Zhenhao Zhang zzh133

2.3

Suppose that the values for a given set of data are grouped into intervals. The intervals and corresponding frequencies are as follows:

age	*frequency*
1–5	200
6–15	450
16–20	300
21–50	1500
51–80	700
81–110	44

Compute an approximate median value for the data.

To find the median value for the data, we need to use a median formula, which is L + ((N/2 – F) / *f*) * *w*.

L is the lower limit of the median class, which is 21.

N is the midpoint of the total number of observations, which N equals 200+450+300+1500+700+44 = 3,194. N / 2 is equal to 1,597.

f is the frequency of the median class, equal to 1500.

F is the cumulative frequency of the class preceding the median class. F equals 200+450+300 = 950.

w is the class width of the median class, which is 50-21=29.

Thus, the median value is 21 + ((1597 – 950) / *1500*) * *29.* = 33.5086666667

2.6

Given two objects represented by the tuples (22, 1, 42, 10) and (20, 0, 36, 8):

(a) Compute the *Euclidean distance* between the two objects.

(b) Compute the *Manhattan distance* between the two objects.

(c) Compute the *Minkowski distance* between the two objects, using $q = 3$.

(d) Compute the *supremum distance* between the two objects.

We used Python to solve this question. First, from scipy.spatial import distance. We were then given inputs tuple_1 = (22, 1, 42, 10) and tuple_2 = (20, 0, 36, 8).

a: Euclidean_dist = distance.euclidean(tuple_1, tuple_2) which is 6.708203932499369

b: Manhattan_dist = distance.cityblock(tuple_1, tuple_2) which is 11

c: Minkowski_dist = distance.minkowski(tuple_1, tuple_2, 3) which is 6.153449493663682

d: supremum_dist = distance.chebyshev(tuple_1, tuple_2) which is 6

2.7

The median is one of the most important holistic measures in data analysis. Propose several methods for median approximation. Analyze their respective complexity under different parameter settings and decide to what extent the real value can be approximated. Moreover, suggest a heuristic strategy to balance between accuracy and complexity and then apply it to all methods you have given.

Simple Random Sampling is a fundamental method that approximates the median of the entire dataset by randomly selecting a subset of data. The complexity of this method depends on the size of the subset chosen.

When data points are associated with weights, the Weighted Median better represents central tendency. Its complexity resembles finding the regular median.

Bin Approximation divides the data into several intervals and approximates the median by identifying the interval where the median might lie. Subsequently, interpolation is conducted within that interval. The number of intervals determines the complexity of this method.

Threshold-based sampling can strike a balance between accuracy and complexity. One strategy is to start with a simple random sampling method, which is appropriate when the size of the data set does not exceed a specific threshold. However, once the size of the dataset exceeds this threshold, more complex sampling techniques, such as a "divide and conquer" strategy, are required. The sample size required for simple random sampling is determined by the tolerable margin of error and the distributional characteristics of the data. To reduce computational complexity, efficient data structures such as balanced trees can be used to handle the weights of the weighted median.

2.8

It is important to define or select similarity measures in data analysis. However, there is no commonly accepted subjective similarity measure. Results can vary depending on the similarity measures used. Nonetheless, seemingly different similarity measures may be equivalent after some transformation.

Suppose we have the following 2-D data set:

	A_1	A_2
x_1	1.5	1.7
x_2	2	1.9
x_3	1.6	1.8
x_4	1.2	1.5
x_5	1.5	1.0

(a) Consider the data as 2-D data points. Given a new data point, x = (1.4,1.6) as a query, rank the database points based on similarity with the query using Euclidean distance, Manhattan distance, supremum distance, and cosine similarity.

(b) Normalize the data set to make the norm of each data point equal to 1. Use Euclidean distance on the transformed data to rank the data points.

