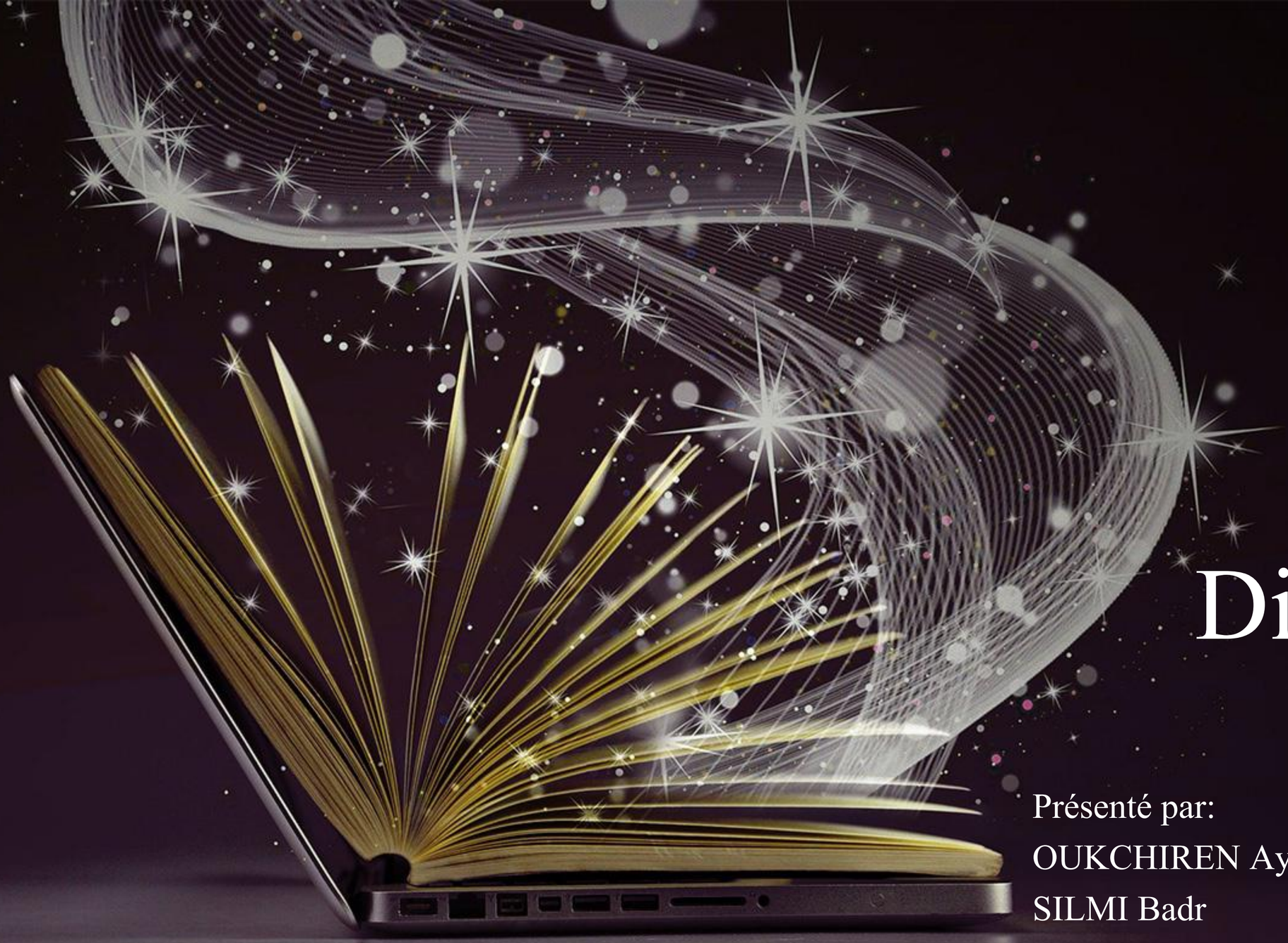


Analyse Factorielle Discriminante (AFD)

