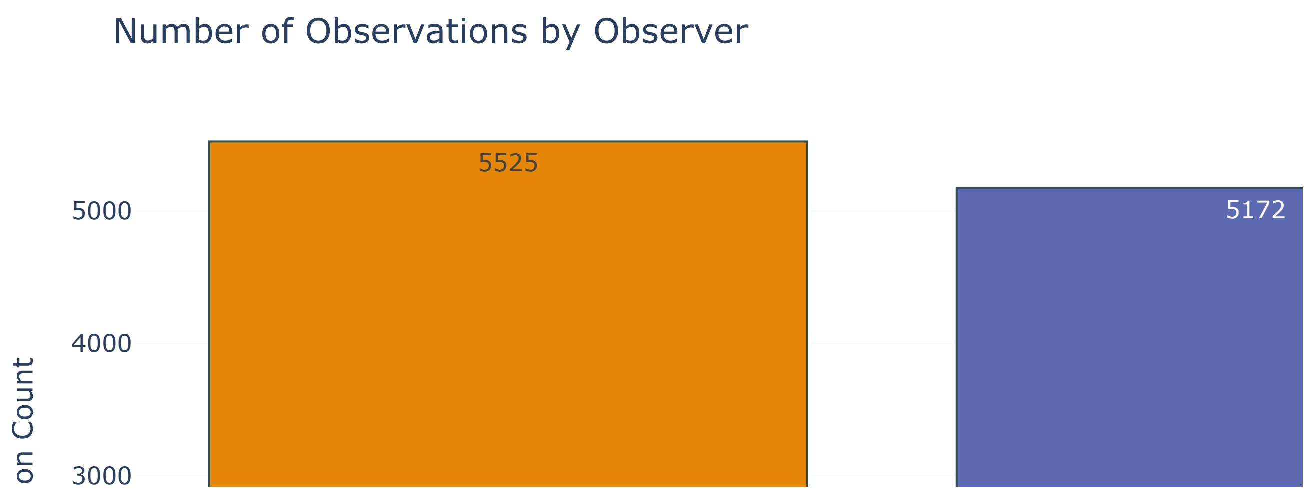
Insights

The bar chart clearly displays the distribution of bird observations across different administrative unit codes.

- Using strong, opaque colors makes each category visually distinct, helping to quickly identify areas with higher or lower observation counts.
- Locations like ANTI and PRWI stand out with notably higher counts, while sites such as WOTR and ROCR have the lowest, suggesting differences in either bird population density, observation effort, or both.

4. Count observations per observer -

In [44]:
```python
obs_counts = bird['Observer'].value_counts().reset_index()
obs_counts.columns = ['Observer', 'Count']

# Create interactive bar chart
fig = px.bar(
    obs_counts,
    x="Observer",
    y="Count",
    color="Observer",
    text="Count",
    color_discrete_sequence=px.colors.qualitative.Vivid
)

# Add interactive features
fig.update_traces(
    hovertemplate="<b>Observer:</b> %{x}<br><b>Observations:</b> %{y}",
    marker=dict(line=dict(width=1, color='DarkSlateGrey'))
)

fig.update_layout(
    title="Number of Observations by Observer",
    xaxis_title="Observer",
    yaxis_title="Observation Count",
    hovermode="closest",
    template="plotly_white"
)

fig.show()
```



