EX02

Download and get to know the "Adult" census data base.

http://archive.ics.uci.edu/ml/datasets/Adult

There are 14 attributes in this data set.

1. For each attribute design and program a similarity metric. This is a function that takes in two values attribute values and returns a result in the interval [0,1]. "0" meaning ' not similar at all' , "1" meaning 'similar'.
2. Write a function that takes two entire records (two rows from the database) and returns a result in the interval [0,1]. "0" meaning ' not similar at all' , "1" meaning 'similar'.
3. You should hand in a pdf that includes 15 subsections - one for each attribute, and one for the overall function. Each section should contain a short (no more than one paragraph) explanation of how the function works why your strategy makes sense for that attribute and the code for the function itself (not the entire program).

Due Feb 4th end of day.

Collaboration encouraged. Groups can be no more than 2 people, but you are highly encouraged to talk among groups. Don't share electronic files, but it's ok to post equation and links to online sites that have looked at this problem. (A LOT of people have worked on this). In your hand in, give references for all the help you have gotten.   **CLASS DISCUSSION ENCOURAGED.**

Group： Zhenhao Zhang   Xuanting Xiong

1. Age

Age, which is an integer data. For a given age value `age1`,'age2', the normalized result `normal_age1``normal_age2` is calculated by subtracting `age1`, `age2` from the minimum age value `min_age` of the column and dividing by the difference between the maximum age `max_age` and the minimum age `min_age`. Use adult_data['age'].max and min to find the maximum and minimum values from the data. This calculation ensures that `normal_age1` and `normal_age2` represent the relative position of `age1` and 'age2` concerning the entire age range and are proportional between 0 and 1. After normalizing the two age values, the function calculates the similarity by computing the complement of the absolute difference between the two normalized values. That is, the similarity equals 1 minus the absolute value of the difference between `normal_age1` and `normal_age2`. This is because when two age values are very close, the absolute value of their difference will be minimal, so the similarity will be very close to 1. Conversely, if the two age values are very different, the absolute value of their difference will be close to 1, so the similarity will be close to 0. Thus, the `similarity_age` function can return a value between 0 and 1 depending on how close the two age values are.

```python
def similarity_age(age1, age2):
    # 获取 "age" 列的最大值和最小值
    max_age = adult_data['age'].max()
    min_age = adult_data['age'].min()

    normal_age1 = (age1 - min_age) / (max_age - min_age)
    normal_age2= (age2 - min_age) / (max_age - min_age)

    similarity = 1 - abs(normal_age1 - normal_age2)
    return similarity
```

2. Workclass

On the other hand, for the workclass category, which is categorical data, binary comparison is used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it’s impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```
def similarity_workclass(workclass1, workclass2):
    # 判断是否一致
    return 1 if workclass1 == workclass2 else 0
```

3. Fnlwgt

Fnlwgt, also called the Final weight, is Integer data. For a given Fnlwgt value ` Fnlwgt1’,’ Fnlwgt2’, the normalized result ` scaled_fnlwgt1’ ‘scaled_fnlwgt2’ is calculated by subtracting ‘Fnlwgt1’, ‘Fnlwgt2’ from the minimum Fnlwgt value `min_ Fnlwgt` of the column and dividing by the difference between the maximum Fnlwgt ‘max_fnlwgt’ and the minimum Fnlwgt `min_ Fnlwgt`. Use adult_data[‘Fnlwgt’].max and min to find the maximum and minimum values from the data. This calculation ensures that ` scaled_fnlwgt1` and ` scaled_fnlwgt2` represent the relative position of ` Fnlwgt1` and Fnlwgt2` concerning the entire Fnlwgt range and are proportional between 0 and 1. After normalizing the two Fnlwgt values, the function calculates the similarity by computing the complement of the absolute difference between the two normalized values. That is, the similarity equals 1 minus the absolute value of the difference between ` scaled_fnlwgt1` and ` scaled_fnlwgt1 `. This is because when two Fnlwgt values are very close, the absolute value of their difference will be minimal, so the similarity will be very close to 1. Conversely, if the two Fnlwgt values are very different, the absolute value of their difference will be close to 1, so the similarity will be close to 0. Thus, the `similarity_ Fnlwgt ` function can return a value between 0 and 1 depending on how close the two Fnlwgt values are.

