Data Engineer Certification - Practical Exam - Supplement Experiments

1001-Experiments makes personalized supplements tailored to individual health needs.

1001-Experiments aims to enhance personal health by using data from wearable devices and health apps.

This data, combined with user feedback and habits, is used to analyze and refine the effectiveness of the supplements provided to the user through multiple small experiments.

The data engineering team at 1001-Experiments plays a crucial role in ensuring the collected health and activity data from thousands of users is accurately organized and integrated with the data from supplement usage.

This integration helps 1001-Experiments provide more targeted health and wellness recommendations and improve supplement formulations.

Task

1001-Experiments currently has the following four datasets with four months of data:

- "user_health_data.csv" which logs daily health metrics, habits and data from wearable devices,
- "supplement_usage.csv" which records details on supplement intake per user,
- "experiments.csv" which contains metadata on experiments, and
- "user_profiles.csv" which contains demographic and contact information of the users.

Each dataset contains unique identifiers for users and/or their supplement regimen.

The developers and data scientsits currently manage code that cross-references all of these data sources separately, which is cumbersome and error-prone.

Your manager has asked you to write a Python function that cleans and merges these datasets into a single dataset.

The final dataset should provide a comprehensive view of each user's health metrics, supplement usage, and demographic information.

- To test your code, your manager will run only the code `merge_all_data('user_health_data.csv', 'supplement_usage.csv', 'experiments.csv', 'user`
- Your `merge_all_data` function must return a DataFrame, with columns as described below.
- All columns must accurately match the descriptions provided below, including names.

Data

The provided data is structured as follows:

The function you write should return data as described below.

There should be a unique row for each daily entry combining health metrics and supplement usage.

Where missing values are permitted, they should be in the default Python format unless stated otherwise.