(a)

For this question, use Python again.

```
import numpy as np
from sklearn.preprocessing import normalize
from scipy.spatial import distance

data = np.array([
    [1.5, 1.7],
    [2, 1.9],
    [1.6, 1.8],
    [1.2, 1.5],
    [1.5, 1.0]
])

query = np.array([1.4, 1.6])

# Part (a): Calculate distances and similarities
euclidean_distances = [distance.euclidean(i, query) for i in data]
manhattan_distances = [distance.cityblock(i, query) for i in data]
```

```
supremum_distances = [distance.chebyshev(i, query) for i in data]
cosine_similarities = [1 - distance.cosine(i, query) for i in data]



print(euclidean_distances,manhattan_distances,supremum_distances,cosine_similarit
ies)
```

The result is:
[0.14142135623730948, 0.6708203932499369, 0.2828427124746191, 0.22360679774997896, 0.608276253029822] [0.19999999999999996, 0.8999999999999999, 0.40000000000000013, 0.30000000000000004, 0.7000000000000002] [0.10000000000000009, 0.6000000000000001, 0.20000000000000018, 0.19999999999999996, 0.6000000000000001] [0.999991391443956, 0.9957522612528876, 0.9999694838187877, 0.999028234937562, 0.9653633930282662]

A graph can represent the data.

	euclidean_distances	manhattan_distances	supremum_distances	cosine_similarities
X1	0.14142135623730948	0.19999999999999996	0.10000000000000009	0.999991391443956
X2	0.6708203932499369	0.8999999999999999	0.6000000000000001	0.9957522612528876
X3	0.2828427124746191	0.40000000000000013	0.20000000000000018	0.9999694838187877
X4	0.22360679774997896	0.30000000000000004	0.19999999999999996	0.999028234937562
X5	0.608276253029822	0.7000000000000002	0.6000000000000001	0.9653633930282662

(b)

```
# Part (b): Normalize the data set and then calculate Euclidean distance
from sklearn.preprocessing import normalize

normalized_data = normalize(data, norm='l2')
normalized_query = normalize(query.reshape(1, -1), norm='l2')

# Calculate Euclidean distances for the normalized data points
normalized_euclidean_distances = [distance.euclidean(i,
normalized_query.flatten()) for i in normalized_data]

# Output the normalized Euclidean distances
print(normalized_euclidean_distances)
```

The output is:

[0.004149350803200864, 0.09217091457843411, 0.00781232119311402, 0.044085486555962686, 0.2631980507972417]

Thus:

	Euclidean distance
X1	004149350803200864
X2	0.09217091457843411
X3	0.00781232119311402
X4	0.044085486555962686
X5	0.2631980507972417

The results of the final ranking of the transformed data points: x1,x3,x4,x2,x5

3.1

Data quality can be assessed regarding several issues, including accuracy, completeness, and consistency. For each of the above three issues, discuss how data quality assessment can depend on the intended use of the data, giving examples. Propose two other dimensions of data quality.

Accuracy:
The degree to which the data reflects the real-world scenario it is supposed to represent. For instance, in medical data, accuracy is paramount because a slight inaccuracy can lead to a wrong diagnosis or treatment plan. However, some inaccuracies may be tolerable for a phone application recommendation system.

Completeness: pertains to the extent to which data is present and is of sufficient breadth and depth for the task. For example, in scientific research, complete data would include all the relevant details needed to be recorded carefully. However, for a computer using time analysis, only a subset of the data may be required, and the data can still be considered complete if it contains that subset.

Consistency: ensuring that the data does not have discrepancies when comparing the dataset. For instance, customer data should be consistent across different organizational databases to provide clarity and communication. Data inconsistency can lead to a lack of credibility and operational inefficiency. Also, dates might be recorded differently across systems, such as “10/12/25” and “25/12/10,” which can cause significant confusion when aggregating or comparing data.

Timeliness: This aspect concerns whether the data is up-to-date and available when needed. For example, stock market data is highly time-sensitive, and even a few seconds' delay can render it useless for traders making real-time decisions.

Believability refers to the extent to which data is accepted or regarded as valid by users. If a database has a history of errors, even if these errors are corrected, the users might not trust the data. This can impact user confidence and the decisions made based on this data.