```
def similarity_fnlwgt(fnlwgt1, fnlwgt2):

    max_fnlwgt = adult_data['fnlwgt'].max()
    min_fnlwgt = adult_data['fnlwgt'].min()

    scaled_fnlwgt1 = (fnlwgt1 - min_fnlwgt) / (max_fnlwgt - 
min_fnlwgt)
    scaled_fnlwgt2 = (fnlwgt2 - min_fnlwgt) / (max_fnlwgt - 
min_fnlwgt)

    similarity = 1 - abs(scaled_fnlwgt1 - scaled_fnlwgt2)
    return similarity
```

4. Education

On the other hand, for the education category, which is categorical data, binary comparison is used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it’s impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```python
def similarity_education (education1, education2):
    # Exact match
    return 1 if education1 == education2 else 0
```

5.   education-num

education-num, which is an Integer data. For a given Edu_num value ` Edu_num1','Edu_num2', the normalized result ` scaled_edu_num1' 'scaled_edu_num2' is calculated by subtracting 'Edu_num1', 'Edu_num2' from the minimum Edu_num value `min_ Edu_num` of the column and dividing by the difference between the maximum Edu_num 'max_edu_num' and the minimum Edu_num `min_ Edu_num`. Using adult_data['education-num'].max and min to find the max and min numbers from the data. This calculation ensures that ` scaled_ edu_num1` and ` scaled_edu_num2` represent the relative position of ` Edu_num1` and Edu_num2` concerning the entire education_num range and are proportional between 0 and 1. After normalizing the two educations_num values, the function calculates the similarity by computing the complement of the absolute difference between the two normalized values. That is, the similarity equals 1 minus the absolute value of the difference between ` scaled_edu_num1` and ` scaled_edu_num1 `. This is because when two educations_num values are very close, the absolute value of their difference will be minimal, so the similarity will be very close to 1. Conversely, if the two educations_num values are very different, the absolute value of their difference will be close to 1, so the similarity will be close to 0. Thus, the `similarity_ Edu_num ` function can return a value between 0 and 1 depending on how close the two educations_num are.

```python
def similarity_education_num(edu_num1, edu_num2):

    max_edu_num = adult_data['education_num'].max()
    min_edu_num = adult_data['education_num'].min()


    scaled_edu_num1 = (edu_num1 - min_edu_num) / (max_edu_num - 
min_edu_num)
    scaled_edu_num2 = (edu_num2 - min_edu_num) / (max_edu_num - 
min_edu_num)

    similarity = 1 - abs(scaled_edu_num1 - scaled_edu_num2)
    return similarity
```

6.   marital status

for the marital-status category, categorical data and binary comparison are used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it's impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```python
def similarity_marital_status(marital_status1, marital_status2):
    return 1 if marital_status1 == marital_status2 else 0
```

7. occupation

for the occupation category, categorical data, binary comparison is used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it’s impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```python
def similarity_occupation(occupation1, occupation2):
    return 1 if occupation1 == occupation2 else 0
```

8. relationship

For categorical data, binary comparison is used to calculate the similarity for the relationship category. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it’s impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```python
def similarity_relationship(relationship1, relationship2):
    return 1 if relationship1 == relationship2 else 0
```

9. race

For categorical data, binary comparison is used to calculate the similarity for the race category. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it’s impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```python
def similarity_race(race1, race2):
    return 1 if race1 == race2 else 0
```

