```
# Chargement des données
import pandas as pd
df = pd.read_csv('./MP-4MLSP.csv')
```

Question a : Créer une variable binaire à partir de la variable "Satisfaction":

Nous allons créer une nouvelle variable binaire à partir de la variable "Satisfaction" en regroupant les classes "Neutral" et "dissatisfied" en une seule classe. Cette nouvelle variable indiquera si un client est "satisfied" ou "not satisfied".

```
# Créer une copie du DataFrame original pour éviter de modifier les données originales
df_binary = df.copy()

# Créer la nouvelle variable binaire "Satisfaction_Binary"
df_binary['Satisfaction_Binary'] = df_binary['satisfaction'].apply(lambda x: 1 if x == 'satisfied' else 0)

# Vérifier la distribution de la nouvelle variable binaire
print(df_binary['Satisfaction_Binary'].value_counts())
```

```
Satisfaction_Binary
0    73452
1    56428
Name: count, dtype: int64
```

Question b : Reprendre les étapes de la question 2 avec cette nouvelle variable comme variable cible

Nous allons reprendre les étapes de prétraitement et d'application des algorithmes de classification en utilisant cette nouvelle variable binaire "Satisfaction_Binary" comme variable cible.

Étape 1 : Prétraitement des Données

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer

# Sélection des variables explicatives
X = df_binary.drop(['satisfaction', 'Satisfaction_Binary'], axis=1)
y = df_binary['Satisfaction_Binary']

# Encoder les variables catégorielles
categorical_cols = X.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in categorical_cols:
```

```
    X[col] = label_encoder.fit_transform(X[col])

# Imputation des valeurs manquantes avec la médiane
imputer = SimpleImputer(strategy='median')
X = imputer.fit_transform(X)

# Normalisation des variables
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Étape 2 : Application des Algorithmes de Classification

Nous allons appliquer plusieurs algorithmes de classification en utilisant la nouvelle variable binaire comme variable cible, et évaluer les performances de chaque modèle.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report,
roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# Définition des hyperparamètres pour GridSearchCV
param_grid_lr = {'C': [0.1, 1, 10, 100]}
param_grid_rf = {'n_estimators': [100, 200, 300], 'max_depth': [None,
10, 20, 30]}
param_grid_svc = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
param_grid_knn = {'n_neighbors': [3, 5, 7, 9]}

# Initialisation des modèles
models = {
    'Logistic Regression': (LogisticRegression(), param_grid_lr),
    'Random Forest': (RandomForestClassifier(), param_grid_rf),
    'Support Vector Classifier': (SVC(probability=True),
param_grid_svc),
    'K-Nearest Neighbors': (KNeighborsClassifier(), param_grid_knn)
}

# Application des modèles et évaluation des performances
results = {}

for model_name, (model, param_grid) in models.items():
    grid_search = GridSearchCV(model, param_grid, cv=5,
scoring='accuracy')
```

```
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    # Vérifier si le modèle supporte predict_proba, sinon utiliser
decision_function
    if hasattr(best_model, "predict_proba"):
        y_proba = best_model.predict_proba(X_test)[:, 1]
    elif hasattr(best_model, "decision_function"):
        y_proba = best_model.decision_function(X_test)
        y_proba = (y_proba - y_proba.min()) / (y_proba.max() -
y_proba.min())  # Normaliser pour avoir des valeurs entre 0 et 1
    else:
        y_proba = None

    cm = confusion_matrix(y_test, y_pred)
    cr = classification_report(y_test, y_pred)

    if y_proba is not None:
        auc = roc_auc_score(y_test, y_proba)
    else:
        auc = None

    results[model_name] = {
        'Best Model': best_model,
        'Confusion Matrix': cm,
        'Classification Report': cr,
        'AUC': auc
    }

    print(f"Model: {model_name}")
    print(f"Best Parameters: {grid_search.best_params_}")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Classification Report:\n{cr}")
    if auc is not None:
        print(f"AUC: {auc}")
    print("\n" + "="*60 + "\n")

Model: Logistic Regression
Best Parameters: {'C': 10}
Confusion Matrix:
[[13258  1400]
 [ 1934  9384]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.90      0.89     14658
           1       0.87      0.83      0.85     11318
```

```
    accuracy                           0.87     25976
   macro avg       0.87      0.87      0.87     25976
weighted avg       0.87      0.87      0.87     25976

AUC: 0.9248036778274891

============================================================
```