Présenté par:
OUKCHIREN Ayoub
SILMI Badr

Encadré par:
Prof. CHAMLAL Hasna

Sommaire

Histoire

Fondation

L'histoire de l'Analyse Factorielle Discriminante (AFD) remonte aux travaux fondateurs de Ronal A. Fisher

1993

Développement

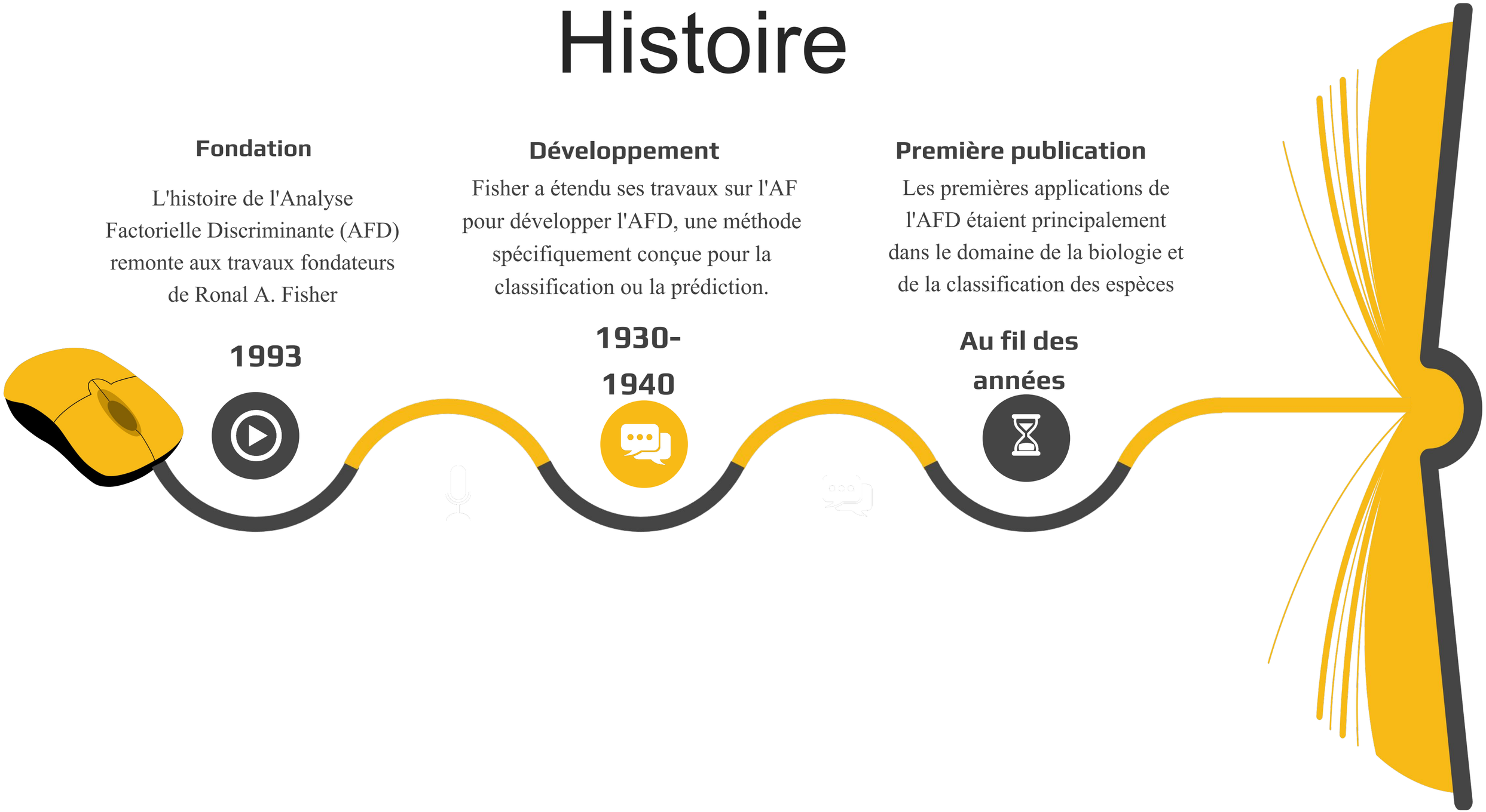
Fisher a étendu ses travaux sur l'AF pour développer l'AFD, une méthode spécifiquement conçue pour la classification ou la prédiction.