10. sex

On the other hand, for the sex category, which is categorical data, binary comparison is used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it’s impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```python
def similarity_sex(sex1, sex2):
    return 1 if sex1 == sex2 else 0
```

11. capital-gain

Capital gain is Integer data. For a given Capital-gain value ` Capital-gain1',' Capital-gain2', the normalized result ` scaled_capital-gain1' 'scaled_capital-gain2' is calculated by subtracting 'Capital-gain1', 'Capital-gain2' from the minimum Capital-gain value `min_ Capital-gain` of the column and dividing by the difference between the maximum Capital-gain 'max_capital-gain' and the minimum Capital-gain `min_ Capital-gain`. Use adult_data['Capital-gain'].max and min to find the maximum and minimum values from the data. This calculation ensures that ` scaled_ capital-gain1` and ` scaled_capital-gain2` represent the relative position of ` Capital-gain1` and Capital-gain2` concerning the entire Capital-gain range and are proportional between 0 and 1. After normalizing the two Capital-gain values, the function calculates the similarity by computing the complement of the absolute difference between the two normalized values. That is, the similarity equals 1 minus the absolute value of the difference between ` scaled_capital-gain1` and ` scaled_capital-gain1 `. This is because when two Capital-gain values are very close, the absolute value of their difference will be minimal, so the similarity will be very close to 1. Conversely, if the two Capital-gain values are very different, the absolute value of their difference will be close to 1, so the similarity will be close to 0. Thus, the `similarity_ Capital-gain ` function can return a value between 0 and 1 depending on how close the two Capital-gain values are.

```
def similarity_capital_gain(capital_gain1, capital_gain2):


    max_capital_gain = adult_data['capital_gain'].max()
    min_capital_gain = adult_data['capital_gain'].min()

    scaled_cg1 = (capital_gain1 - min_capital_gain) / 
(max_capital_gain - min_capital_gain)
    scaled_cg2 = (capital_gain2 - min_capital_gain) / 
(max_capital_gain - min_capital_gain)

    similarity = 1 - abs(scaled_cg1 - scaled_cg2)
    return similarity
```

12. capital-loss

Capital loss is Integer data. For a given Capital-loss value ` Capital-loss1',' Capital-loss2', the normalized result ` scaled_capital-loss1' 'scaled_capital-loss2' is calculated by subtracting 'Capital-loss1', 'Capital-loss2' from the minimum Capital-loss value `min_ Capital-loss` of the column and dividing by the difference between the maximum Capital-loss 'max_capital-loss' and the minimum Capital-loss `min_ Capital-loss`. Use adult_data['Capital-loss'].max and min to find the maximum and minimum values from the data. This calculation ensures that ` scaled_ capital-loss1` and ` scaled_capital-loss2` represent the relative position of ` Capital-loss1` and Capital-loss2` concerning the entire Capital-loss range and are proportional between 0 and 1. After normalizing the two Capital-loss values, the function calculates the similarity by computing the complement of the absolute difference between the two normalized values. That is, the similarity equals 1 minus the absolute value of the difference between ` scaled_capital-loss1` and ` scaled_capital-loss1 `. This is because when two Capital-loss values are very close, the absolute value of their difference will be minimal, so the similarity will be very close to 1. Conversely, if the two Capital-loss values are very different, the absolute value of their difference will be close to 1, so the similarity will be close to 0. Thus, the `similarity_ Capital-loss ` function can return a value between 0 and 1 depending on how close the two Capital-loss values are.

```
def similarity_capital_loss(capital_loss1, capital_loss2):


    max_capital_loss = adult_data['capital_loss'].max()
    min_capital_loss = adult_data['capital_loss'].min()

    scaled_cl1 = (capital_loss1 - min_capital_loss) /
(max_capital_loss - min_capital_loss)
    scaled_cl2 = (capital_loss2 - min_capital_loss) /
(max_capital_loss - min_capital_loss)

    similarity = 1 - abs(scaled_cl1 - scaled_cl2)
    return similarity
```

13. hours-per-week

Hours-per-week is Integer data. For a given Hours-per-week value ` Hours-per-week1',' Hours-per-week2', the normalized result ` scaled_hours-per-week1' 'scaled_hours-per-week2' is calculated by subtracting 'Hours-per-week1', 'Hours-per-week2' from the minimum Hours-per-week value `min_ Hours-per-week` of the column and dividing by the difference between the maximum Hours-per-week 'max_hours-per-week' and the minimum Hours-per-week `min_ Hours-per-week`. Use adult_data['Hours-per-week'].max and min to find the maximum and minimum values from the data. This calculation ensures that ` scaled_ hours-per-week1` and ` scaled_hours-per-week2` represent the relative position of ` Hours-per-week1` and Hours-per-week2` concerning the entire Hours-per-week range and are proportional between 0 and 1. After normalizing the two Hours-per-week values, the function calculates the similarity by computing the complement of the absolute difference between the two normalized values. That is, the similarity equals 1 minus the absolute value of the difference between ` scaled_hours-per-week1` and ` scaled_hours-per-week1 `. This is because when two Hours-per-week values are very close, the absolute value of their difference will be minimal, so the similarity will be very close to 1. Conversely, if the two Hours-per-week values are very different, the absolute value of their difference will be close to 1, so the similarity will be close to 0. Thus, the `similarity_ Hours-per-week ` function can return a value between 0 and 1 depending on how close the two Hours-per-week values are.