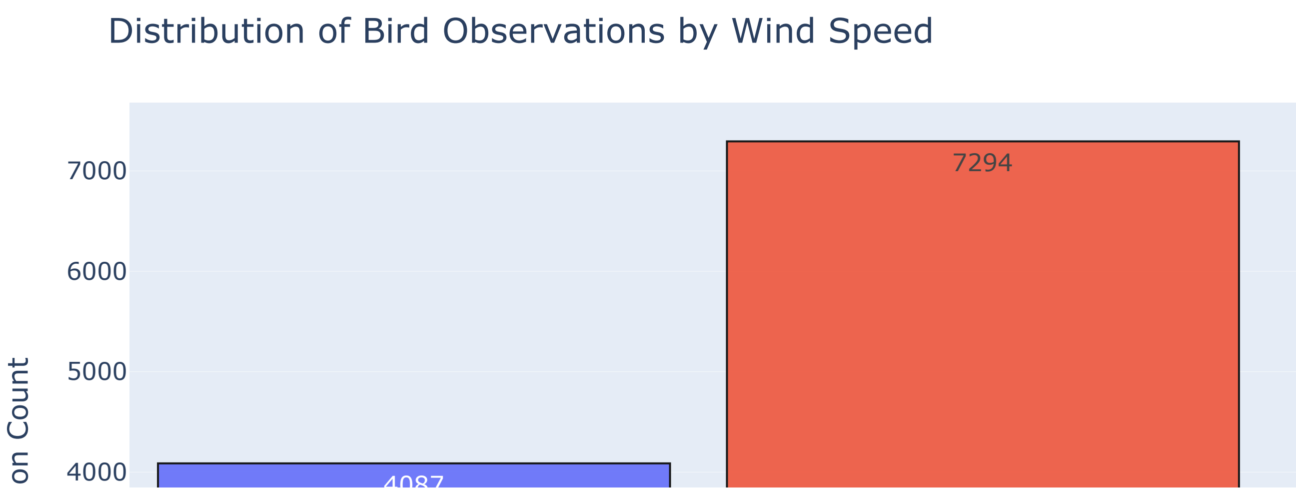
Ingights

- Elizabeth Oswald recorded the highest number of observations, slightly ahead of Kimberly Serno.
- Kimberly Serno follows closely behind, suggesting both contribute almost equally to data collection.
- Brian Swimelar has noticeably fewer observations than the other two, roughly 25–30% lower than Elizabeth.
- The observation effort is not evenly distributed — two observers are responsible for the majority of data.

5. Birds observe in different wind category -

In [45]:

```
fig = px.histogram(
    bird,
    x="Wind_Category",
    color="Wind_Category",  # color bars
    opacity=0.9,
    text_auto=True,       # show counts on bars
    hover_data=["Wind_Speed"]  # additional columns can be added here
)

fig.update_traces(
    marker_line_width=1,
    marker_line_color="black"
)

fig.update_layout(
    title="Distribution of Bird Observations by Wind Speed",
    xaxis_title="Wind Speed",
    yaxis_title="Observation Count",
    bargap=0.1
)

fig.show()
```



Insights

The histogram shows how bird observations are distributed across different wind speed categories.

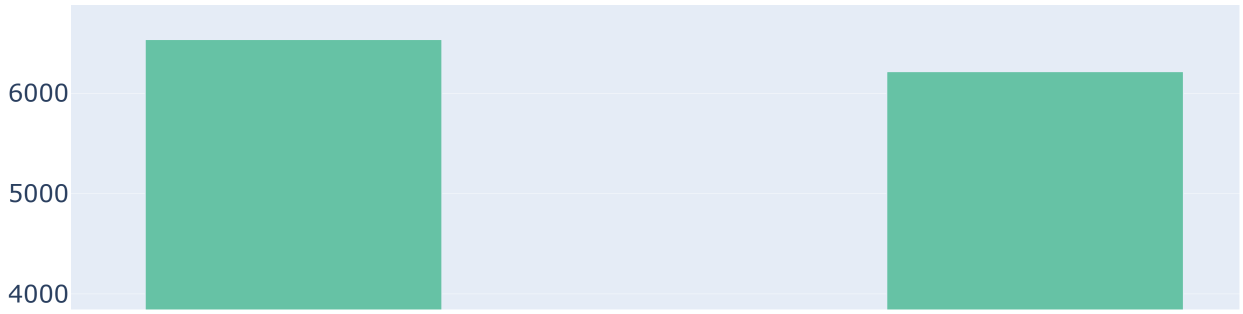
- Most observations occur in moderate wind conditions, with fewer sightings during extreme low or high winds.
- This suggests that bird activity—and possibly observer effort—is more common when wind speeds are comfortable.
- Extremely high or calm wind conditions appear less favorable for bird detection, possibly due to changes in bird behavior or observation challenges.