1930-1940

Première publication

Les premières applications de l'AFD étaient principalement dans le domaine de la biologie et de la classification des espèces

Au fil des années



Présentation générale

L'AFD simplifie les données tout en préservant les informations importantes pour faciliter l'interprétation et améliorer les performances des modèles prédictifs.

L'AFD optimise la discrimination entre les groupes, ce qui permet de développer des modèles prédictifs plus précis en se basant sur les caractéristiques les plus discriminantes.

L'AFD identifie les combinaisons linéaires de variables qui optimisent la séparation entre les groupes, permettant une meilleure compréhension des facteurs impliqués et la création de frontières de décision claires.

L'AFD réduit la dimensionnalité, maximise la séparation entre les groupes et améliore la prédiction, utilisée dans divers domaines cruciaux.

Objectif de notre étude

Problématique

L'objectif de notre problématique est de prédire la variable "Fatal" dans le contexte des attaques de requins.

En d'autres termes, nous cherchons à déterminer si une attaque de requin est susceptible de provoquer des blessures mortelles ou non.

Objectif final

L'objectif final est d'utiliser des techniques d'analyse et de modélisation des données pour développer un modèle prédictif précis qui peut aider à évaluer le risque de décès lors d'une attaque de requin. Cela peut avoir des implications importantes pour la prévention des attaques et la sécurité des personnes dans les zones à risque.

Prétraitement des données

Chargement des données

Nous avons importé les bibliothèques nécessaires (numpy et pandas) et lu le fichier CSV contenant les données des attaques de requins.

```
In 2  1  df = pd.read_csv("attacks.csv", encoding="ISO-8859-1")
      2  df.head()
```

Supression des lignes vides

La ligne de code au dessous utilise la fonction dropna() pour supprimer les lignes contenant des valeurs manquantes d'un DataFrame. Le paramètre how="any" indique que la ligne doit être supprimée si elle contient au moins une valeur manquante

```
In 9  1  df = df.dropna(how="any")
```

Nettoyage des données

```
In 10
import re
import string

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\n', ' ', text)
    text = re.sub(r'\d', '', text)
    text = text.translate(str.maketrans("", "", string.punctuation))
    words = text.split()
    words = [re.sub(r'(.)\1{1,}', r'\1\1', word) for word in words]
    words = [word.strip() for word in words if len(word.strip()) > 1]

    text = " ".join(words)
    return text

In 11
df["Species"] = df["Species"].apply(preprocess_text)
```

- Particules et position

La fonction preprocess_text effectue des étapes de prétraitement du texte pour nettoyer et préparer le texte en vue d'une utilisation ultérieure dans des tâches d'analyse.

- **Les mots traités sont ensuite réassemblés en une seule chaîne de caractères, séparés par des espaces.**

Nettoyage des données

```
In 12  1  df = df[df["Country"] == "USA"]

In 13  1  df["Species"] = df["Species"].apply(lambda text: text.replace("to", "").strip())

In 14  1  df["Species"] = df["Species"].apply(lambda text: text.replace("feet", "").strip())

In 15  1  df["Species"] = df["Species"].apply(lambda text: text.replace("less than", "").strip())

In 16  1  df["Species"] = df["Species"].apply(lambda text: text.replace("possibly", "").strip())

In 17  1  df["Species"] = df["Species"].apply(lambda text: text.replace("small", "").strip())

In 18  1  df["Species"] = df["Species"].apply(lambda text: text.replace("tall", "").strip())

In 19  1  df["Species"] = df["Species"].apply(lambda text: text.replace("brown", "").strip())
       2  df["Species"] = df["Species"].apply(lambda text: text.replace("cm", "").strip())
       3  df["Species"] = df["Species"].apply(lambda text: text.replace("greycolored", "").strip())
```

Filtrage des données

Le DataFrame est filtré pour ne conserver que les lignes où la valeur de la colonne "Country" est "USA". Cela permet de restreindre l'analyse aux données spécifiques à ce pays.

