```
##Visualisation de données
import seaborn as sns
#Analyse de données
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
#Manipulation des matrices
import numpy as np
# une classe pour encoder des variables catégorielles
from sklearn.preprocessing import OneHotEncoder
#Visualisation statique
import matplotlib.pyplot as plt
 # une classe pour encoder des variables catégorielles
from sklearn.preprocessing import LabelEncoder
 # Une classe pour transformer des colonnes
from sklearn.compose import ColumnTransformer
# pour mettre à l'échelle vos caractéristiques numériques
from sklearn.preprocessing import StandardScaler
# Importer les metriques
from sklearn.metrics import accuracy_score, precision_score,
recall_score ,f1_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
# Charger le dataset Titanic depuis seaborn
titanic = sns.load_dataset('titanic')
#Charger le dataset dapre un fichier excel ou csv
df=pd.read_csv("seattle-weather.csv")
# Afficher les 5 premières lignes
print(titanic.head())
# Afficher les 10 dernières lignes du DataFrame
print(df.tail(10))
##Afficher les entete des colonnes de dataset
print(titanic.columns)
#afficher le type de chaque colonne
print(titanic.dtypes)
#verifier si le dataset contient des valeur null et compter les
titanic.isnull().sum()
# le nombre total de valeurs manquantes. Interprétez les résultats.
titanic.isnull().sum().sum(
#pour supprimer les ligne qui contient les valeurs null
df1 = titanic.dropna()
#pour supprimer les colonnes qui contient les valeurs null
df=titanic.dropna(axis=1)
#pour afficher le nombre de ligne et de colonnes(dimension)
df.shape
#Afficher les types de colonnes
print(df.dtypes)
# Afficher des informations générales
df.info()
# Afficher une description statistique
df.describe()
#supprimer une ligne ou bien colonne
df2 = titanic.drop('nom du ligne /colonne', axis=1/colonne 0/ligne)
#créer un dataframe une partie du dataset contenant les variables
age et fare
df3=titanic[['age','fare']] #si characteristique double [[ee]]
#remplacer les valeurs manquantes de la variable age par médiane
median = df3['age'].median()
df3['age'].fillna(median, inplace=True)
#importer et instancier MinMaxScaler et appliquer MinMaxScaler
scaler = MinMaxScaler()
scaler_fit=scaler.fit_transform(df3)
df3_scaled = pd.DataFrame(scaler_fit)
# Instanciation du One-Hot Encoder
encoder = OneHotEncoder(sparse_output=False)
# Application du One-Hot Encoding sur la colonne 'month'
one_hot_encoded = encoder.fit_transform(df[['month']])
# Affichage de one_hot_encoded et de son type
print(one_hot_encoded,type(one_hot_encoded))
# Séparation des données (feature & target)
X = df[['Salary']]
y = df['YearsExperience']
# Division en un ensemble d'entrainement et un
ensemble de test
X_train, X_test,y_train,y_test=train_test_split(X,y,test_size
=0.3,random_state=42)
#Afficher tout les element (infinite)
np.set_printoptions(threshold=np.inf)
print(one_hot_encoded,type(one_hot_encoded))
# Création d'un DataFrame des colonnes encodées
encoded_df=pd.DataFrame(one_hot_encoded,columns=e
ncoder.get_feature_names_out(['month']))
# Fusionner les colonnes encodées avec le DataFrame
original sans la variable 'month'
df_encoded = pd.concat([df.drop('month', axis=1),
encoded_df], axis=1)
#tracer des histogrammes pour les colonnes numériques
for column in df.columns[:-1]: # Exclure la colonne
'species'
 plt.figure(figsize=(8, 4))
 sns.histplot(df[column], kde=True)
 plt.title(f'Histogramme de {column}')
 plt.xlabel(column)
 plt.ylabel('Fréquence')
 plt.show()
#Calcule du Moyenne
df[colonne].mean()
#Calcul median
df[colonne].
