```python
import pandas as pd
from fuzzywuzzy import fuzz

# Lecture du fichier CSV
dfA = pd.read_csv('D:/BUT/Annee 1/SAE 1-02 Ecriture et Lecture de Fichiers de DonnÃ©es/DonnÃ©es Projet/fichierClientA.csv')

# Ouverture/crÃ©ation d'un fichier pour enregistrer les erreurs
open("erreur.txt", "w").close()


def fichier_erreur(ecrit):
    """
    Cette fonction Ã©crit les erreurs dans un fichier
    """
    with open("erreur.txt", "a") as fichier:
        fichier.write(ecrit + "\n")


def remplace2(diplome, colonne):
    """
    Cette fonction calcule la mÃ©diane de la colonne spÃ©cifiÃ©e pour les lignes
    ayant le diplÃ´me spÃ©cifiÃ©.
    """
    s = list(dfA)
    bb = dfA[dfA["Education"] == diplome][[s[colonne]]]
    return int(bb.median())


def remplace(col, j):
    """
    Cette fonction calcule la mÃ©diane pour les lignes ayant une valeur non nulle
    dans la colonne spÃ©cifiÃ©e.
    """
    return dfA[dfA[col].notnull()][dfA.columns[j]].median()


def calculer_medianes(df):
    """
    Cette fonction calcule les mÃ©dianes pour les colonnes "Income", "Nombre_commandes_Tot"
    et "Depences_Tot" en fonction de "Marital_Status"
    """
    # Calculer les mÃ©dianes par statut matrimonial
    med_income = df.groupby("Marital_Status")["Income"].median().to_dict()
    med_nombre_commandes = df.groupby("Marital_Status")["Nombre_commandes_Tot"].median().to_dict()
    med_depences = df.groupby("Marital_Status")["Depences_Tot"].median().to_dict()
    return med_income, med_nombre_commandes, med_depences


def trouver_statut_proche_mediane(x, medianes):
    """
    Cette fonction trouve le statut matrimonial avec la mÃ©diane la plus proche de x
    """
    # Trouver le statut matrimonial avec la mÃ©diane la plus proche de x
    lis_dis_med = {}
    lis_dis_med = {(b - x) ** 2: a for a, b in medianes.items()}
    return lis_dis_med[min(lis_dis_med.keys())]


def remplace_statB(row):
    """
    Cette fonction remplace les valeurs manquantes par la mediane en fonction du statut marital
    """

    # Fonction pour remplacer les valeurs manquantes par la mediane en fonction du statut marital
    if pd.isnull(row['Income']):
        row['Income'] = Med_RevB[row['Marital_Status']]
    if pd.isnull(row['Nombre_commandes_Tot']):
        row['Nombre_commandes_Tot'] = Med_ComB[row['Marital_Status']]
    if pd.isnull(row['Depences_Tot']):
        row['Depences_Tot'] = Med_DepB[row['Marital_Status']]
    return row


def trouver_statut_commun(x, y, z, medianes_income, medianes_nombre_commandes, medianes_depences):
    """
    Cette fonction trouve le statut matrimonial commun entre les medianes des colonnes "Income", "Nombre_commandes_Tot"
    et "Depences_Tot" en utilisant la fonction "trouver_statut_proche_mediane"
    """
    aa = trouver_statut_proche_mediane(x, medianes_income)
    bb = trouver_statut_proche_mediane(y, medianes_nombre_commandes)
    cc = trouver_statut_proche_mediane(z, medianes_depences)

    if aa == bb:
        return aa
    elif aa == cc:
        return aa
    elif bb == cc:
        return bb
    else:
        return aa


def remplaceB(diplome, colonne):
    """
    Cette fonction calcule la mÃ©diane pour la colonne spÃ©cifiÃ©e en fonction du diplÃ´me spÃ©cifiÃ©
    """
    return dfB[dfB["Education"] == diplome][colonne].median()


def remplacer_nan_par_mediane(df):
    """
    Cette fonction remplace les valeurs nulles dans la colonne "Marital_Status" par la mÃ©diane commune
    trouvÃ©e Ã  l'aide des colonnes "Income", "Nombre_commandes_Tot" et "Depences_Tot"