Nettoyage de la colonne "Species"

- Plusieurs transformations sont appliquées à la colonne "Species" pour supprimer des mots spécifiques ou des termes redondants, simplifiant ainsi les descriptions des espèces et réduisant les redondances.

Nettoyage des données

```
In 20   1 def or_split(text):
        2     if "or" in text:
        3         return text.split("or")[0].strip()
        4     else:
        5         return text

In 21   1 df["Species"] = df["Species"].apply(or_split)
```

Traitement des descriptions multiples

La fonction or_split vérifie si le mot "or" est présent dans la description de l'espèce, et si oui, elle divise la chaîne en utilisant "or" comme séparateur et retourne la première partie sans espaces supplémentaires. Sinon, elle renvoie simplement la chaîne d'origine.

Application de la fonction

En utilisant la fonction or_split et en appliquant cette fonction à la colonne "Species", nous avons pu traiter les descriptions d'espèces multiples en ne conservant que la première partie, si le mot "or" était présent. Cette étape de préparation des données vise à simplifier les descriptions d'espèces et à faciliter les analyses ultérieures.

Nettoyage des données

```
In 22   1  def len_strip(text):
        2      if len(text.split()) > 2:
        3          words = text.split()[:2]
        4          text = " ".join(words)
        5          return text
        6      else:
        7          return text

In 23   1  df["Species"] = df["Species"].apply(len_strip)
```

Traitement la colonne "Species" du DataFrame

Cette fonction vérifie la longueur du texte et, si elle contient plus de deux mots, elle conserve uniquement les deux premiers mots et les renvoie sous forme de texte modifié. Cela nous permet de simplifier les descriptions des espèces dans la colonne "Species" en limitant le nombre de mots utilisés. Cette étape vise à réduire la complexité des données et à faciliter leur traitement ultérieur.

Nettoyage des données

```
In 27
1 def activity_split(text):
2     if "/" in text:
3         text = text.split("/")[0].strip()
4         return text
5
6     else:
7         return text
```

Normalisation de la colonne "Species"

Les transformations appliquées à la colonne "Species" suppriment les mots non informatifs pour obtenir des descriptions concises et cohérentes des espèces animales, facilitant ainsi l'analyse ultérieure.

Traitement des variations avec "or"

Cette fonction divise le texte en utilisant le mot "or" et conserve uniquement la première partie avant "or". Par exemple, si la description était "Tiger shark or Bull shark", après l'application de cette fonction, la description deviendra simplement "Tiger shark"

Nettoyage des données

```
In 29  1 def activity_finder(text):
       2     activities = ["Surfing", "Swimming", "Wading", "Spearfishing", "Stand
       3     for i in range(len(activities)):
       4         if activities[i] in text:
       5             return activities[i]
       6
       7     return "Other"
```

Fonction "activity_finder"

La fonction recherche des mots-clés spécifiques dans le texte pour assigner des activités correspondantes ou assigner la valeur "Other" si aucun mot-clé n'est trouvé, facilitant ainsi le regroupement des activités en catégories spécifiques pour une analyse ultérieure.

Application de la fonction "activity_finder"

Nous appliquons la fonction "activity_finder" à la colonne "Activity" en utilisant la méthode "apply". Cela permet d'assigner les activités correspondantes à chaque observation de la colonne, en remplaçant les descriptions plus détaillées par des catégories plus générales.