3.3

Exercise 2.2 gave the following data (in increasing order) for the attribute *age*: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

1. **(a) Use *smoothing by bin means* to smooth these data, using a bin depth of 3. Illustrate your steps. Comment on the effect of this technique for the given data.**
2. **(b) How might you determine *outliers* in the data?**
3. **(c) What other methods are there for *data smoothing*?**

(a)

```
ages = [13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35,
35, 35, 35, 36, 40, 45, 46, 52, 70]



def smooth_by_bin_means(data, bin_depth):

    sorted_data = sorted(data)

    bin_means = []
    for i in range(0, len(sorted_data), bin_depth):
        bin_sum = sum(sorted_data[i:i + bin_depth])
        bin_mean = bin_sum / bin_depth
        bin_means.append(bin_mean)

    smoothed_data = []
    for mean in bin_means:
        smoothed_data.extend([mean] * bin_depth)

    return bin_means, smoothed_data

bin_means, smoothed_data = smooth_by_bin_means(ages, 3)

print(bin_means, smoothed_data)
```

Thus, the result is:

Bin Means：[14.67, 18.33, 21.00, 24.00, 26.67, 33.67, 35.00, 40.33, 56.00]

Smoothed Data

Bin			
1	14.666666666666666	14.666666666666666	14.666666666666666
2	18.333333333333332	18.333333333333332	18.333333333333332
3	21	21.0	21.0
4	24	24.0	24

5	26.666666666666668	26.666666666666668	26.666666666666668
6	33.666666666666664	33.666666666666664	33.666666666666664
7	35.0	35.0	35.0
8	40.333333333333336	40.333333333333336	40.333333333333336
9	56.0	56.0	56.0

(b)

Average data follows a statistical distribution; points deviating from this distribution can be considered outliers. In the case of a normal distribution, the Z-score method is commonly used. In this method, data points deviating more than three standard deviations from the mean are flagged as outliers. Alternatively, visual inspection of data plots can reveal outliers. Scatter plots, histograms, and other graphical representations can help identify points that appear to deviate from other data patterns.

(c)

Binning, a method commonly employed for data smoothing, involves distributing sorted data into distinct bins or intervals. This technique offers several approaches for achieving smoothing effects. One method is bin mean smoothing, where each value within a bin is substituted with the mean value of that bin. Another technique is regression, which involves fitting data values to mathematical functions.

3.5

What are the value ranges of the following normalization methods?

(a) min-max normalization

(b) z-score normalization

(c) z-score normalization using the mean absolute deviation instead of the standard deviation

(d) normalization by decimal scaling

(a) Min-max normalization typically scales values to fit within a [0, 1] range, but it can be adjusted to any specific range.

(b) Z-score normalization transforms the dataset with a mean and a standard deviation. This method does not have a fixed range, as it depends on the data distribution.

(c) Z-score normalization using the mean absolute deviation is a variation that uses the mean absolute deviation instead of the standard deviation. The range is not fixed and will depend on the data distribution.

(d) Normalization by decimal scaling adjusts the values by moving the decimal point. The number of decimal places moved depends on the maximum absolute value of the attribute. This method scales the values into a range that relies on the data, but all values will be between -1 and 1, not including -1 and 1 themselves.

3.7

Using the data for age given in Exercise 3.3, answer the following:

(a) Use min-max normalization to transform the value 35 for age onto the range [0.0, 1.0].

(b) Use z-score normalization to transform the value 35 for age, where the standard deviation of age is 12.94 years.

(c) Use normalization by decimal scaling to transform the value 35 for age.

(d) Comment on which method you would prefer to use for the given data, giving reasons as to why.

(a)

```
# Given age data set
ages = [13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35,
35, 35, 35, 36, 40, 45, 46, 52, 70]

# Value to normalize
value_to_normalize = 35

# (a) Min-max normalization
min_age = min(ages)
max_age = max(ages)
min_max_normalized = (value_to_normalize - min_age) / (max_age - min_age)

print(min_max_normalized)
```

Thus, 35 is transformed to 0.38596491228070173.

(b)

```
# (b) Z-score normalization
mean_age = sum(ages) / len(ages)
std_dev_age = 12.94  # given standard deviation
z_score_normalized = (value_to_normalize - mean_age) / std_dev_age

print(z_score_normalized)
```

Thus, 35 is transformed to 0.38926097658709724.