6. Species diversity per visit -

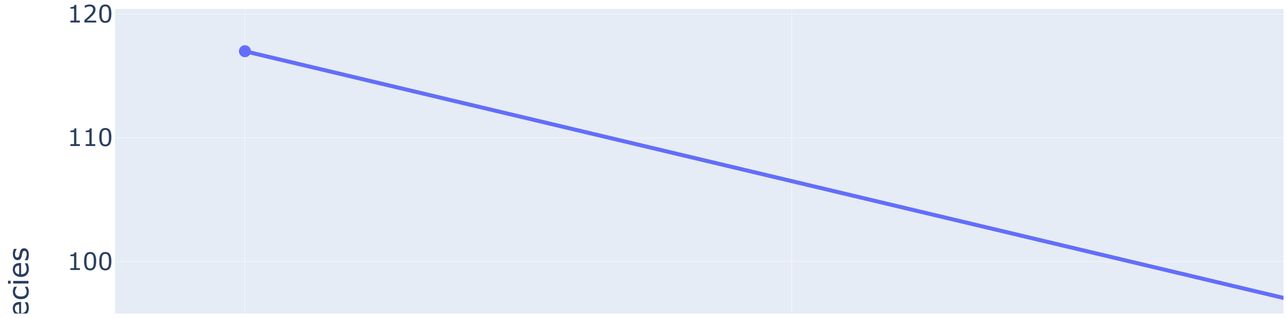
In [46]:

```
visit_stats = bird.groupby("Visit").agg(
    total_observations=("Scientific_Name", "count"),
    species_diversity=("Scientific_Name", pd.Series.nunique)
).reset_index()

# Bar chart: total observations vs species diversity
fig = px.bar(
    visit_stats,
    x="Visit",
    y=["total_observations", "species_diversity"],
    barmode="group",
    title="Effect of Repeated Visits on Species Count and Diversity",
    labels={"value": "Count", "variable": "Metric"},
    color_discrete_sequence=px.colors.qualitative.Set2
)

fig.show()

# Optional: line chart for diversity trend
fig_line = px.line(
    visit_stats,
    x="Visit",
    y="species_diversity",
    markers=True,
    title="Species Diversity by Visit Number",
    labels={"species_diversity": "Unique Species"}
)
fig_line.show()
```

Effect of Repeated Visits on Species Count and Diversity



Species Diversity by Visit Number



Insights

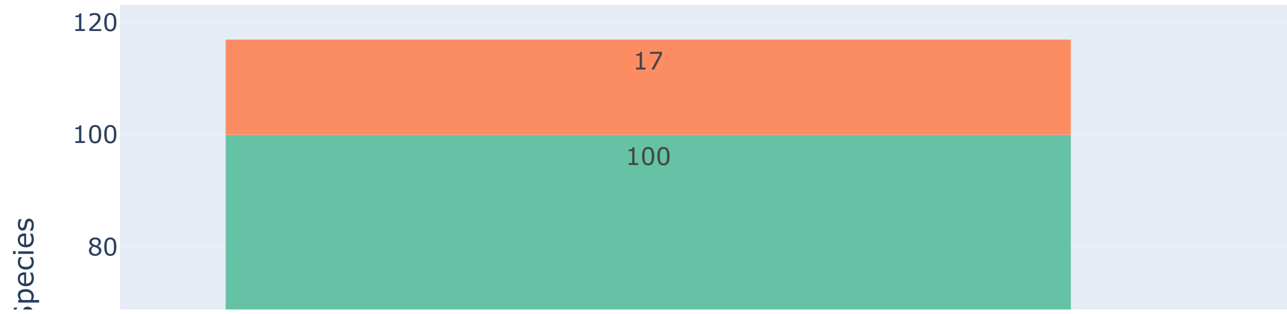
The analysis shows how repeated visits influence both total bird observations and species diversity.

- In most cases, total observations tend to be higher during earlier visits, suggesting that fresh survey areas yield more sightings initially.
- Species diversity may peak at specific visit numbers, then stabilize or decline, indicating that repeated visits might detect fewer new species over time.
- A consistent diversity trend across visits could suggest that the surveyed habitat has a stable bird community, while fluctuating trends may indicate migratory activity or seasonal variation.