Nettoyage des données

```
In 32   1  df.head()
```

Out 33

5 rows × 6 columns

	Type	Activity	Sex	Age	Species	Fatal
6	Provoked	Other	F	20	unknown	N
12	Invalid	Surfing	F	18	unknown	N
23	Unprovoked	Surfing	M	28	tiger shark	N
33	Unprovoked	Swimming	F	17	unknown	N
64	Unprovoked	Other	F	58	tiger shark	N

- **Affichage des premières lignes du DataFrame**

  La fonction recherche des mots-clés spécifiques dans le texte pour assigner des activités correspondantes ou "Other" si aucun mot-clé n'est trouvé, simplifiant ainsi le regroupement des activités en catégories pour une analyse ultérieure.

- **Ces étapes de préparation des données sont importantes pour rendre les données plus adaptées à l'analyse et pour se concentrer sur les variables d'intérêt.**

Nettoyage des données

```
In 52  1 df["Sex"] = df["Sex"].replace({
       2     "lli": "F"
       3 })
```

Remplacement des valeurs de la colonne"Sex"

La fonction "replace" est utilisée pour remplacer les occurrences spécifiées dans la colonne "Sex". Dans ce cas, la valeur "lli" est remplacée par "F". Cela peut être utile pour corriger une erreur de saisie ou harmoniser les valeurs de la variable.

```
In 53  1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 869 entries, 6 to 6018
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Type      869 non-null    int32
 1   Activity  869 non-null    int32
 2   Sex       869 non-null    int32
 3   Age       869 non-null    int64
 4   Species   869 non-null    int32
 5   Fatal     869 non-null    int32
dtypes: int32(5), int64(1)
memory usage: 30.6 KB
```

Vérification de l'information du DataFrame

La fonction "info" est utilisée pour afficher des informations sur le DataFrame, notamment le nombre total d'entrées non nulles dans chaque colonne et le type de données de chaque colonne. Cela peut être utile pour vérifier si les modifications apportées aux données ont été appliquées correctement et si les types de données sont cohérents.

Preprocessing

```
In 43  1  from sklearn.preprocessing import LabelEncoder
       2
       3  def label_encoder(column):
       4      le = LabelEncoder().fit(column)
       5      print(column.name, le.classes_)
       6      return le.transform(column)

In 44  1  cols = ["Type", "Activity", "Sex", "Species", "Fatal"]
       2  for col in cols:
       3      df[col] = label_encoder(df[col])
```

Importation du module LabelEncoder

- Le code commence par l'importation de la classe LabelEncoder à partir du module sklearn.preprocessing. Cette classe est utilisée pour convertir des variables catégorielles en variables numériques.

Définition de la fonction label_encoder

Cette fonction prend une colonne (série) en entrée et applique l'encodage de l'étiquette en utilisant la classe LabelEncoder. L'encodage de l'étiquette attribue un entier unique à chaque catégorie présente dans la colonne. La méthode fit(column) de LabelEncoder est utilisée pour ajuster le codage sur la colonne donnée.

Transformation des colonnes sélectionnées

Une liste de noms de colonnes est définie dans la variable cols. Ensuite, une boucle for est utilisée pour itérer sur chaque nom de colonne dans cols. Pour chaque colonne, la fonction label_encoder est appelée avec la colonne correspondante de DataFrame df en tant qu'argument. La colonne est remplacée par les valeurs encodées retournées par la fonction

Preprocessing

```
In 56  1  X = df.drop("Fatal", axis=1)
       2  y = df["Fatal"]

In 57  1  from sklearn.model_selection import train_test_split
       2
       3  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Division des données

- La fonction train_test_split divise les données en ensembles d'entraînement et de test, où les caractéristiques et les cibles sont séparées, et 20% des données sont utilisées comme ensemble de test et 80% comme ensemble d'entraînement.

Fonctionnement de la mise à jour de la position et la vitesse

Le PSO utilise un ensemble de particules pour explorer l'espace de recherche en ajustant leurs positions et vitesses. Il s'inspire du comportement des essaims naturels pour trouver des solutions optimales à des problèmes d'optimisation.

Preprocessing