#Calcul des Quartiles
   Q1 = df[colonne].quantile(0.25)
   Q2 = df[colonne].median()
   Q3 = df[colonne].quantile(0.75)
#Calcul du varience
df[colonne].var()
#Calcul d'ecart type
df[colonne].std()
#  Calcul de l'Étendue Interquartile (IQR)
Q3-Q1
#Définition des limites inférieure et supérieure pour les
valeurs aberrantes
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q1 + 1.5 * IQR
## Filtrage des données pour ne conserver que les
années d'experience qui se trouvent entre les limites
inférieure et supérieure
data = df[(df['YearsExperience'] >= lower_bound) &
(df['YearsExperience'] <= upper_bound)]
#Détection des valeurs aberrantes
for column in df.columns[:-1]: # Exclure la colonne
'species'
   plt.figure(figsize=(8, 4))
   sns.boxplot(x=df[column])
   plt.title(f'Boxplot de {column}')
   plt.xlabel(column)
# statistique data object  type object "qualitatif"
df.describe(include=['object'])
# Analyse de la Variable Cible
df['weather'].value_counts()
# Supprimez la colonne date
del df["date"]
#Encodez la variable weather en valeurs numériques
le=LabelEncoder()
df["weather_encode"]=le.fit_transform(df["weather"])
#matrice de correlation
cor=df.drop(['weather'],axis=1).corr()
sns.heatmap(cor,annot=True)
```

```
# KNeighborsClassifier avec n_neighbors = 5
KNN = KNeighborsClassifier(n_neighbors=5)
#Les variables de testes pour les arbre random=true
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
#entrainer ainsi pour les arbres et regression lineare
KNN.fit(X_train, y_train)
#calculer prediction y_pred
y_pred = KNN.predict(X_test)
#score
KNN.score(X_test, y_test)
# y_test converti en np.ndarray pour l'affichage
print(np.array(y_test))
# Create a confusion matrix
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
# Visualiser la matrice de confusion avec un heatmap
plt.figure(figsize=(7, 5), dpi=100)
sns.heatmap(conf_matrix, annot=True, fmt="d",xticklabels=['Rain', 'Sun', 'Fog', 'Drizzle', 'Snow'],yticklabels=['Rain', 'Sun', 'Fog', 'Drizzle', 'Snow'])
plt.ylabel('Véritables catégories')#titre pour l'axe des y
plt.xlabel('Prédictions')#titre pour l'axe des x
plt.title('Matrice de Confusion pour les Prédictions Météo')#titre matrice
plt.show()#affichage matrice
#Affihcer Précision, rappel et F1-score pour KNN
print(classification_report(y_test, y_pred))
############################################################
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': range(1, 20), 'metric':['euclidean', 'manhattan', 'minkowski']}
# Application de GridSearchCV avec validation croisée
grid_KNN = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_KNN.fit(X_train, y_train)
print(grid_KNN.best_params_)
# calcul de la nouvelle y_pred
Final_model = KNeighborsClassifier(metric ='manhattan', n_neighbors = 11, weights = 'distance')
# Performance du modèle optimal
Final_model = grid_KNN.best_estimator
# Meilleur k trouvé
best_k = grid_KNN.best_params_['n_neighbors']
#hyperparametre du decisiontree c'est max_depth et l'indice de genie plusier indice dont dispo
model=DecisionTreeClassifier(criterion="gini",max_depth=5)
#imports des biblo pour visualiser l'arbre
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
# Visualisation de l'arbre de décision
plt.figure(figsize=(50,15), dpi = 200)  # Augmenter la taille de la figure
plot_tree(model,filled=True,feature_names=X_train.columns,#
feature_names c'est le nom des caractéristiques
class_names=["Classe 0", "Classe 1", "Classe 2", "Classe 3", "Classe 4"], rounded=True,fontsize=14)
# font_sizeAugmenter la taille de la police
# class_name c'est le noms des classes
plt.tight_layout()
plt.savefig("tree_model.png", dpi = 200)
plt.show() #affichage
#import des metriques pour la regression lineare
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,root_mean_squared_error
#pour le modele du regression lineare
from sklearn.linear_model import LinearRegression
#check for duplicate rows et sum pour voire la somme des ligne dups
housing.duplicated().sum()
#visualisation des points
plt.scatter(X_train['Area'], y_train, s=1)#nuage de points#['Area']si on a plusieur feature dans le X_train
plt.title('Price vs Area')
plt.xlabel('Area')
plt.ylabel('Price')
#entrener le modele
lr = LinearRegression()
lr.fit(X_train, y_train)
# Afficher les paramètres du modèle donner l'expression du polynome h(x)
print(lr.intercept_)#valeur du theta zero
print(lr.coef_)#coef de l'equation de regression theta 1,2..
#pour tracer la droite on ajoute ses 2 ligne a la partie du visualisation apres plt.scatter
y_pred = lr.predict(X_test)#predire les valeur en utilisant le vecteur
plt.plot(X_test['Area'], y_pred, color='red')#tracer la droite
#pour clalculer l'erreur
MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)
RMSE=root_mean_squared_error(y_test, y_pred)#import root_mean_squared_error
r2 = r2_score(y_test, y_pred)
#Diviser la base de données en deux tableaux Xm (tableau d'observations) et ym (la variable cible)
Xm = boston.iloc[:,0:12]#:pour toute les ligne 0:12 pour les colonnes iloc pour extraire les colonnes sans les nommees
ym = boston['MEDV']#derniere colonnes du target
#standardisation
# Initialize StandardScaler
scaler = StandardScaler()#import bilbo
Xm_train_sc=scaler.fit_transform(Xm_train)
Xm_test_sc = scaler.transform(Xm_test)
Tracer les valeurs réelles et les prédictions sur l'ensemble de test
# Visualisation
plt.scatter(X_test , y_test, color='blue', label='Valeurs réelles')
plt.plot(X_test, y_pred, color='red', label='Ligne de régression')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Régression linéaire simple')
plt.legend()
plt.show()
#Tracer l'arbre de décision pour le model: final_model
plt.figure(figsize=(12, 8))
plot_tree(final_model, filled=True,
feature_names=X_train.columns, class_names=['0 (Bénin)', '1 (Malin)'], rounded=True)
plt.title('Arbre de Décision')
plt.show()
```