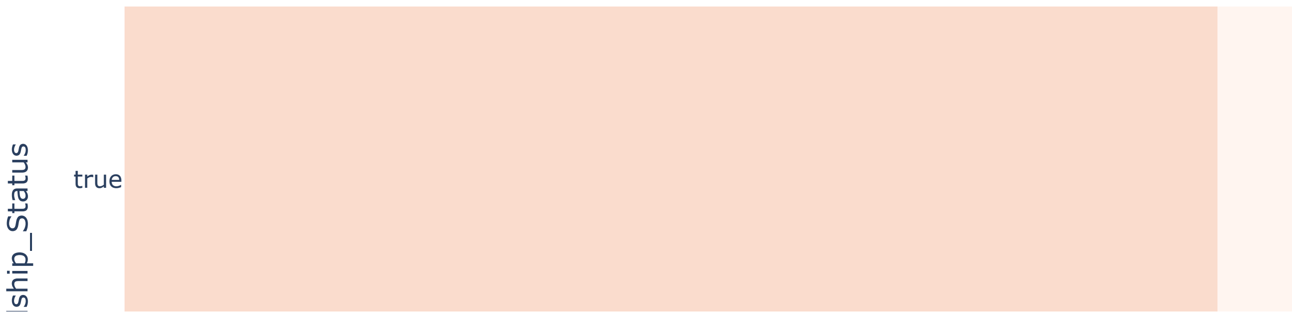
7. Group by both status columns -

In [47]:
```
watchlist_stats = bird.groupby(
    ["PIF_Watchlist_Status", "Regional_Stewardship_Status"]
)["Scientific_Name"].nunique().reset_index()

watchlist_stats.rename(columns={"Scientific_Name": "Species_Count"}, inplace=True)

# Stacked bar chart
fig = px.bar(
    watchlist_stats,
    x="PIF_Watchlist_Status",
    y="Species_Count",
    color="Regional_Stewardship_Status",
    title="Watchlist and Regional Stewardship Trends",
    labels={"Species_Count": "Number of Species"},
    color_discrete_sequence=px.colors.qualitative.Set2,
    text="Species_Count"
)

fig.update_layout(barmode="stack")
fig.show()

# Optional: Heatmap for visual clarity
fig_heat = px.density_heatmap(
    watchlist_stats,
    x="PIF_Watchlist_Status",
    y="Regional_Stewardship_Status",
    z="Species_Count",
    color_continuous_scale="Reds",
    title="Heatmap: At-Risk Species by Watchlist & Regional Status"
)
fig_heat.show()
```

Watchlist and Regional Stewardship Trends



Heatmap: At-Risk Species by Watchlist & Regional Status



Insights

The analysis reveals clear patterns in at-risk bird species based on the PIF Watchlist Status and Regional

Stewardship Status.

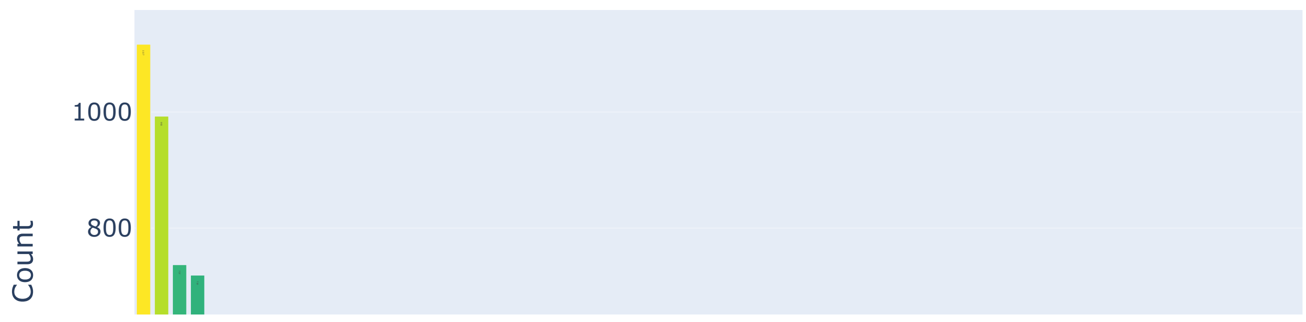
- Certain watchlist categories (e.g., “High Concern”) are strongly associated with higher species counts under high regional stewardship priorities, suggesting urgent local conservation needs.
- Some watchlist statuses show low species counts but still align with high stewardship importance, indicating that even a few species in these categories may be critical to protect.
- The heatmap highlights intersections where both watchlist concern and regional stewardship priority are high, helping conservationists target resources effectively.

