Naive Bayes predictor

In [1]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import StratifiedKFold
import numpy as np
import pickle
from sklearn.metrics import accuracy_score, classification_report, confusion
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Load the Data Set

will be split to test and train

In [2]:
```
X = np.load("../X.npy")
y = np.load("../y.npy")

with open("../label_map.pkl", "rb") as f:
    label_map = pickle.load(f)
```

Split the data to n-fold

In [3]:
```
nfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=777)
fold = 1
```

Separate them into 5 different modules

to see how the module will act on the data

In [4]:
```
from sklearn.metrics import precision_recall_fscore_support

accuracies = []
precisions, recalls, f1s = [], [], []
conf_matrices = []
scaler = StandardScaler()

for training_index, testing_index in nfold.split(X, y):# the loop calls next
    # train_index: A NumPy array holds all indices of the training items in

    # Get the training and testing data for this fold
    X_train, X_test = X[training_index], X[testing_index]
    y_train, y_test = y[training_index], y[testing_index]


    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```python
    # Normalize feature values
    """
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    """

    # Train Naive Bayes model
    model = GaussianNB()
    model.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = model.predict(X_test)

    # Save performance measurements
    # Accuracy
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)
    # Precision, Recall, F1 (macro average)
    p, r, f, _ = precision_recall_fscore_support(y_test, y_pred, average='ma
    precisions.append(p)
    recalls.append(r)
    f1s.append(f)
    # Confusion matrix (optional)
    cm = confusion_matrix(y_test, y_pred)
    conf_matrices.append(cm)

    fold += 1

print("\n=== Average Cross-Validation Results ===")
print(f"Average Accuracy: {np.mean(accuracies):.4f}")
print(f"Average Precision: {np.mean(precisions):.4f}")
print(f"Average Recall:    {np.mean(recalls):.4f}")
print(f"Average F1-score:  {np.mean(f1s):.4f}")
```

```
=== Average Cross-Validation Results ===
Average Accuracy: 0.7284
Average Precision: 0.7375
Average Recall:    0.7287
Average F1-score:  0.7265
```

More Statistics

In [5]:

```python
avg_cm = np.mean(conf_matrices, axis=0).astype(int)
# Print in clean table format
print("\nAverage Confusion Matrix:")
labels = list(label_map.values())
header = "Predicted →\t\t" + "\t".join(labels)
print(header)
for i, row in enumerate(avg_cm):
    row_str = "\t".join(f"{val:.2f}" for val in row)
    print(f"Actual {labels[i]}:\t{row_str}")
```

```
Average Confusion Matrix:
Predicted →              cats    panda   spiders
Actual cats:    136.00  39.00   24.00
Actual panda:   24.00   169.00  4.00
Actual spiders: 34.00   36.00   130.00
```

Scale the data

In [6]:
```
# Normalize to improve model performance
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [7]:
```
import joblib
joblib.dump(scaler, "NB_scaler.pkl")
```

Out[7]:
```
['NB_scaler.pkl']
```

Add the final module

In [8]:
```
final_model = GaussianNB()
final_model.fit(X, y)
# save the final model:
with open("NB_model.pkl", "wb") as f:
    pickle.dump(final_model, f)
```