```
def similarity_hours_per_week(hours_per_week1, hours_per_week2):


    max_hours_per_week = adult_data['hours_per_week'].max()
    min_hours_per_week = adult_data['hours_per_week'].min()

    scaled_hpw1 = (hours_per_week1 - min_hours_per_week) /
(max_hours_per_week - min_hours_per_week)
    scaled_hpw2 = (hours_per_week2 - min_hours_per_week) /
(max_hours_per_week - min_hours_per_week)
    return 1 - abs(scaled_hpw1 - scaled_hpw2)
```

14. native-country

On the other hand, for the native-country category, which is categorical data, binary comparison is used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it's impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```
def similarity_native_country(country1, country2):
    return 1 if country1 == country2 else 0
```

income

On the other hand, for the income category, which is categorical data, binary comparison is used to calculate the similarity. Since the values of categorical attributes are discrete and do not have a fixed numerical distance, it's impossible to estimate their differences as do for numerical attributes. Instead, compare the similarity of two values to see if they are identical. If the two values are similar, the similarity is 1; if they are different, it is 0.

```

def similarity_income (income1, income2):
    return 1 if income1 == income2 else 0
```

15. overall function

compares_records take two inputs and calls the similarity function for each attribute to compute similarity scores for the corresponding attribute. These scores reflect the similarity of the two records concerning specific attributes. The similarity scores for all attributes are averaged to get a composite score between 0 and 1. The closer the score is to 1, the more similar the two records are in all attributes. tested the first 100 data at the end and output the similarity of each data separately.

```
def compare_records(record1, record2):
    similarities = [
        similarity_age(record1['age'], record2['age']),
        similarity_workclass(record1['workclass'],
record2['workclass']),
        similarity_fnlwgt(record1['fnlwgt'], record2['fnlwgt']),
        similarity_education(record1['education'],
record2['education']),
        similarity_education_num(record1['education_num'],
record2['education_num']),
        similarity_marital_status(record1['marital_status'],
record2['marital_status']),
        similarity_occupation(record1['occupation'],
record2['occupation']),
        similarity_relationship(record1['relationship'],
record2['relationship']),
        similarity_race(record1['race'], record2['race']),
        similarity_sex(record1['sex'], record2['sex']),
        similarity_capital_gain(record1['capital_gain'],
record2['capital_gain']),
        similarity_capital_loss(record1['capital_loss'],
record2['capital_loss']),
        similarity_hours_per_week(record1['hours_per_week'],
record2['hours_per_week']),
        similarity_native_country(record1['native_country'],
record2['native_country'])
    ]
    overall_similarity = np.mean(similarities)  # 所有相似度的简单平均值
    return overall_similarity


# 获取数据集的记录数量
N = len(adult_data)

# 初始化一个空的相似度矩阵
similarity_matrix = np.zeros((N, N))

# 计算相似度
for i in range(N):
    for j in range(i + 1, N):  # 避免重复计算，j 从 i + 1 开始
        similarity = compare_records(adult_data.iloc[i],
adult_data.iloc[j])
        similarity_matrix[i][j] = similarity
        similarity_matrix[j][i] = similarity  # 相似度是对称的，可以节省一
些计算

# 将对角线元素设置为 1（记录与自身的相似度）
np.fill_diagonal(similarity_matrix, 1)

# 打印相似度矩阵的一部分以供查看
print(similarity_matrix)
```