8. Group by AOU_Code for counts -

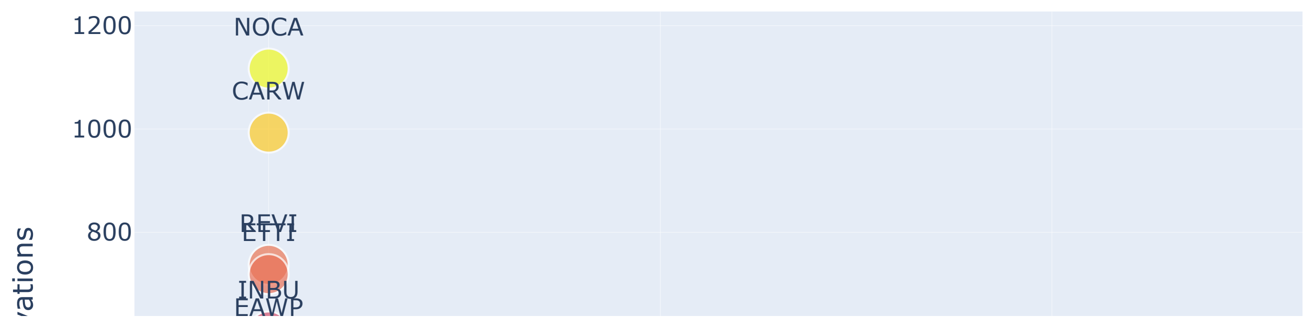
In [48]:

```
aou_stats = bird.groupby("AOU_Code").agg(
    total_observations=("Scientific_Name", "count"),
    species_diversity=("Scientific_Name", pd.Series.nunique)
).reset_index()

# Sort for better visualization
aou_stats = aou_stats.sort_values("total_observations", ascending=False)

# Bar chart for total observations
fig = px.bar(
    aou_stats,
    x="AOU_Code",
    y="total_observations",
    text="total_observations",
    title="Distribution of Observations by AOU Code",
    labels={"total_observations": "Observation Count", "AOU_Code": "AOU Code"},
    color="total_observations",
    color_continuous_scale="Viridis"
)
fig.show()

# Optional: Scatter to compare abundance vs diversity
fig_scatter = px.scatter(
    aou_stats,
    x="species_diversity",
    y="total_observations",
    text="AOU_Code",
    title="AOU Code: Species Diversity vs Total Observations",
    labels={
        "species_diversity": "Unique Species per AOU Code",
        "total_observations": "Total Observations"
    },
    color="total_observations",
    color_continuous_scale="Plasma",
    size="species_diversity"
)
fig_scatter.update_traces(textposition="top center")
fig_scatter.show()
```

Distribution of Observations by AOU Code



AOU Code: Species Diversity vs Total Observations



Insights

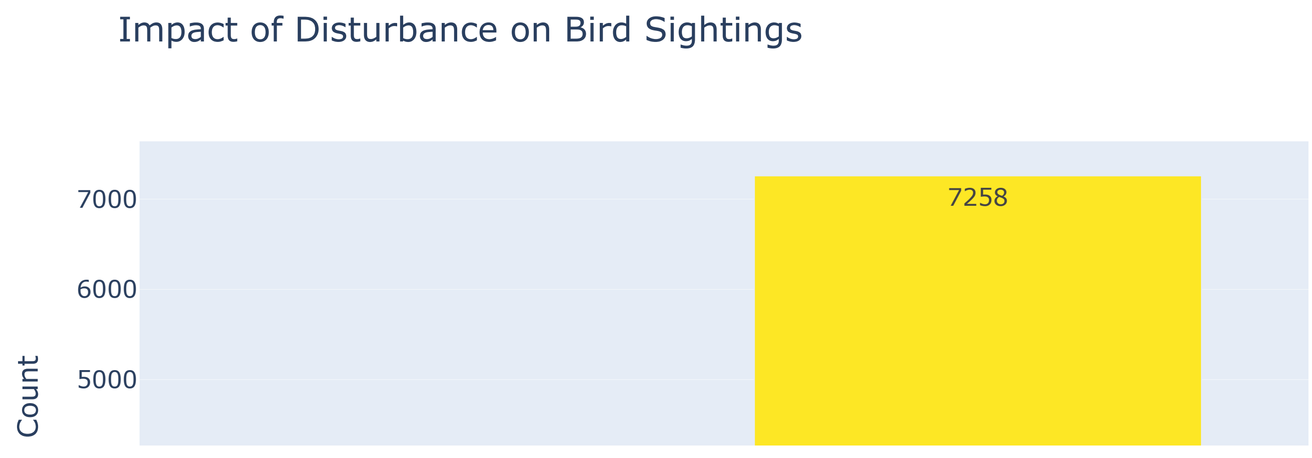
The scatter plot comparison helps identify priority monitoring areas — where diversity is high but total numbers are low, signaling possible conservation concern.