```
In 58  1  from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
       2
       3  lda = LinearDiscriminantAnalysis()
       4  X_train_lda = lda.fit_transform(X_train, y_train)
       5  X_test_lda = lda.transform(X_test)
       6
```

Importation du module et de la classe

- Le code commence par l'importation du module sklearn.discriminant_analysis, qui contient des outils pour effectuer des analyses discriminantes. La classe LinearDiscriminantAnalysis est importée à partir de ce module.

Réduction de dimensionnalité

La classe LinearDiscriminantAnalysis est utilisée pour ajuster un modèle sur les ensembles d'entraînement X_train et y_train, puis transformer les ensembles d'entraînement en utilisant l'analyse discriminante linéaire pour obtenir X_train_lda, et enfin transformer l'ensemble de test X_test pour obtenir X_test_lda.

En résumé:

L'analyse discriminante linéaire maximise la séparation entre les classes en trouvant les combinaisons linéaires des variables, ce qui réduit la dimensionnalité des données et améliore les performances des modèles d'apprentissage automatique.

Preprocessing

```
In 59  1  from sklearn.linear_model import LogisticRegression
       2
       3  logistic_regression = LogisticRegression()
       4  logistic_regression.fit(X_train_lda, y_train)
       5
```

```
Out 59  ▾ LogisticRegression
        LogisticRegression()
```

Importation du module et de la classe

- Le code commence par l'importation du module sklearn.discriminant_analysis, qui contient des outils pour effectuer des analyses discriminantes. La classe LinearDiscriminantAnalysis est importée à partir de ce module.

Création et entraînement du modèle

La méthode fit est utilisée pour ajuster un modèle de régression logistique sur les ensembles d'entraînement transformés X_train_lda et y_train, permettant de trouver la meilleure relation entre les variables indépendantes et la variable cible.

En résumé:

La régression logistique est une technique d'apprentissage supervisé utilisée pour prédire une variable binaire en se basant sur des caractéristiques transformées, permettant ainsi de prédire la variable cible pour de nouvelles données non vues.

Exécution du modèle

```
In 60  1  y_pred = logistic_regression.predict(X_test_lda)
       2
       3  from sklearn.metrics import accuracy_score, confusion_matrix
       4
       5  accuracy = accuracy_score(y_test, y_pred)
       6  cm = confusion_matrix(y_test, y_pred)
       7
```

Prédiction des classes

Le code utilise la méthode predict de l'objet logistic_regression pour effectuer des prédictions sur les caractéristiques d'essai transformées X_test_lda. Les prédictions sont stockées dans la variable y_pred.

Évaluation de la performance

Le code utilise la fonction accuracy_score pour calculer la précision du modèle en comparant les prédictions avec les vraies étiquettes, et la fonction confusion_matrix pour obtenir une matrice qui visualise les résultats des prédictions en termes de vrais positifs, vrais négatifs, faux positifs et faux négatifs.

En résumé:

La précision évalue la capacité du modèle à classifier correctement les données, tandis que la matrice de confusion fournit des informations détaillées sur les erreurs de classification effectuées par le modèle.

visualisation