(c)

```
# (c) Decimal scaling
j = 1
max_abs_age = max(ages, key=abs)
while max_abs_age / (10 ** j) >= 1:
```

```
    j += 1
decimal_scaled = value_to_normalize / (10 ** j)

print(decimal_scaled)
```

Thus, when j = 2, 35 is transformed to 0.35

(d)

I prefer using min-max normalization for the given data because this method effectively scales the data into the range [0, 1] without distorting the relative distances between values, making it easier to compare data points directly.

3.11

Using the data for age given in Exercise 3.3,

(a) Plot an equal-width histogram of width 10.

(b) Sketch examples of each of the following sampling techniques: SRSWOR, SRSWR, cluster sampling, and stratified sampling. Use samples of size 5 and the strata "youth," "middle-aged," and "senior."

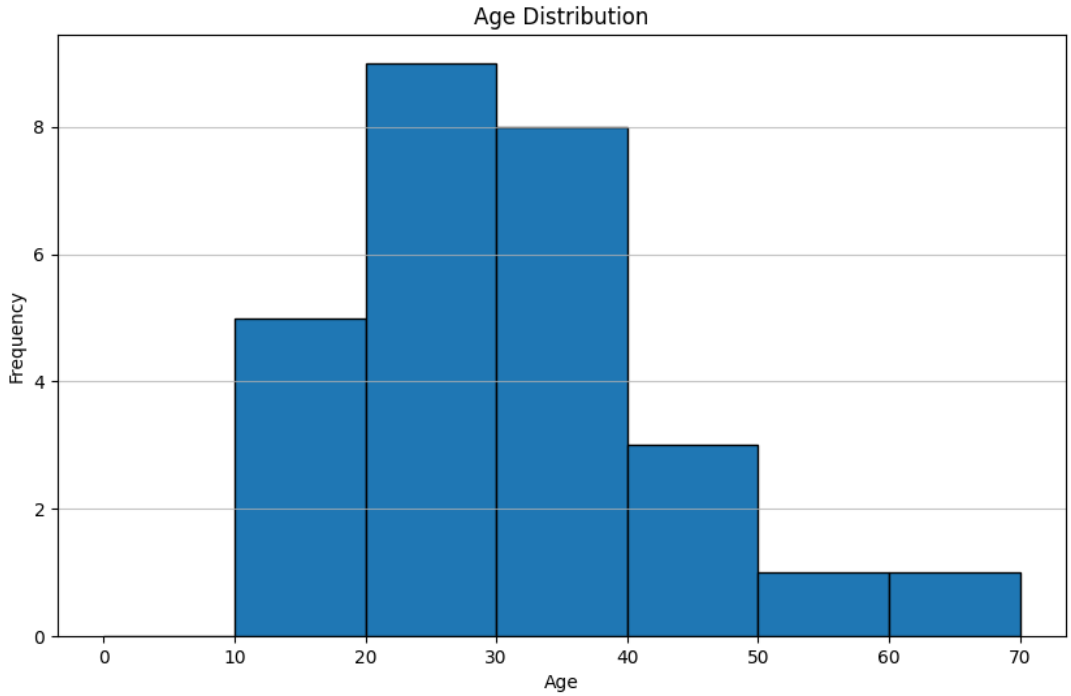
(a)

```
# Zhenhao Zhang 2/16/24
import matplotlib.pyplot as plt

ages = [13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35,
35, 35, 35, 36, 40, 45, 46, 52, 70]

plt.figure(figsize=(10, 6))
plt.hist(ages, bins=range(0, max(ages) + 10, 10), edgecolor='black')  # Adjusted
bins to start from 0
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)

# Showing the adjusted plot
plt.show()
```



(b)

Age Data: 13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30, 33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70.

SRSWOR: Individuals are randomly selected from the entire population, ensuring no repeats.

Eg: [13 15 25 25 33]

SRSWR: Individuals are randomly selected from the entire population, allowing repeats.

Eg: [13 13 15 25 25]

Cluster sampling: The population is divided into clusters based on age groups, and one cluster is selected to sample all members.