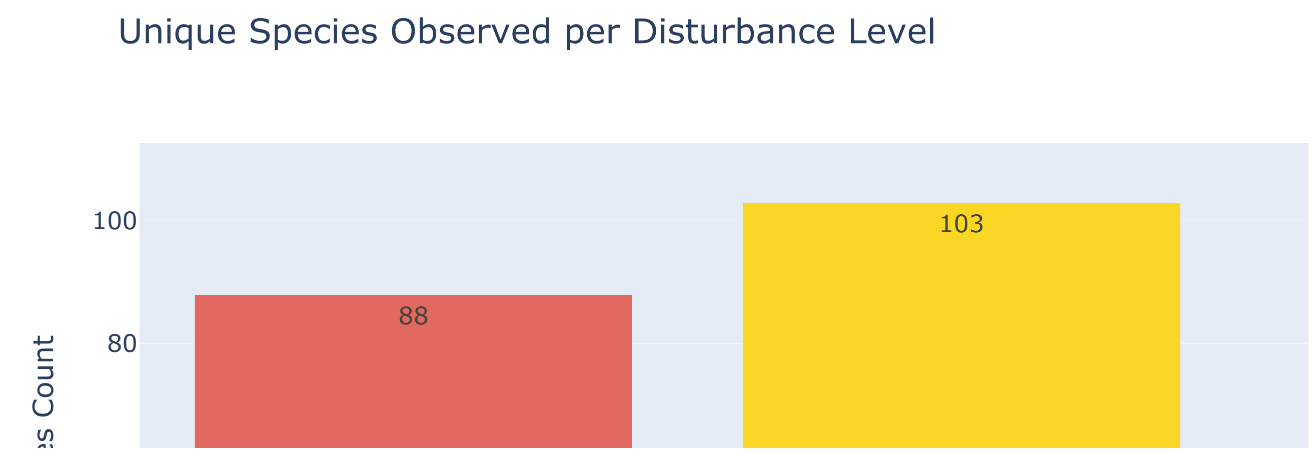
- A few AOU codes dominate in total observations, suggesting either abundant species or species that are easier to detect during surveys.
- Codes with high diversity and high observation counts represent regions or species groups with both richness and abundance — potential biodiversity hotspots.
- Some AOU codes show high diversity but low total counts, indicating the presence of many unique species that are individually rare.

9. Group by disturbance category -

In [49]:

```python
disturbance_stats = bird.groupby("Disturbance").agg(
    total_observations=("Scientific_Name", "count"),
    unique_species=("Scientific_Name", pd.Series.nunique)
).reset_index()

# Bar chart: total observations per disturbance level
fig_bar = px.bar(
    disturbance_stats,
    x="Disturbance",
    y="total_observations",
    text="total_observations",
    title="Impact of Disturbance on Bird Sightings",
    labels={"total_observations": "Observation Count", "Disturbance": "Disturbance Level"},
    color="total_observations",
    color_continuous_scale="Viridis"
)
fig_bar.show()

# Optional: unique species per disturbance level
fig_species = px.bar(
    disturbance_stats,
    x="Disturbance",
    y="unique_species",
    text="unique_species",
    title="Unique Species Observed per Disturbance Level",
    labels={"unique_species": "Unique Species Count", "Disturbance": "Disturbance Level"},
    color="unique_species",
    color_continuous_scale="Plasma"
)
fig_species.show()
```





