import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_auc\_score

# Génération de données hypothétiques

np.random.seed(42)

num\_samples = 1000

data = pd.DataFrame({

'Salaire': np.random.randint(30000, 100000, size=num\_samples),

'Target': np.random.choice([1, 2], size=num\_samples)

})

# Fonction pour calculer le taux de risque et l'IV pour une classe donnée

def calculate\_risk\_and\_iv(data, class\_var, target\_var):

class\_data = data.groupby(class\_var)[target\_var].agg(['mean', 'count', 'sum'])

class\_data.columns = ['mean\_target', 'total\_count', 'positive\_count']

class\_data['negative\_count'] = class\_data['total\_count'] - class\_data['positive\_count']

class\_data['TR'] = class\_data['positive\_count'] / class\_data['total\_count']

class\_data['IV'] = (class\_data['mean\_target'] - (1 - class\_data['mean\_target'])) \* np.log(

(class\_data['mean\_target'] + 0.0001) / (1 - class\_data['mean\_target'] + 0.0001))

return class\_data

# Découpage du salaire en classes (par exemple, 3 classes)

data['Salaire\_Class'] = pd.qcut(data['Salaire'], q=3, labels=['Bas', 'Moyen', 'Haut'])

# Calcul du taux de risque et de l'IV

risk\_iv\_data = calculate\_risk\_and\_iv(data, 'Salaire\_Class', 'Target')

# Vérification de la monotonie

is\_monotonic = all(risk\_iv\_data['TR'].diff().dropna() >= 0) or all(risk\_iv\_data['TR'].diff().dropna() <= 0)

# Tracé du graphique

plt.figure(figsize=(10, 6))

plt.bar(risk\_iv\_data.index, risk\_iv\_data['IV'])

plt.title("Information Value (IV) en Fonction des Classes de Salaire")

plt.xlabel("Classes de Salaire")

plt.ylabel("IV Value")

plt.xticks(rotation=45)

plt.tight\_layout()

# Affichage du résultat

if is\_monotonic:

print("Les classes de salaire sont monotones en termes de taux de risque (TR).")

else:

print("Les classes de salaire ne sont pas monotones en termes de taux de risque (TR).")

plt.show()

-gggggggggg

import pandas as pd

import numpy as np

from sklearn.model\_selection import KFold

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

# Génération de données hypothétiques

np.random.seed(42)

num\_samples = 1000

data = pd.DataFrame({

'Variable': np.random.rand(num\_samples) \* 100,

'Target': np.random.choice([0, 1], size=num\_samples)

})

# Définition des différentes valeurs de q à tester

quantile\_values = [5, 10, 15, 20, 25]

# Initialisation du meilleur IV, du meilleur F1-score et du meilleur q

best\_iv = 0

best\_f1\_score = 0

best\_q = None

# Validation croisée pour chaque valeur de q

for q in quantile\_values:

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

iv\_values = []

f1\_scores = []

for train\_idx, test\_idx in kf.split(data):

train\_data, test\_data = data.iloc[train\_idx], data.iloc[test\_idx]

# Binning des variables avec pd.qcut

train\_data['Binned\_Variable'], bins = pd.qcut(train\_data['Variable'], q=q, labels=False, retbins=True)

test\_data['Binned\_Variable'] = pd.cut(test\_data['Variable'], bins=bins, labels=False)

# Entraînement d'un modèle (par exemple, une régression logistique)

X\_train = train\_data[['Binned\_Variable']]

y\_train = train\_data['Target']

X\_test = test\_data[['Binned\_Variable']]

y\_test = test\_data['Target']

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Prédiction et calcul du F1-score

y\_pred = model.predict(X\_test)

f1 = f1\_score(y\_test, y\_pred)

f1\_scores.append(f1)

# Calcul de l'IV

y\_proba = model.predict\_proba(X\_test)[:, 1]

iv = roc\_auc\_score(y\_test, y\_proba)

iv\_values.append(iv)

# Calcul du F1-score moyen et de l'IV moyen pour cette valeur de q

avg\_f1\_score = np.mean(f1\_scores)

avg\_iv = np.mean(iv\_values)

# Vérification de la monotonie du risque d'erreur

is\_monotonic = all(f1\_scores[i] <= f1\_scores[i + 1] for i in range(len(f1\_scores) - 1))

# Mise à jour du meilleur q si nécessaire (monotonie et IV élevé)

if is\_monotonic and avg\_iv > best\_iv:

best\_iv = avg\_iv

best\_f1\_score = avg\_f1\_score

best\_q = q

print(f"La meilleure valeur de q est : {best\_q}")

print(f"F1-score moyen associé : {best\_f1\_score:.2f}")

* print(f"IV moyen associé : {best\_iv:.2f}")