    """
    medianes_income, medianes_nombre_commandes, medianes_depences = calculer_medianes(df)
    for i in range(len(df)):
        if pd.isnull(df.iloc[i, 3]):
            df.iloc[i, 3] = trouver_statut_commun(df.iloc[i, 6], df.iloc[i, 22], df.iloc[i, 23], medianes_income,
                                                  medianes_nombre_commandes, medianes_depences)


def corriger_erreurs(df, erreurs):
    """
    Cette fonction corrige les erreurs en remplaÃ§ant les valeurs incorrectes par celles spÃ©cifiÃ©es
```

```
    dans la liste "erreurs" et en enregistrant les erreurs dans le fichier "erreur.txt"
    """
    for ligne, colonne, valeur, message in erreurs:
        df.iloc[ligne, colonne] = valeur
        fichier_erreur(message)


erreurs = [
    (9, 1, 1950, "erreur date: ligne 9 colonne 1"),
    (9, 2, "PhD", "erreur Typos: ligne 9 colonne 2"),
    (139, 1, 1965, "erreur date: ligne 139 colonne 1"),
    (139, 3, "Divorced", "erreur Typos: ligne 139 colonne 3"),
    (224, 1, 1962, "erreur date: ligne 224 colonne 1"),
    (224, 3, "Divorced", "erreur Typos: ligne 224 colonne 3"),
    (230, 1, 1966, "erreur date: ligne 230 colonne 1"),
    (230, 2, "Graduation", "erreur Typos: ligne 230 colonne 2"),
    (310, 1, 1976, "erreur date: ligne 310 colonne 1"),
    (310, 2, "Graduation", "erreur Typos: ligne 310 colonne 2"),
    (310, 3, "Divorced", "erreur Typos: ligne 310 colonne 3"),
    (61, 2, "Graduation", "erreur Typos: ligne 61 colonne 2"),
    (61, 3, "Together", "erreur Typos: ligne 61 colonne 3"),
    (217, 2, "Master", "erreur Typos: ligne 217 colonne 2"),
    (217, 3, "Married", "erreur Typos: ligne 217 colonne 3"),
    (740, 2, "Phd", "erreur Typos: ligne 740 colonne 2"),
    (740, 3, "Single", "erreur Typos: ligne 740 colonne 3"),
]

# Appel des fonctions pour corriger les erreurs et remplacer les valeurs nulles par les mÃ©dianes
corriger_erreurs(dfA, erreurs)

# Convertir la colonne "Year_Birth" en type entier
dfA["Year_Birth"] = dfA["Year_Birth"].astype(int)

# Ajouter 1900 aux valeurs de la colonne "Year_Birth" infÃ©rieures Ã  100
dfA.loc[dfA["Year_Birth"] < 100, "Year_Birth"] += 1900

# Enregistrer une erreur dans le fichier erreur.txt pour la colonne "Year_Birth"
fichier_erreur("erreur date: colonne Year_Birth")

# dfA['Year_Birth'] = pd.to_datetime(dfA['Year_Birth'], format='%Y')

# Capitaliser la premiÃ¨re lettre de la colonne "Marital_Status" et "Education"
column_names = dfA.columns.tolist()
dfA[column_names[2]] = dfA[column_names[2]].apply(lambda x: x.capitalize() if isinstance(x, str) else x)
dfA[column_names[3]] = dfA[column_names[3]].apply(lambda x: x.capitalize() if isinstance(x, str) else x)

# Dictionnaire des fautes d'orthographe Ã  corriger dans la colonne "Marital_Status"
typos = {"Togeter": "Together", "Maried": "Married", "Alone": "Single"}

# Appliquer la correction des fautes d'orthographe et capitaliser la premiÃ¨re lettre de la colonne "Marital_Status"
dfA.iloc[:, 2] = dfA.iloc[:, 2].apply(lambda x: str(x).capitalize() if pd.notnull(x) else x)
dfA.iloc[:, 2] = dfA.iloc[:, 2].apply(lambda x: typos[x] if x in typos else x)

# Enregistrer une erreur dans le fichier erreur.txt pour les lignes contenant des fautes d'orthographe corrigÃ©es
dfA.apply(lambda x: fichier_erreur(f"erreur Typo: ligne {x.name} colonne 3") if x[2] in typos else None, axis=1)

# Liste des corrections possibles pour la colonne "Education"
corrections = ["PhD", "Graduation", "Master", "2n Cycle"]

