Étude du Dataset Pokémon

Objectifs :

- Explorer un jeu de données et identifier les variables importantes (EDA).
- Analyser la distribution des données et la présence de valeurs manquantes.
- Étudier les corrélations entre variables.
- Effectuer un prétraitement complet : gestion des valeurs manquantes, encodage, création de nouvelles features et normalisation.
- Préparer les données pour un futur modèle de Machine Learning.

L'objectif est de construire un modèle permettant de **prédire si un Pokémon est légendaire ou non** à partir de ses caractéristiques.

Travail demandé :

Étape 1 : Importation des bibliothèques

Étape 2 : Importation de la dataset

Etapes 3 : Exploration de dataset

Étape 7 : Prétraitement des données

1. Importation des bibliothèques

Les bibliothèques fondamentales dont on aura besoin sont les suivantes :

2. Importation de la dataset

En utilisant la bibliothèque Pandas, on importe la dataset qui est dans l'exemple ci-après sous format CSV.

```
from google.colab import files
import io
uploaded = files.upload()
df = pd.read_csv(io.BytesIO(uploaded[pokemon.csv']),encoding = "utf-8")
```

3. Exploration des données

Explorer la forme et les principales caractéristiques de votre dataset (taille, types de variables, valeurs disponibles, premières lignes, statistiques descriptives)

Vérifier :

a. La dimension (ndim)
b. La taille du dataset (shape)
c. Les 5 premières lignes , les 5 derniere lignes
d. Les types de variables et valeurs manquantes
e. Les statistiques descriptives
f. Les valeurs uniques de la variable species et leurs fréquences.
g. Les valeurs manquantes
h. Les valeurs catégorielles
i. Les correlation (features/features et features /target)
   - Pour les variables **numériques ↔ numériques**, on utilise le **coefficient de corrélation de Pearson** et on affiche avec scatterplot, heatmap corrélations.

```
corr_matrix = df.corr(numeric_only=True)
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

   - +1 : corrélation positive forte
   - -1 : corrélation négative forte
   - 0 : aucune corrélation linéaire
   - Corrélation entre une **feature numérique et la target binaire**

     On utilise :

On importe la fonction pointbiserialr depuis le module scipy.stats. Elle retourne : le **coefficient de corrélation (corr)** entre -1 et +1, et la **p-value** associée (pour tester la significativité statistique).

```
from scipy.stats import pointbiserialr

for col in ['attack','defense','speed','capture_rate','hp']:

    corr, p = pointbiserialr(df[col], df['is_legendary'])

    print(f"{col} → corr={corr:.3f}, p-value={p:.3f}")
```

- corr proche de 0 → aucune corrélation
- corr > 0.3 ou < -0.3 → corrélation notable
- p-value < 0.05 → corrélation significative statistiquement

- Corrélation entre **features catégorielles** et la target on utilise le **test du chi-deux (χ^2)** :

chi2 : fonction qui permet de calculer la statistique du test du **chi-deux** et sa **p-value** pour évaluer la dépendance entre deux variables catégorielles.

LabelEncoder : convertit des **valeurs textuelles (catégorielles)** en **valeurs numériques** (car le test du chi-deux ne travaille que sur des nombres).

```
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder
X_cat = df[['type1', 'generation', 'has_gender']]
for col in X_cat.columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
chi_scores = chi2(df[X_cat.columns], df['is_legendary'])
chi2_df = pd.DataFrame({'Feature': X_cat.columns, 'Chi2': chi_scores[0], 'p-value': chi_scores[1]})
*******************************
ct = pd.crosstab(df['type1'], df['is_legendary'], normalize='index')
ct.plot(kind='bar', stacked=True)
```

- chi2_df.sort_values(by='Chi2', ascending=False)
- p-value < 0.05 → relation significative entre la catégorie et is_legendary.
- Par exemple : generation est souvent corrélée à is_legendary car certains légendaires appartiennent à des générations spécifiques.

j. La distribution normale des feautres numeric

4. Pre-processing

a. Valeurs manquantes

S'assurer que le dataset ne contient pas de valeurs NaN ou nulles qui empêcheraient l'apprentissage du modèle.

- **Suppression** : si la colonne contient trop de valeurs manquantes.

```
df.drop(columns=['colonne_inutile'], inplace=True)
```

- **Imputation** : si la colonne est utile.
  - Par la moyenne (numérique) :

```
df['age'].fillna(df['age'].mean(), inplace=True)

df['height_m'].replace( np.nan , df ['height_m'].mean(),inplace=True )
```

  - Par le mode (catégorielle)

```
df['type1'].fillna(df['type1'].mode()[0], inplace=True)
```

b. Détection et traitement des doublons

Supprimer les lignes dupliquées pour éviter de biaiser le modèle.

```
df.duplicated().sum()
df.drop_duplicates(inplace=True)
```

c. Colonnes non pertinentes (bruit)

Supprimer les colonnes qui :

- N'apportent aucune information utile pour la prédiction,
- Ou qui fuitent la cible (*data leakage*).

```
df.drop(['pokedex_number', 'name', 'japanese_name'], axis=1, inplace=True)
```

d. Création ou transformation de variables

Créer de nouvelles features utiles ou simplifier celles existantes.

Exemple : Créer une variable binaire two_types indiquant si un Pokémon a deux types ou non

```
df['two_types'] = df['type2'].notnull().astype(int)
```

ou

```
for index,row in df.iterrows():
    try:
    np.isnan(row['type2'])
```

```
df.at[index,'Tow_types'] = True
except:
df.at[index,'Tow_types'] = False
```

e. Encodage des variables catégorielles

Convertir les colonnes de type texte (object) en valeurs numériques exploitables par les algorithmes.

- **Label Encoding**

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['type1'] = le.fit_transform(df['type1'])
```

- **One-Hot Encoding**

```
df = pd.get_dummies(df, columns=['type1', 'type2'], drop_first=True)
```

f. Normalisation / Standardisation des données

Mettre les variables numériques sur la même échelle pour éviter qu'une variable à grande valeur domine les autres.

- **Standardisation :**

(centrer et réduire : moyenne = 0, écart-type = 1)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.select_dtypes('number'))
```

- **Normalisation :**

(ramène les valeurs entre 0 et 1)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(df.select_dtypes('number'))
```

g. Séparation des features et de la cible

Séparer la variable à prédire (target) des autres variables (features).

```
X = df.drop('is_legendary', axis=1)
```

```
y = df['is_legendary']
```

h. Division du dataset en ensembles d'entraînement et de test

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```