```
In 62   1  # Visualize the confusion matrix
        2  plt.figure(figsize=(8, 6))
        3  sns.heatmap(cm, annot=True, fmt=".0f", cmap="Pastel2")
        4  plt.xlabel("Predicted")
        5  plt.ylabel("Actual")
        6  plt.title("Confusion Matrix")
        7  plt.show()
        8
        9  # Print the accuracy
       10  print("Accuracy:", accuracy)
```

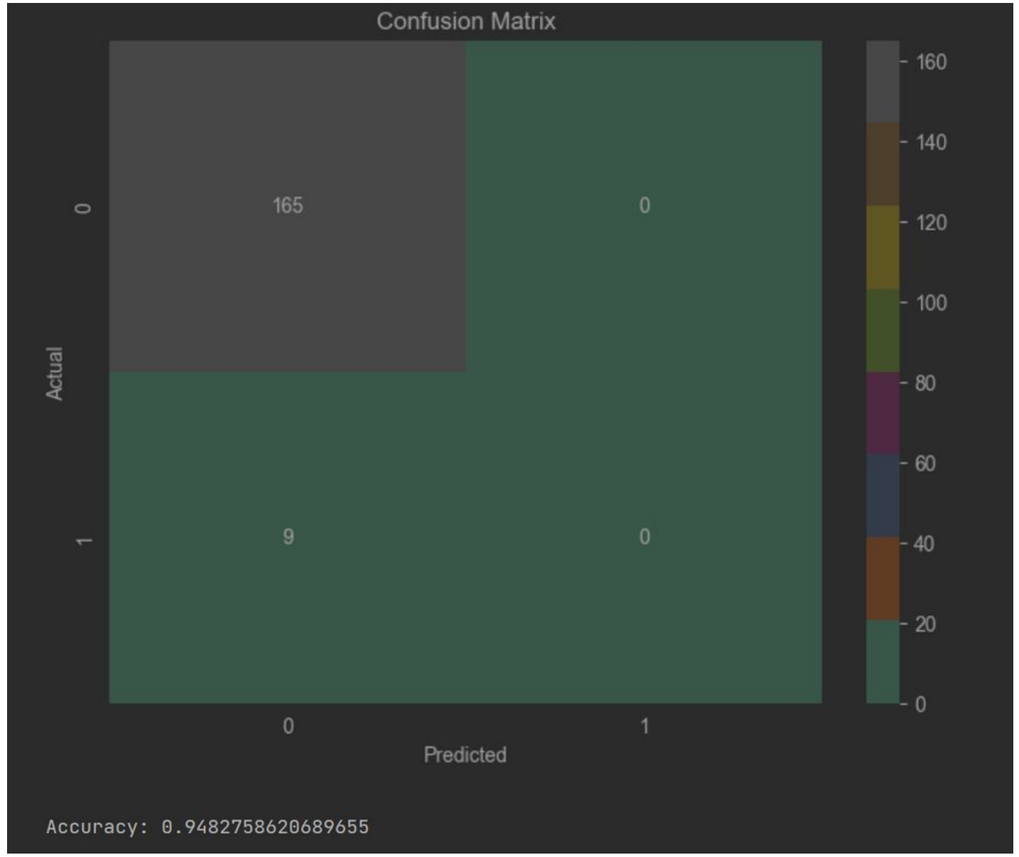
- **Visualisation de la matrice de confusion**

  Le code utilise la fonction heatmap de seaborn pour créer un diagramme de chaleur de la matrice de confusion, affichant les prédictions du modèle par rapport aux vraies étiquettes sous forme d'un tableau annoté avec des couleurs indiquant les performances du modèle.

- **Affichage de la précision**

  Le code utilise la fonction print pour afficher la précision du modèle, qui a été calculée précédemment et stockée dans la variable accuracy. La précision est affichée en tant que pourcentage et permet d'évaluer la performance globale du modèle.

Evaluation du modèle



Haute précision

L'exactitude de 0.94 indique que votre modèle de classification a réussi à prédire avec précision 94 % des cas dans votre ensemble de test. Cela témoigne de la capacité de votre modèle à bien généraliser et à effectuer des prédictions précises sur de nouvelles données

Performance solide

Une exactitude de 0.94 indique une performance solide, confirmant que le modèle peut distinguer précisément les catégories de la variable cible, renforçant ainsi la confiance dans ses prédictions pour des applications pratiques.

Remarque

L'exactitude seule ne donne pas une image complète de la performance du modèle ; il est recommandé de considérer d'autres métriques d'évaluation telles que la rappel, la précision, le score F1 ou la courbe ROC pour une évaluation plus complète.

Test du modèle

```
#dt = pd.DataFrame(data)
cols = ["Type", "Activity", "Sex", "Species"]
for col in cols:
    dt[col] = label_encoder(dt[col])
print(dt)
```

1ere etape

La fonction label_encoder est utilisée pour encoder les colonnes catégorielles du DataFrame dt en leur attribuant des valeurs numériques uniques, permettant ainsi de représenter les données sous une forme adaptée aux algorithmes d'apprentissage automatique.

```
#Appliquez le modèle de prédiction
X_new = dt

X_test_ld = lda.transform(X_new)

prediction = logistic_regression.predict(X_test_ld)

# Affichez la prédiction
print("Prediction:", prediction)
```

2eme etape

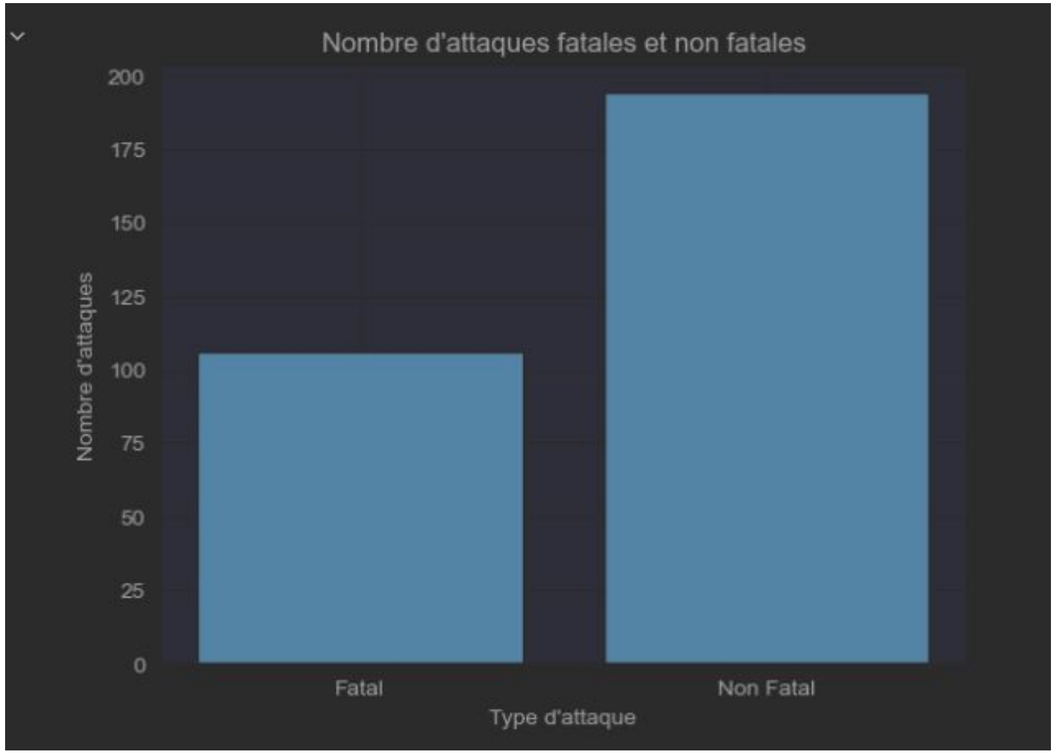
Le code utilise le modèle de régression logistique préalablement entraîné pour prédire les classes correspondantes sur de nouvelles données après avoir encodé les colonnes pertinentes du DataFrame dt et les avoir transformées à l'aide de l'analyse discriminante linéaire (LDA), et affiche la prédiction finale.

Affichage des résultats

1er affichage

La prédiction finale est affichée avec la mention "Prediction" suivie des valeurs prédites pour chaque donnée.

```
Prediction: [0 0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 1
 1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0
 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0
 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0
 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0
 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 0
 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1
 0 0 0 0]
```



2ème affichage

En utilisant la fonction np.count_nonzero, le code compte le nombre d'occurrences de prédictions "Fatal" et "Non Fatal" dans la variable "prediction", puis crée un graphique à barres personnalisé affichant le nombre d'attaques fatales et non fatales.

Avantages et Limitations

Avantages

Haute précision ce qui indique sa capacité à effectuer des prédictions précises,
impénétrabilité permet de comprendre les facteurs qui influencent les prédictions du modèle

Limitations

Dépendance des données d'entrée, Sensibilité aux hypothèses, Généralisation limitée.

CONCLUSION

Merci
pour votre
attention!