# Capitaliser la premiÃ¨re lettre de la colonne "Education"
dfA.iloc[:, 2] = dfA.iloc[:, 2].apply(lambda x: str(x).capitalize() if pd.notnull(x) else x)

# Appliquer la correction la plus similaire possible pour les valeurs de la colonne "Education" en utilisant FuzzyWuzzy
dfA.iloc[:, 2] = dfA.iloc[:, 2].apply(lambda x: max(corrections, key=lambda correction: fuzz.token_set_ratio(x, correction)) if pd.notnull(x) else x)

# Ajouter une nouvelle colonne "nombre_personne_foyer" qui est la somme des colonnes "Kidhome" et "Teenhome"
dfA['nombre_personne_foyer'] = dfA['Kidhome'].add(dfA['Teenhome'])

# Ajouter 2 aux valeurs de la colonne "nombre_personne_foyer" pour les lignes oÃ¹ la colonne "Marital_Status" est "Together" ou "Married"
mask = dfA['Marital_Status'].isin(['Together', 'Married'])
dfA.loc[mask, 'nombre_personne_foyer'] = dfA.loc[mask, 'nombre_personne_foyer'].add(2)

# Ajouter 1 aux valeurs de la colonne "nombre_personne_foyer" pour les lignes oÃ¹ la colonne "Marital_Status" n'est pas "Together" ou "Married"
mask = dfA['Marital_Status'].isin(['Together', 'Married'])
dfA.loc[~mask, 'nombre_personne_foyer'] = dfA.loc[~mask, 'nombre_personne_foyer'].add(1)

# Enregistrer une erreur pour l'ajout de la colonne "nombre_personne_foyer"
fichier_erreur("Ajout de nouvelle colonne 'nombre_personne_foyer'")

# Ajouter une nouvelle colonne "Revenu_moyen" qui est le rapport entre la colonne "Income" et la colonne "nombre_personne_foyer"
dfA['Revenu_moyen'] = dfA['Income'].div(dfA['nombre_personne_foyer'])
fichier_erreur("Ajout de nouvelle colonne 'Revenu_moyen'")

# Ajouter une nouvelle colonne "Depences_Tot" qui est la somme des colonnes "MntFruits", "MntMeatProducts", "MntWines", "MntFishProducts", "MntSweetProducts" et "MntGoldProds"
dfA['Nombre_commandes_Tot'] = dfA[
    ['NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases']].sum(axis=1)

fichier_erreur("Ajout de nouvelle colonne 'Nombre_commandes_Tot'")

# Enregistrer une erreur pour l'ajout de la colonne "Depences_Tot"
dfA['Depences_Tot'] = dfA[
    ['MntFruits', 'MntMeatProducts', 'MntWines', 'MntFishProducts', 'MntSweetProducts', 'MntGoldProds']].sum(axis=1)


fichier_erreur("Ajout de nouvelle colonne 'Depences_Tot'")

# Ajouter une nouvelle colonne "Quantite_depensee_moyenne_commande" qui est le rapport entre la colonne "Depences_Tot" et la colonne "Nombre_commandes_Tot"
dfA['Quantite_depensee_moyenne_commande'] = dfA['Depences_Tot'].div(dfA['Nombre_commandes_Tot'])

# Remplacer les valeurs manquantes de la colonne "Quantite_depensee_moyenne_commande" par 0
dfA['Quantite_depensee_moyenne_commande'].fillna(0, inplace=True)
# Enregistrer une erreur pour l'ajout de la colonne "Quantite_depensee_moyenne_commande"
fichier_erreur("Ajout de nouvelle colonne 'Quantite_depensee_moyenne_commande'")

# Ajouter une nouvelle colonne "Quantite_depensee_moyenne_personne" qui est le rapport entre la colonne "Depences_Tot" et la colonne "nombre_personne_foyer"
dfA['Quantite_depensee_moyenne_personne'] = dfA['Depences_Tot'].div(dfA['nombre_personne_foyer'])

# Enregistrer une erreur pour l'ajout de la colonne "Quantite_depensee_moyenne_personne"
fichier_erreur("Ajout de nouvelle colonne 'Quantite_depensee_moyenne_personne'")

# Remplacer les valeurs manquantes de la colonne "Income" par la mÃ©diane en fonction de l'Education
dfA['Income'].fillna(remplace('Education', 6), inplace=True)

remplacer_nan_par_mediane(dfA)
```

```
# Liste des colonnes Ã  traiter
colonnes_a_traiter = [6] + list(range(8, len(list(dfA))))

# Parcours des colonnes Ã  traiter
for colonne in colonnes_a_traiter:
    # Parcours de chaque ligne de la colonne
    for i in range(len(dfA)):
        # Si la valeur est manquante, on la remplace par la mÃ©diane
        if pd.isnull(dfA.iloc[i, colonne]):
            dfA.iloc[i, colonne] = remplace2(dfA.iloc[i, 2], colonne)
            fichier_erreur(
                f"remplace valeur manquante par {int(remplace2(dfA.iloc[i, 2], colonne))} ligne {i} colonne {colonne}fichier A")

################################################################################

dfB = pd.read_csv(
    'D:/BUT/Annee 1/SAE 1-02 Ecriture et Lecture de Fichiers de DonnÃ©es/DonnÃ©es Projet/fichierClientB.csv')


# convert years in the 'Year' column to 4-digit years
dfB['Year_Birth'] = dfB['Year_Birth'].apply(lambda x: x + 1900 if x < 100 else x)

# standardize the values in the 'Marital_Status' column
dfB['Marital_Status'] = dfB['Marital_Status'].str.capitalize()
dfB['Marital_Status'].replace({'Togeter': 'Together', 'Maried': 'Married'}, inplace=True)

# standardize the values in the 'Education' column
dfB['Education'].replace({'Phd': 'PhD', 'Gradation': 'Graduation', 'Mastere': 'Master', '2n cycle': '2n Cycle', 'Basique': 'Basic'}, inplace=True)

# add a new column 'nombre_personne_foyer'
dfB['nombre_personne_foyer'] = dfB['Kidhome'] + dfB['Teenhome']

# update the value of 'nombre_personne_foyer' based on the value of 'Marital_Status'
mask = dfB['Marital_Status'].isin(['Together', 'Married'])
dfB.loc[mask, 'nombre_personne_foyer'] += 2
dfB.loc[~mask, 'nombre_personne_foyer'] += 1


# Calcul du revenu moyen par personne dans le foyer
dfB['Revenu_moyen'] = dfB['Income'] / dfB['nombre_personne_foyer']

# Calcul du nombre total de commandes
dfB['Nombre_commandes_Tot'] = dfB['NumDealsPurchases'] + dfB['NumWebPurchases'] + dfB['NumCatalogPurchases'] + dfB['NumStorePurchases']

# Calcul du montant total dÃ©pensÃ©
dfB['Depences_Tot'] = dfB['MntFruits'] + dfB['MntMeatProducts'] + dfB['MntWines'] + dfB['MntFishProducts'] + dfB['MntSweetProducts'] + dfB['MntGoldProds']

# Calcul de la quantitÃ© dÃ©pensÃ©e en moyenne par commande
dfB['Quantite_depensee_moyenne_commande'] = dfB.iloc[:, 23] / dfB.iloc[:, 22].where(dfB.iloc[:, 22] != 0, other=1)

# Calcul de la quantitÃ© dÃ©pensÃ©e en moyenne par personne dans le foyer
dfB['Quantite_depensee_moyenne_personne'] = dfB['Depences_Tot'] / dfB['nombre_personne_foyer']

# Remplacement des valeurs manquantes dans les colonnes du DataFrame par la mÃ©diane des valeurs correspondantes
dfB['Revenu_moyen'].fillna(dfB.groupby('Education')['Revenu_moyen'].transform('median'), inplace=True)


dfB[dfB.columns[4]].fillna(dfB.apply(lambda row: remplaceB(row["Education"], dfB.columns[4]), axis=1), inplace=True)

for j in range(8, len(dfB.columns)):
    dfB[dfB.columns[j]].fillna(dfB.apply(lambda row: remplaceB(row["Education"], dfB.columns[j]), axis=1), inplace=True)


# Calcul de la mediane des colonnes 'Income', 'Nombre_commandes_Tot' et 'Depences_Tot' par statut marital
Med_RevB = dfB.groupby('Marital_Status')['Income'].median()
Med_ComB = dfB.groupby('Marital_Status')['Nombre_commandes_Tot'].median()
Med_DepB = dfB.groupby('Marital_Status')['Depences_Tot'].median()

# Appliquer la fonction 'remplace_statB' Ã  chaque ligne du dataframe
dfB = dfB.apply(remplace_statB, axis=1)


df = pd.concat([dfA, dfB])
df.drop_duplicates(subset='ID', keep='first', inplace=True) #pour supprimer les ID qui sont egaux
df.reset_index(drop=True, inplace=True) # Supprime la colonne des index


df.to_csv("fichierAB.csv")
```