Insights

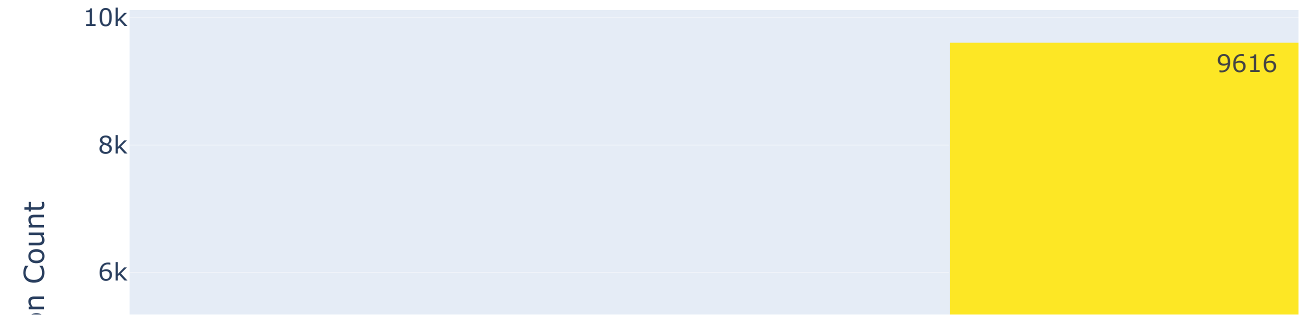
The disturbance effect analysis reveals clear relationships between habitat disturbance levels and bird observations:

- Moderate disturbance levels often show relatively high observation counts, which could indicate that certain bird species tolerate or even exploit partially disturbed habitats.
- High disturbance areas generally see lower species diversity and fewer total observations, suggesting that excessive human or environmental disruption reduces habitat suitability.
- Low disturbance zones tend to support more unique species, highlighting their importance for conservation and biodiversity protection.

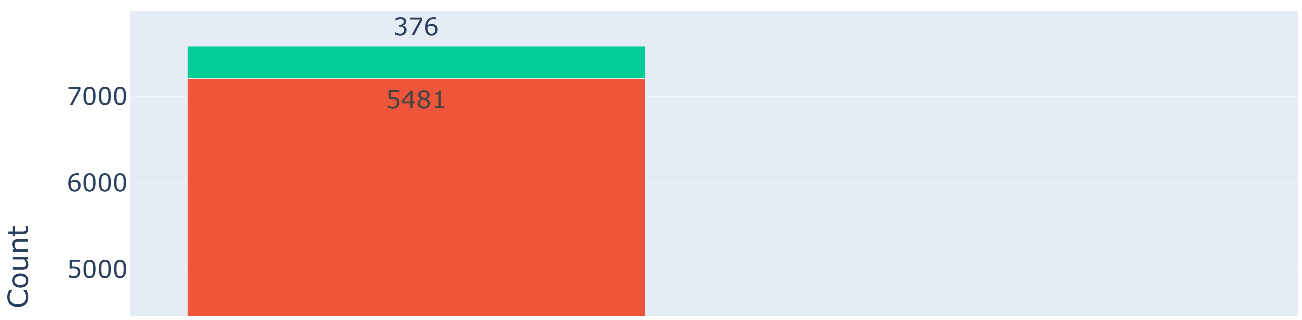
10. Count occurrences of each activity type -

In [50]:
```
activity_counts = bird.groupby("ID_Method").size().reset_index(name="count")

# Bar chart for most common activities
fig_bar = px.bar(
    activity_counts,
    x="ID_Method",
    y="count",
    text="count",
    title="Most Common Activity Types",
    labels={"count": "Observation Count", "ID_Method": "Activity Type"},
    color="count",
    color_continuous_scale="Viridis"
)
fig_bar.update_xaxes(tickangle=45)
fig_bar.show()

# Activity by interval length
activity_interval = bird.groupby(["Interval_Length", "ID_Method"]).size().reset_index(name="c

# Stacked bar chart
fig_stack = px.bar(
    activity_interval,
    x="Interval_Length",
    y="count",
    color="ID_Method",
    title="Activity Types by Interval Length",
    labels={"count": "Observation Count", "Interval_Length": "Interval Length (min)", "ID_Met
    text="count"
)
fig_stack.show()
```

Most Common Activity Types



Activity Types by Interval Length



Insights

The activity pattern analysis highlights both the most common bird observation methods and how they vary by survey duration:

- Singing is the most frequently recorded activity type, suggesting it is a primary indicator used by observers to detect species presence
- Calling and visual sightings also contribute significantly but are less dominant compared to singing-based detections.
- Shorter intervals (e.g., 1–3 minutes) tend to capture quick, highly detectable activities like singing and calling.
- Longer intervals increase the likelihood of recording a wider variety of activities, including less frequent behaviors.

In [ ]:

In [ ]:

In [ ]:

In [ ]: