# Scientific Programming in Python

**Winter Semester 2024/25**

**Instructor:** Krumnack, Ulf, Ph.D.

**Tutors:**

- Görlich, Jonas
- Bose, Krishnendu
- Taghizadeh Motlagh, Seyed Arman

**Participant:**

- Martin Enrique Iribarren

# Ethogram 2.0

## Preface

The concept of an ethogram can be defined in a single line, as stated by **Eibl-Eibesfeldt (1979):**

"*An ethogram is the exact catalog of all the behavioral forms typical of an animal*"

However, a more extensive description considers not only the classification of behaviors but also their underlying structure and function. In this regard, **Ferrari (2005)** expands the definition by integrating elements of semantic relationships, the role of the self, and the temporal dynamics of behavior:

"*An ethogram is a list of the forms of interaction of the subject with the environment (understanding that a grooming behavior is, ultimately, a way of relating, of preparing to relate to the environment). It is an analytical way of approaching behavior. It thus requires a subsequent synthesis movement to recompose that uninterrupted flow of actions. This synthesis movement can be carried out in different ways*""

## 1. Introduction

This notebook presents a review of approaches at the intersection of ethology and machine learning methodologies.

To achieve this, I will refer to Hoffmann et al. work, *A Benchmark for Computational Analysis of Animal Behavior Using Animal-Borne Tags* (2024) The goal is to showcase the structure and types of data available and evaluate the suitability of different machine learning and deep learning techniques for behavior classification. This will help address

the question of how and why these methodologies and protocols are essential for conducting an efficient and precise analysis of animal behavior.

This study does not aim to assess the hardware efficiency of different bio-loggers. Instead, by systematically reviewing the species studied, the structure and characteristics of the collected data, and the challenges encountered in behavioral annotation, this work seeks to provide a comprehensive explanation of the existing benchmark.

The goal is to optimize the alignment between data properties and analytical methodologies, enabling a more effective selection of ecological data and behavioral signals. By considering machine learning techniques, sensor modalities, and species-specific movement dynamics, this approach ensures methodological rigor and enhances the ecological validity of computational ethology.

# 2. General Data Analysis and Visualization

In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import os

# Definir el path relativo al dataset
dataset_path = os.path.join("..", "dataset", "experiments_by_fold_behavior.csv")

# Upload Dataset
df = pd.read_csv(dataset_path)

#Data Wranglin and Cleaning

# Dictionary mapping original names to new names
mapping_names = {
    "HAR": "Humans",
    "baglione_crows": "Crows",
    "desantis_rattlesnakes": "Rattlesnakes",
    "friedlaender_whales": "Whales",
    "jeantet_turtles": "Turtles",
    "ladds_seals": "Seals",
    "maekawa_gulls": "Gulls",
    "pagano_bears": "Polar Bears",
    "vehkaoja_dogs": "Dogs"
}

# Replace the values in the 'dataset' column
df["dataset"] = df["dataset"].replace(mapping_names)


mapping_experiments = {
    "CRNN": "CRNN",
    "CNN": "CNN",
    "dt": "DT",
    "rf": "RF",
```

```
    "svm": "SVM",
    "wavelet_dt": "Wavelet DT",
    "wavelet_rf": "Wavelet RF",
    "wavelet_svm": "Wavelet SVM",
    "harnet_random_nogyr": "HarNet Random NoGyr",
    "harnet_unfrozen_nogyr": "HarNet Unfrozen NoGyr",
    "RNN_nogyr": "RNN",
    "CRNN_nogyr": "CRNN NoGyr",
    "rf_nogyr": "RF NoGyr",
    "harnet_low_data_nogyr": "HarNet Low Data NoGyr",
    "harnet_random_low_data_nogyr": "HarNet Random Low Data NoGyr",
    "harnet_unfrozen_low_data_nogyr": "HarNet Unfrozen Low Data NoGyr",
    "RNN_low_data_nogyr": "RNN Low Data NoGyr",
    "wavelet_RNN_low_data_nogyr": "Wavelet RNN Low Data NoGyr",
    "CRNN_low_data_nogyr": "CRNN Low Data NoGyr",
    "rf_low_data_nogyr": "RF Low Data NoGyr",
    "CRNN_low_data": "CRNN Low Data",
    "rf_low_data": "RF Low Data"
}

# Replace the values in the 'experiment' column
df["experiment"] = df["experiment"].replace(mapping_experiments)


mapping_behavior = {
    'LAYING': 'Lying',
    'SITTING': 'Sitting',
    'STANDING': 'Standing',
    'WALKING': 'Walking',
    'WALKING_DOWNSTAIRS': 'Walking Downstairs',
    'WALKING_UPSTAIRS': 'Walking Upstairs',
    'flying': 'Flying',
    'in_nest': 'In Nest',
    'Moving': 'Moving',
    'Not_Moving': 'Stationary',
    'exploratory': 'Exploring',
    'feed': 'Feeding',
    'rest': 'Resting',
    'travel': 'Traveling',
    'breathing': 'Breathing',
    'feeding': 'Feeding',
    'gliding': 'Gliding',
    'resting': 'Resting',
    'scratching': 'Scratching',
    'staying_at_surface': 'At Surface',
    'swimming': 'Swimming',
    'grooming': 'Grooming',
    'travelling': 'Traveling',
    'foraging': 'Foraging',
    'stationary': 'Stationary',
    'dig': 'Digging',
    'eat': 'Eating',
    'groom': 'Grooming',
    'head_shake': 'Head Shake',
    'pounce': 'Pouncing',
    'roll': 'Rolling',
    'run': 'Running',
    'swim': 'Swimming',
    'walk': 'Walking',
    'galloping': 'Galloping',
```

```
    'lying_chest': 'Lying Chest',
    'panting_lying_chest': 'Panting (Lying Chest)',
    'panting_sitting': 'Panting (Sitting)',
    'panting_standing': 'Panting (Standing)',
    'shaking': 'Shaking',
    'sitting': 'Sitting',
    'sniffing': 'Sniffing',
    'standing': 'Standing',
    'trotting': 'Trotting',
    'walking': 'Walking'
}

# Replace in dataset
df["behavior_class"] = df["behavior_class"].replace(mapping_behavior)


# Rename Column
df.rename(columns={"train_ground_truth_label_counts": "num_samples_train"}, inpl

#Cleaning Columns
df = df.drop(columns=['fig4', 'fig5d', 'fig5e'])

# Convert categorical columns to category type
df['dataset'] = df['dataset'].astype('category')
df['dataset'] = df['dataset'].astype('category')
df['behavior_class'] = df['behavior_class'].astype('category')
df['experiment'] = df['experiment'].astype('category')


# Display the cleaned dataset
df = df.round(2)
df.head()
```

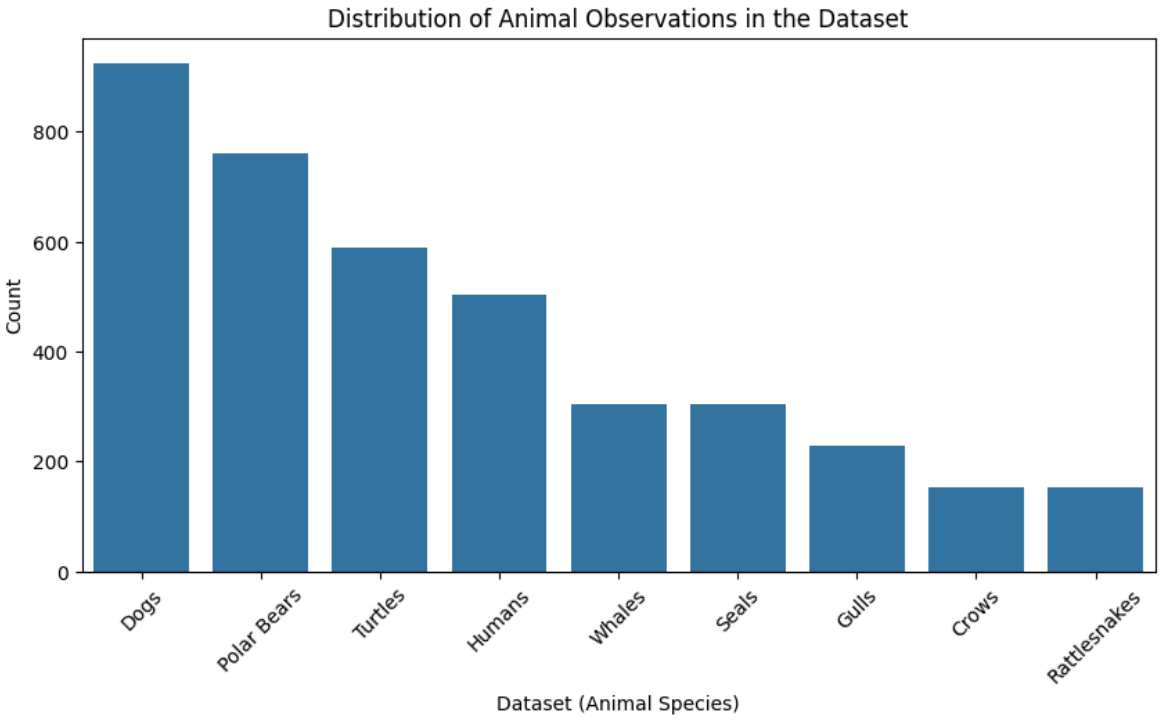
Out[ ]:

| | dataset | experiment | fold | behavior_class | f1 | precision | recall | num_samples_trai |
|---|---|---|---|---|---|---|---|---|
| **0** | Humans | harnet_nogyr | 1 | Lying | 1.00 | 1.00 | 1.00 | 10999 |
| **1** | Humans | harnet_nogyr | 1 | Sitting | 0.96 | 0.99 | 0.93 | 10203 |
| **2** | Humans | harnet_nogyr | 1 | Standing | 0.97 | 0.94 | 1.00 | 10998 |
| **3** | Humans | harnet_nogyr | 1 | Walking | 1.00 | 1.00 | 1.00 | 9856 |
| **4** | Humans | harnet_nogyr | 1 | Walking Downstairs | 1.00 | 1.00 | 1.00 | 8704 |

In [ ]:

```
# Dataset Distribution (Animals)
plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='dataset', order=df['dataset'].value_counts().index)
plt.xticks(rotation=45)
plt.title("Distribution of Animal Observations in the Dataset")
plt.xlabel("Dataset (Animal Species)")
plt.ylabel("Count")
plt.show()

# Count behavioral observations per dataset (summing all observations across all
total_observations = df.groupby("dataset").size().reset_index(name="num_observat
total_observations = total_observations.sort_values(by="num_observations", ascen
```

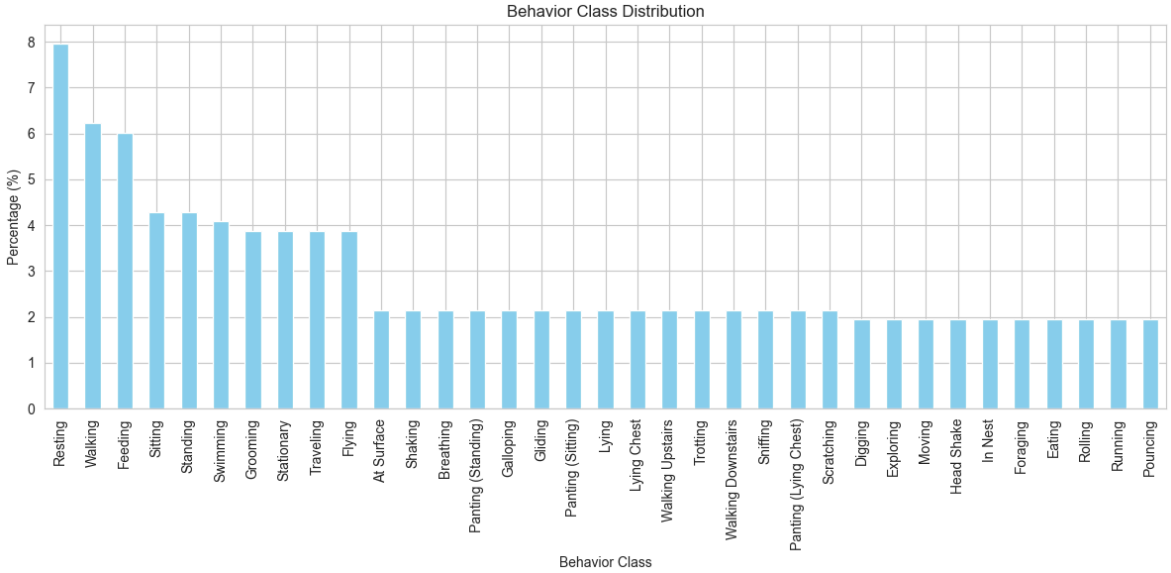
```
# Display results
print("Total observations per dataset:", "\n")
print(total_observations)
```




```
Total observations per dataset:
        dataset  num_observations
1          Dogs               924
4   Polar Bears               760
7       Turtles               588
3        Humans               504
8        Whales               304
6         Seals               304
2         Gulls               228
0         Crows               152
5  Rattlesnakes               152
```

In [337…

```
# Behavior Classes Distribution
# Behavior class distribution in percentage
behavior_counts = df["behavior_class"].value_counts(normalize=True) * 100

plt.figure(figsize=(12, 6))
behavior_counts.plot(kind='bar', color='skyblue')
plt.xticks(rotation=90)
plt.xlabel('Behavior Class')
plt.ylabel('Percentage (%)')
plt.title('Behavior Class Distribution')
plt.tight_layout()
plt.show()

behavior_distribution = pd.DataFrame(behavior_counts).reset_index()
behavior_distribution.columns = ["behavior_class", "percentage"]
behavior_distribution["percentage"] = behavior_distribution["percentage"].round(

# Display results
print("\nBehavior class distribution:", "\n")
print(behavior_distribution)
```

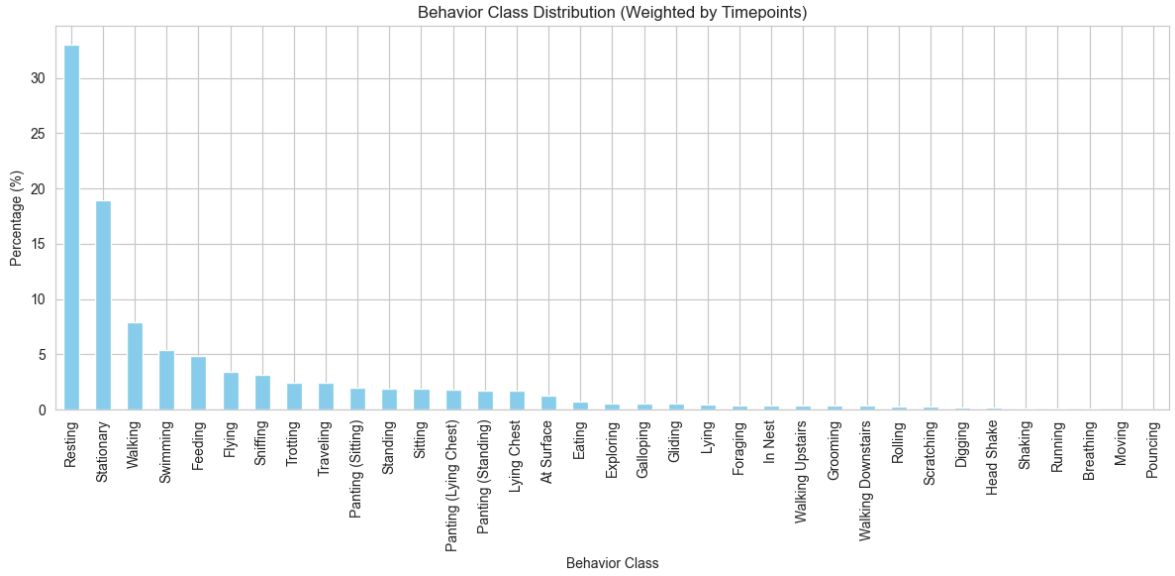



Behavior class distribution:

| | behavior_class | percentage |
|---|---|---|
| 0 | Resting | 7.97 |
| 1 | Walking | 6.23 |
| 2 | Feeding | 6.03 |
| 3 | Sitting | 4.29 |
| 4 | Standing | 4.29 |
| 5 | Swimming | 4.09 |
| 6 | Grooming | 3.88 |
| 7 | Stationary | 3.88 |
| 8 | Traveling | 3.88 |
| 9 | Flying | 3.88 |
| 10 | At Surface | 2.15 |
| 11 | Shaking | 2.15 |
| 12 | Breathing | 2.15 |
| 13 | Panting (Standing) | 2.15 |
| 14 | Galloping | 2.15 |
| 15 | Gliding | 2.15 |
| 16 | Panting (Sitting) | 2.15 |
| 17 | Lying | 2.15 |
| 18 | Lying Chest | 2.15 |
| 19 | Walking Upstairs | 2.15 |
| 20 | Trotting | 2.15 |
| 21 | Walking Downstairs | 2.15 |
| 22 | Sniffing | 2.15 |
| 23 | Panting (Lying Chest) | 2.15 |
| 24 | Scratching | 2.15 |
| 25 | Digging | 1.94 |
| 26 | Exploring | 1.94 |
| 27 | Moving | 1.94 |
| 28 | Head Shake | 1.94 |
| 29 | In Nest | 1.94 |
| 30 | Foraging | 1.94 |
| 31 | Eating | 1.94 |
| 32 | Rolling | 1.94 |
| 33 | Running | 1.94 |
| 34 | Pouncing | 1.94 |

In [ ]:
```
# Behavior Classes Distribution using Sample Train
behavior_counts = df.groupby("behavior_class")["num_samples_train"].sum()

behavior_counts_percentage = (behavior_counts / behavior_counts.sum()) * 100
```

```
plt.figure(figsize=(12, 6))
behavior_counts_percentage.sort_values(ascending=False).plot(kind='bar', color='
plt.xticks(rotation=90)
plt.xlabel('Behavior Class')
plt.ylabel('Percentage (%)')
plt.title('Behavior Class Distribution (Weighted by Timepoints)')
plt.tight_layout()
plt.show()
```




# 3. Behavior in Dataset

In [ ]:

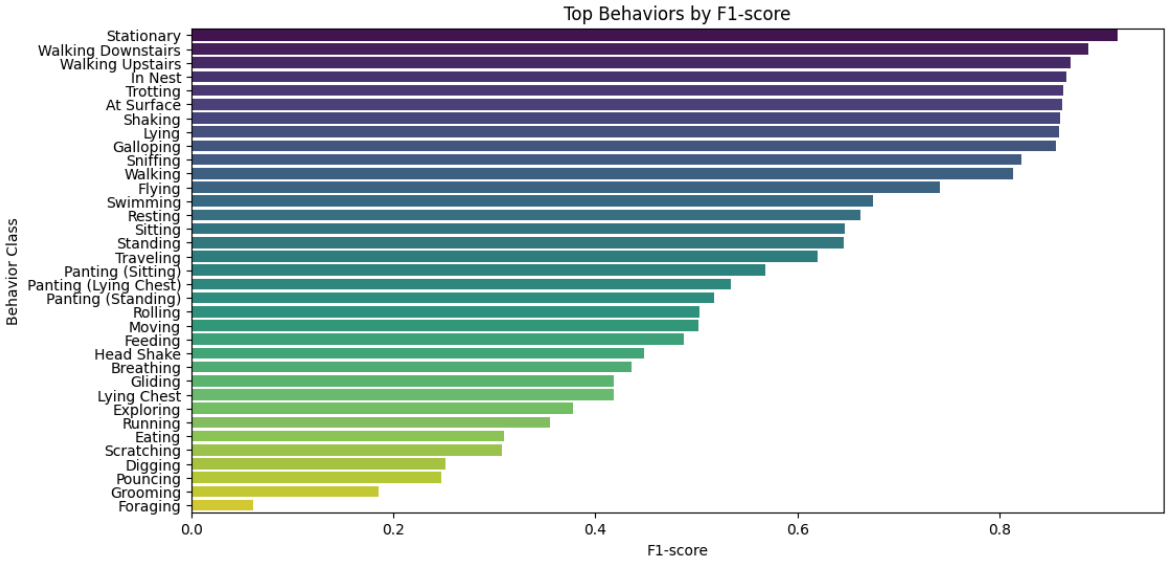
```
# Contar las clases de comportamiento únicas por dataset
unique_behaviors_per_dataset = df.groupby("dataset")["behavior_class"].nunique()
unique_behaviors_per_dataset.rename(columns={"behavior_class": "unique_behaviors

# Calcular el número total de comportamientos únicos en todos los datasets
total_unique_behaviors = df["behavior_class"].nunique()

# Calcular el porcentaje de comportamientos únicos por dataset
unique_behaviors_per_dataset["percentage"] = (unique_behaviors_per_dataset["uniq

# Redondear los valores a dos decimales
unique_behaviors_per_dataset["percentage"] = unique_behaviors_per_dataset["perce

# Ordenar por porcentaje en orden descendente
unique_behaviors_per_dataset = unique_behaviors_per_dataset.sort_values(by="perc

# Mostrar la tabla resultante
display(unique_behaviors_per_dataset)
```

| | dataset | unique_behaviors | percentage |
|---|---|---|---|
| **1** | Dogs | 11 | 31.43 |
| **4** | Polar Bears | 10 | 28.57 |
| **7** | Turtles | 7 | 20.00 |
| **3** | Humans | 6 | 17.14 |
| **8** | Whales | 4 | 11.43 |
| **6** | Seals | 4 | 11.43 |
| **2** | Gulls | 3 | 8.57 |
| **0** | Crows | 2 | 5.71 |
| **5** | Rattlesnakes | 2 | 5.71 |

In [325…

```
# Agrupar por comportamiento y calcular la media de las métricas
behavior_performance = df.groupby("behavior_class")[["f1", "precision", "recall"

# Ordenar por cada métrica en orden descendente
best_f1 = behavior_performance.sort_values(by="f1", ascending=False)
best_precision = behavior_performance.sort_values(by="precision", ascending=Fals
best_recall = behavior_performance.sort_values(by="recall", ascending=False)

# Mostrar los 5 mejores comportamientos en cada métrica
print("Top 5 behaviors by F1-score:\n", best_f1[["f1"]].head(5), "\n")
print("Top 5 behaviors by Precision:\n", best_precision[["precision"]].head(5),
print("Top 5 behaviors by Recall:\n", best_recall[["recall"]].head(5), "\n")

# Visualización con Seaborn
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))
sns.barplot(x=best_f1["f1"], y=best_f1.index, palette="viridis", order=best_f1.i
plt.title("Top Behaviors by F1-score")
plt.xlabel("F1-score")
plt.ylabel("Behavior Class")
plt.show()
```

```
Top 5 behaviors by F1-score:
                          f1
behavior_class
Stationary          0.916579
Walking Downstairs  0.887976
Walking Upstairs    0.870357
In Nest             0.866053
Trotting            0.863452

Top 5 behaviors by Precision:
                    precision
behavior_class
In Nest              0.934079
Stationary           0.917632
At Surface           0.897143
Lying                0.893929
Walking Downstairs   0.884167

Top 5 behaviors by Recall:
                       recall
behavior_class
Stationary          0.937039
Shaking             0.933810
Walking Upstairs    0.905952
Walking Downstairs  0.905119
Galloping           0.889643
```

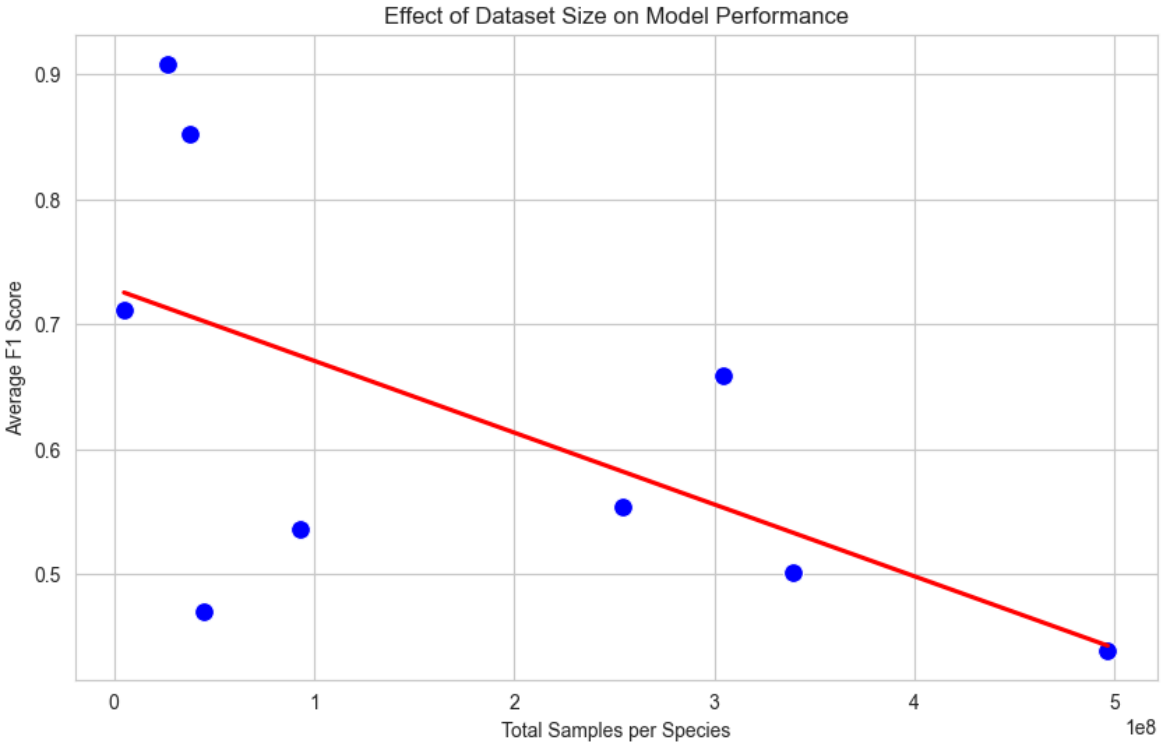



In [339…

```python
from scipy.stats import spearmanr

# Suponiendo que la columna 'train_ground_truth_label_counts' representa el núme
# se puede calcular el tamaño total del dataset (por especie) sumando estos valo
dataset_size = df.groupby("dataset")["num_samples_train"].sum().reset_index()
dataset_size.rename(columns={"num_samples_train": "total_samples"}, inplace=True

# Calcular el rendimiento promedio (F1 score) por especie
avg_f1_by_species = df.groupby("dataset")["f1"].mean().reset_index()

# Combinar ambas informaciones en un solo DataFrame
df_size_perf = pd.merge(dataset_size, avg_f1_by_species, on="dataset")

print("Dataset Size and Average F1 Score by Species:")
print(df_size_perf)
```

```
# Scatter plot para visualizar la relación entre el tamaño del dataset y el rend
plt.figure(figsize=(10, 6))
sns.scatterplot(x="total_samples", y="f1", data=df_size_perf, s=100, color="blue
sns.regplot(x="total_samples", y="f1", data=df_size_perf, scatter=False, color="
plt.xlabel("Total Samples per Species")
plt.ylabel("Average F1 Score")
plt.title("Effect of Dataset Size on Model Performance")
plt.show()

# Calcular la correlación de Spearman
corr_size, pval_size = spearmanr(df_size_perf["total_samples"], df_size_perf["f1
print(f"Spearman Correlation (Dataset Size vs. F1): {corr_size:.2f} (p-value: {p
```

```
Dataset Size and Average F1 Score by Species:
        dataset  total_samples        f1
0         Crows       26779750  0.908355
1          Dogs      304680061  0.659264
2         Gulls      339729437  0.501535
3        Humans       37748333  0.852619
4   Polar Bears      496699460  0.439013
5  Rattlesnakes        5084372  0.711908
6         Seals       44865886  0.470329
7       Turtles      254365629  0.553418
8        Whales       92794977  0.536316
```




```
Spearman Correlation (Dataset Size vs. F1): -0.72 (p-value: 0.030)
```

The scatter plot above, shows a downward trend suggesting that, in this analysis and with the given data, fewer samples are associated with higher F1 scores. This is in contrast to the expected assumption that larger datasets would enhance the model.

# 4. Comparing Models

In [307…

```
# Model Distribution
experiment_counts = df["experiment"].value_counts(normalize=True) * 100
```

```
# Dataframe
df_experiment_distribution = pd.DataFrame({
    "Experiment": experiment_counts.index,
    "Percentage": experiment_counts.values
})

df_experiment_distribution["Percentage"] = df_experiment_distribution["Percentag

# Display
display(df_experiment_distribution)
```

| | Experiment | Percentage |
|---|---|---|
| **0** | CNN | 5.01% |
| **1** | CRNN | 5.01% |
| **2** | HarNet Low Data NoGyr | 5.01% |
| **3** | DT | 5.01% |
| **4** | HarNet Unfrozen Low Data NoGyr | 5.01% |
| **5** | HarNet Random NoGyr | 5.01% |
| **6** | HarNet Random Low Data NoGyr | 5.01% |
| **7** | Wavelet RF | 5.01% |
| **8** | Wavelet RNN Low Data NoGyr | 5.01% |
| **9** | Wavelet SVM | 5.01% |
| **10** | HarNet Unfrozen NoGyr | 5.01% |
| **11** | RF | 5.01% |
| **12** | SVM | 5.01% |
| **13** | RNN Low Data NoGyr | 5.01% |
| **14** | RNN | 5.01% |
| **15** | harnet_nogyr | 5.01% |
| **16** | Wavelet DT | 5.01% |
| **17** | CRNN Low Data | 2.55% |
| **18** | RF Low Data | 2.55% |
| **19** | CRNN Low Data NoGyr | 2.45% |
| **20** | CRNN NoGyr | 2.45% |
| **21** | RF NoGyr | 2.45% |
| **22** | RF Low Data NoGyr | 2.45% |

In [343…

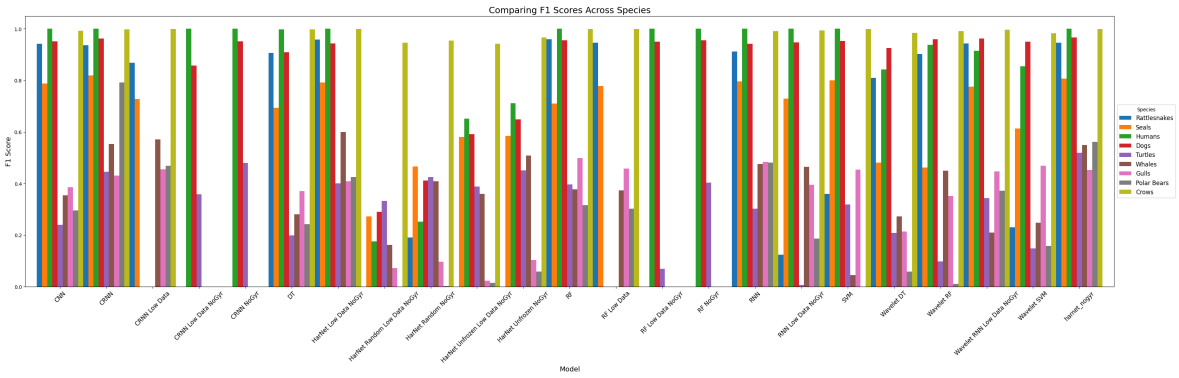
```
#Data Rearrange by Model Criteria

deep_learning_models = [
    "CRNN", "CNN", "RNN", "CRNN NoGyr", "RNN NoGyr", "CRNN Low Data", "RNN Low D
```

```
    "HarNet Random NoGyr", "HarNet Unfrozen NoGyr", "HarNet Low Data NoGyr",
    "HarNet Random Low Data NoGyr", "HarNet Unfrozen Low Data NoGyr",
    "Wavelet RNN Low Data NoGyr"
]

machine_learning_models = [
    "DT", "RF", "SVM", "Wavelet DT", "Wavelet RF", "Wavelet SVM",
    "RF NoGyr", "RF Low Data", "RF Low Data NoGyr"
]

# DL models appear first in the order
ordered_models = deep_learning_models + machine_learning_models

# Group the dataset by 'experiment' and 'dataset' and compute the mean F1 score
df_grouped = df.groupby(["experiment", "dataset"], observed=False)["f1"].mean().

# Create the pivot table
df_pivoted = df_grouped.pivot(index="experiment", columns="dataset", values="f1"

# Remove duplicate
df_cleaned = df.drop_duplicates(subset=["experiment", "dataset"])

# Reorder DataFrame
existing_models = [model for model in ordered_models if model in df_pivoted.inde
df_pivoted = df_pivoted.loc[existing_models]

# Crear el heatmap
plt.figure(figsize=(12, 7))
sns.heatmap(df_pivoted, annot=True, fmt=".3f", cmap="magma", linewidths=0.5)

# Create the heatmap based on Figure 4 (Hoffmann, et al. 2024)
plt.title("Mean Test F1 (Standard Deviation)")
plt.xlabel("Dataset")
plt.ylabel("Model")

# Mostrar el gráfico
plt.show()
```

In the spirit of the analysis conducted in Figure 4 of Hoffmann et al. (2024), we observe that the datasets for Crows, Dogs, and Humans exhibit well-defined classes and high-quality data, making them highly valuable for the current benchmark. In contrast, the datasets for Gulls, Bears, Turtles, and Whales probably contains significant noise, complex analytical patterns, and consequently, difficulty in an accurate classification.

In [306…

```
import matplotlib.pyplot as plt

# Selected species to compare
selected_species = ["Dogs", "Polar Bears", "Turtles", "Humans", "Whales", "Seals

# Species in df_pivoted
available_species = list(set(selected_species) & set(df_pivoted.columns))

# Filter the dataset
df_subset = df_pivoted[available_species].copy()

# Bar chart
df_subset.plot(kind='bar', figsize=(40, 10), width=1)

plt.title("Comparing F1 Scores Across Species", fontsize = 18)
plt.xlabel("Model", fontsize=14)
plt.ylabel("F1 Score", fontsize= 14)
plt.xticks(rotation=45, fontsize=12)
plt.legend(title="Species", loc='center left', bbox_to_anchor=(1, 0.5), fontsize

plt.show()
```




In the bar chart above, we observe the F1 scores for Dogs, Polar Bears, Turtles, Humans, and Whales, highlighting clear performance differences across species. Humans and Crows consistently achieve the highest F1 scores across various models, while Turtles, Whales, and Polar Bears show lower and more variable performance.

We are likely encountering a higher number of recorded data points, though not necessarily a greater quality or variety of behaviors. While the dog is the only domestic animal present in the datasets, we should also consider the different environments in which these species live. Both dogs and humans are limited to terrestrial movement, whereas other species include aerial, aquatic, or hybrid movement patterns, which significantly alter the way data is collected.

# 5. Conclusion

With this, I have only scratched the surface of what Marks et al (2020) have referred to as the *Deep Learning Swiss Knife*.

Leveraging the entirety of raw data, as well as integrating multiple datasets, represents a step toward pure and necessary research. With enhanced computational power and a personal advancement in methodological precision, this notebook remains an open project, serving as a dynamic space for data visualization, related literature compilation, and continuous advancements in the field.

Future iterations could incorporate more refined preprocessing techniques, larger datasets, and alternative modeling approaches to further explore the capabilities and limitations of deep learning methodologies, while improving the acquisition of raw data and integrating more precise signal measurements.

This is not the end, but rather a foundation for future explorations, improvements, and discoveries.

**Bibliography:**


- Anderson, D. J., & Perona, P. (2014). Toward a Science of Computational Ethology. Neuron, 84(1), 18–31. https://doi.org/10.1016/j.neuron.2014.09.005
- Bohnslav, J. P., Wimalasena, N. K., Clausing, K. J., Dai, Y. Y., Yarmolinsky, D. A., Cruz, T., Kashlan, A. D., Chiappe, M. E., Orefice, L. L., Woolf, C. J., & Harvey, C. D. (2021). DeepEthogram, a machine learning pipeline for supervised behavior classification from raw pixels. eLife, 10, e63377. https://doi.org/10.7554/eLife.63377
- Fazzari, E., Romano, D., Falchi, F., & Stefanini, C. (2024). Animal behavior analysis methods using deep learning: A survey. arXiv preprint arXiv:2405.14002. http://arxiv.org/abs/2405.14002
- Ferrari, H. R. (2002). Sobre el etograma, 1: del etograma como lenguaje al lenguaje de los etogramas. Revista de Etología, 4(2), 3–14.
- Ferrari, H. R., Lahitte, H. B., & Lázaro, L. C. (2005). Etogramática: Teoría y práctica de la descripción en ciencias del comportamiento. Editorial Nobuko.
- Hoffman, B. et. al (2024). A benchmark for computational analysis of animal behavior, using animal-borne tags. Movement Ecology, 12(1), 78. https://doi.org/10.1186/s40462-024-00511-8
- Lorenz, K. (1981). The Foundations of Ethology. Springer-Verlag.
- Tinbergen, N. (1963). On aims and methods of ethology. Zeitschrift für Tierpsychologie, 20, 410–433. https://doi.org/10.1111/j.1439-0310.1963.tb01161.x