Eg:

Group 1: [13, 15, 16, 16, 19]

Group 2: [20, 20, 21, 22, 22]

Group 3: [25, 25, 25, 25, 30]

Group 4: [33, 33, 35, 35, 35]

Group 5: [35, 36, 40, 45, 46]

Group 6: [52, 70]

Choose Group 3: [25, 25, 25, 25, 30]

Stratified sampling.:

Eg:

Group 1: [13, 15, 16, 16, 19, 20, 20, 21, 22]

Group 2: [22, 25, 25, 25, 25, 30, 33, 33, 35]

Group 3: [35, 35, 35, 36, 40, 45, 46, 52, 70]

Choose：[13,25, 35, 20, 33]

3.13

Propose an algorithm, in pseudocode or in your favorite programming language, for the following:

(a) The automatic generation of a concept hierarchy for nominal data based on the number of distinct values of attributes in the given schema.

(b) The automatic generation of a concept hierarchy for numeric data based on the equal-width partitioning rule.

(c) The automatic generation of a concept hierarchy for numeric data based on the equal-frequency partitioning rule.

Note: [MUST be done in a programming language - pseudo code does NOT count]

(a)

```
# Zhenhao Zhang 2/16/24

import pandas as pd

data = {
    'Attribute_1': ['1', '2', '3', '4'],
    'Attribute_2': ['a', 'd', 'c', 'd'],
    'Attribute_3': ['A', 'B', 'C', 'D']
}
df = pd.DataFrame(data)

count_ary = []
concept_hierarchy = []

for i in df.columns:
    distinct_count = df[i].nunique()
    count_ary.append({'name': i, 'count': distinct_count})


count_ary.sort(key=lambda x: x['count'])

for item in count_ary:
    concept_hierarchy.append(item['name'])

print(count_ary,concept_hierarchy)
```

The sample output is: [{'name': 'Attribute_2', 'count': 3}, {'name': 'Attribute_1', 'count': 4}, {'name': 'Attribute_3', 'count': 4}] ['Attribute_2', 'Attribute_1', 'Attribute_3']

(b)

```
# Zhenhao Zhang 2/16/24


range_min = 0  # 最小数据值
range_max = 100  # 最大数据值
step = 25  # 分箱的宽度


concept_hierarchy = []

# 根据等宽分割规则初始化概念层次结构
for i in range(range_min, range_max, step):
    j = {
        'name': len(concept_hierarchy),
        'min': i,
        'max': min(i + step - 1, range_max),
        'sum': 0,
        'count': 0
    }
    concept_hierarchy.append(j)


data = [23, 45, 12, 37, 84, 10, 66, 91, 15, 29]


# 将每个值分配到相应的分箱中，并增加相应的 sum 和 count 值
for i in data:
    for j in concept_hierarchy:
        if j['min'] <= i <= j['max']:
            j['sum'] += i
            j['count'] += 1
            break


for i in concept_hierarchy:
    if i['count'] > 0:
        i['mean'] = i['sum'] / i['count']
    else:
        i['mean'] = None
```

```
for i in concept_hierarchy:
    print("Level",f"{i['name']}: Min={i['min']}, Max={i['max']}, "
          f"Mean={i['mean']}, Sum={i['sum']}, "
          f"Count={i['count']}")
```

The sample output is:
Level 0: Min=0, Max=24, Mean=12.5, Sum=50, Count=4
Level 1: Min=25, Max=49, Mean=37.0, Sum=111, Count=3
Level 2: Min=50, Max=74, Mean=66.0, Sum=66, Count=1
Level 3: Min=75, Max=99, Mean=87.5, Sum=175, Count=2

(c)

```
# Zhenhao Zhang 2/17/24


import pandas as pd
import numpy as np


data = pd.DataFrame({ 'numeric_attribute': np.random.randint(0, 100, 100)})


num_bins = 5  # 例如，我们想要将数据等频分割成 5 个分箱

data['bin'] = pd.qcut(data['numeric_attribute'], q=num_bins, labels=[f'Bin {i+1}'
for i in range(num_bins)])

concept_hierarchy = data.groupby('bin')['numeric_attribute'].agg(['min', 'max',
'mean', 'sum', 'count']).reset_index()

print(concept_hierarchy)
```

The result is:

	bin	min	max	mean	sum	count
0	Bin 1	0	17	7.761905	163	21
1	Bin 2	19	37	28.421053	540	19
2	Bin 3	38	59	48.150000	963	20
3	Bin 4	61	75	69.636364	1532	22
4	Bin 5	77	99	88.222222	1588	18