Column Name	Description
user_id	Unique identifier for each user. There should not be any missing values.
date	The date the health data was recorded or the supplement was taken, in date format. There should not be any missing values.
email	Contact email of the user. There should not be any missing values.
user_age_group	The age group of the user, one of: 'Under 18', '18-25', '26-35', '36-45', '46-55', '56-65', 'Over 65' or 'Unknown' where the age is missing.
experiment_name	Name of the experiment associated with the supplement usage. Missing values for users that have user health data only is permitted.
supplement_name	The name of the supplement taken on that day. Multiple entries are permitted. Days without supplement intake should be encoded as 'No intake'.
dosage_grams	The dosage of the supplement taken in grams. Where the dosage is recorded in mg it should be converted by division by 1000. Missing values for days without supplement intake are permitted.
is_placebo	Indicator if the supplement was a placebo (true/false). Missing values for days without supplement intake are permitted.
average_heart_rate	Average heart rate as recorded by the wearable device. Missing values are permitted.
average_glucose	Average glucose levels as recorded on the wearable device. Missing values are permitted.
sleep_hours	Total sleep in hours for the night preceding the current day's log. Missing values are permitted.
activity_level	Activity level score between 0-100. Missing values are permitted.

```python
# Use as many python cells as you wish to write your code
import pandas as pd
import numpy as np

def extract(file):

    return pd.read_csv(file)

def get_age_group(age):
    if age == 'Unknown':
        return 'Unknown'
    if age < 18:
```

```
        return 'Under 18'
    elif 18 <= age <= 25:
        return '18-25'
    elif 26 <= age <= 35:
        return '26-35'
    elif 36 <= age <= 45:
        return '36-45'
    elif 46 <= age <= 55:
        return '46-55'
    elif 56 <= age <= 65:
        return '56-65'
    else:
        return 'Over 65'

def missed_values(df):

    # Checking missing values of all needed to data, also implement fillna when needed
    assert not df['user_id'].isna().any(), "There are missing values in 'user_id'"

    assert not df['date'].isna().any(), "There are missing values in 'date'"

    assert not df['email'].isna().any(), "There are missing values in 'email'"

    assert not df['supplement_name'].isna().any(), "There are missing values in
'supplement_name'"

    # While there is supplement takn, there cann't be any missing values in 'name',
'dosage', 'is_placebo'
    supplement_taken_df = df[df['supplement_name'] != 'No intake']

    assert not supplement_taken_df['name'].isna().any(), "There are missing values in
'name'"

    assert not supplement_taken_df['dosage'].isna().any(), "There are missing values in
'dosage'"

    assert not supplement_taken_df['is_placebo'].isna().any(), "There are missing values in
'is_placebo'"

    return df

def fill_missed_values(df):

    # Have to be a data in it
    df['user_id'] = df['user_id'].fillna('default_id')
    df['date'] = df['date'].fillna('2020-07-18')
    df['email'] = df['email'].fillna('default@example.com')

    # Got from describtion info how to fill
    df['age'] = df['age'].fillna('Unknown')
    df['supplement_name'] = df['supplement_name'].fillna("No intake")
```

```
    # Missing values are permited only when 'supplement_name' has 'No intake'
    supplement_taken_df = df[df['supplement_name'] != 'No intake']
    df.loc[supplement_taken_df.index, 'name'] = supplement_taken_df['name'].fillna('placebo
testing')
    df.loc[supplement_taken_df.index, 'dosage'] = supplement_taken_df['dosage'].fillna(1000)
    df.loc[supplement_taken_df.index, 'is_placebo'] =
supplement_taken_df['is_placebo'].fillna(True)

    # Missing values are permited
    df['average_heart_rate'] = df['average_heart_rate'].fillna('')
    df['average_glucose'] = df['average_glucose'].fillna('')
    df['sleep_hours'] = df['sleep_hours'].fillna('')
    df['activity_level'] = df['activity_level'].fillna('')

    return df

def transform(df):

    # Clean 'date'
    df['date'] = pd.to_datetime(df['date'], errors='coerce')

    # Clean 'user_age_group'
    df['user_age_group'] = df['age'].apply(get_age_group)

    # Clean 'experiment_name'
    df = df.rename(columns={'name': 'experiment_name'})

    # Clean 'dosage_grams'
    df.loc[df['dosage_unit'] == 'mg', 'dosage'] = df.loc[df['dosage_unit'] == 'mg',
'dosage'] / 1000
    df = df.rename(columns={'dosage': 'dosage_grams'})

    # Clean 'sleep_hours'
    df['sleep_hours'] = df['sleep_hours'].astype(str)
    df['sleep_hours'] = df['sleep_hours'].str.replace('h', '', case=False)
    df['sleep_hours'] = pd.to_numeric(df['sleep_hours'], errors='coerce')

    # Drop unnecessary columns
    df.drop(columns=['dosage_unit', 'experiment_id', 'description', 'age'], inplace=True)


    # Reorder columns
    desired_order = [
        'user_id', 'date', 'email', 'user_age_group', 'experiment_name',
        'supplement_name', 'dosage_grams', 'is_placebo',
        'average_heart_rate', 'average_glucose', 'sleep_hours', 'activity_level'
    ]
    df = df.reindex(columns=desired_order)

    # Clean 'activity_level'
    activity_level_map = {
        1: 0,
```

```
        2: 33,
        3: 66,
        4: 100
    }
    df['activity_level'] =
df['activity_level'].map(activity_level_map).fillna(df['activity_level'])
    df['activity_level'] = pd.to_numeric(df['activity_level'],
errors='coerce').fillna(0).astype(int)

    return df

def merge_all_data(file1, file2, file3, file4):

    # Load all files to df
    user_health_data = extract(file1)
    supplement_usage = extract(file2)
    experiments = extract(file3)
    user_profiles = extract(file4)

    # Merge tables
    merged_df = pd.merge(user_health_data, user_profiles, on='user_id', how='left')
    merged_df = pd.merge(merged_df, supplement_usage, on=['user_id', 'date'], how='outer')
    merged_df = pd.merge(merged_df, experiments, on='experiment_id', how='left')

    # Identify and replace missing values
    merged_df = fill_missed_values(merged_df)

    # Check for missed values
    merged_df = missed_values(merged_df)

    # Transform data
    merged_df = transform(merged_df)

    return merged_df

merged_df = merge_all_data("user_health_data.csv", "supplement_usage.csv",
"experiments.csv", "user_profiles.csv")
merged_df['user_id'] = merged_df['user_id'].astype('string')
merged_df['email'] = merged_df['email'].astype('string')
merged_df['user_age_group'] = merged_df['user_age_group'].astype('category')
merged_df['experiment_name'] = merged_df['experiment_name'].astype('category')
merged_df['supplement_name'] = merged_df['supplement_name'].astype('category')
merged_df['dosage_grams'] = merged_df['dosage_grams'].astype('float64')
merged_df['is_placebo'] = merged_df['is_placebo'].astype('bool')
merged_df['average_heart_rate'] = merged_df['average_heart_rate'].astype('float64')
merged_df['average_glucose'] = merged_df['average_glucose'].astype('float64')
merged_df['activity_level'] = merged_df['activity_level'].astype('int64')
merged_df['sleep_hours'] = merged_df['sleep_hours'].astype('float64')
print(merged_df.dtypes)
print(merged_df.head())
```

```
user_id                       string
date                  datetime64[ns]
email                         string
user_age_group              category
experiment_name             category
supplement_name             category
dosage_grams                 float64
is_placebo                      bool
average_heart_rate           float64
average_glucose              float64
sleep_hours                  float64
activity_level                 int64
dtype: object
                               user_id       date  ... sleep_hours activity_level
0  c6ae338a-9f95-481c-a88d-24a58bc8fc71 2018-01-31  ...         8.8              0
1  c6ae338a-9f95-481c-a88d-24a58bc8fc71 2018-02-28  ...         8.0             66
2  c6ae338a-9f95-481c-a88d-24a58bc8fc71 2018-03-31  ...        11.9              0
3  c6ae338a-9f95-481c-a88d-24a58bc8fc71 2018-03-31  ...        11.9              0
4  c6ae338a-9f95-481c-a88d-24a58bc8fc71 2018-04-30  ...         5.1              0

[5 rows x 12 columns]
```