数据挖掘课程作业
作业1

计科210X 甘晴void 202108010XXX

第一题

假设所分析的数据包括属性 age,它在数据元组中的值（以递增序）为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。

- a. 该数据的均值是多少？中位数是什么？
- b. 该数据的众数是什么？讨论数据的模态（即二模、三模等）。
- c. 该数据的中列数是多少？
- d. 你能（粗略地）找出该数据的第一个四分位数（*Q*1）和第三个四分位数（*Q*3）吗？
- e. 给出该数据的五数概括。
- f. 绘制该数据的盒图。
- g. 分位数-分位数图与分位数图有何不同？

解：

a. 该数据的均值是多少？中位数是什么？

```
均值：809/27=29.96
中位数：25
```

b. 该数据的众数是什么？讨论数据的模态（即二模、三模等）。

```
众数是 25 和 35，它们都出现了 4 次。
这个数据集是二模的，有两个众数，即两个峰态，因此是双峰众数。
```

c. 该数据的中列数是多少？

```
中列数：极大值与极小值的平均
极大值：70   极小值：13
中列数：（70+13）/2=41.5
```

d. 你能（粗略地）找出该数据的第一个四分位数（Q1）和第三个四分位数（Q3）吗？

> 四分位数：将数据集分成四等分的值。
> Q1 表示数据的第 25% 位置处的值，而 Q3 表示数据的第 75% 位置处的值。
> 第一个四分位数为[27/4]=7处，Q1=20,；第三个四分位数为21处，Q3=35。

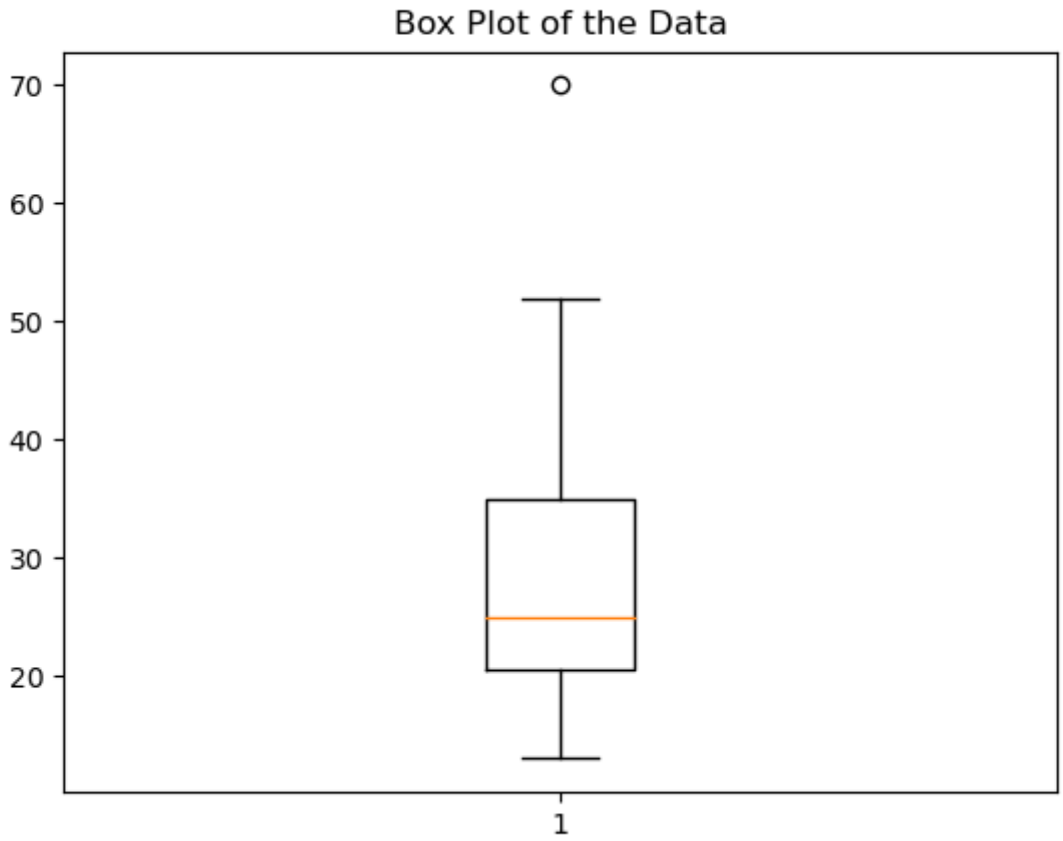
e. 给出该数据的五数概括。

> 五数概括包括最小值、第一四分位数（Q1）、中位数、第三四分位数（Q3）和最大值。
> 最小值：13
> Q1（第一个四分位数）：20
> 中位数：25
> Q3（第三个四分位数）：35
> 最大值：70

f. 绘制该数据的盒图。

```python
#python代码如下
import matplotlib.pyplot as plt
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]
plt.boxplot(data)
plt.title('Box Plot of the Data')
plt.show()
```

绘制图像如下：



g. 分位数-分位数图与分位数图有何不同？

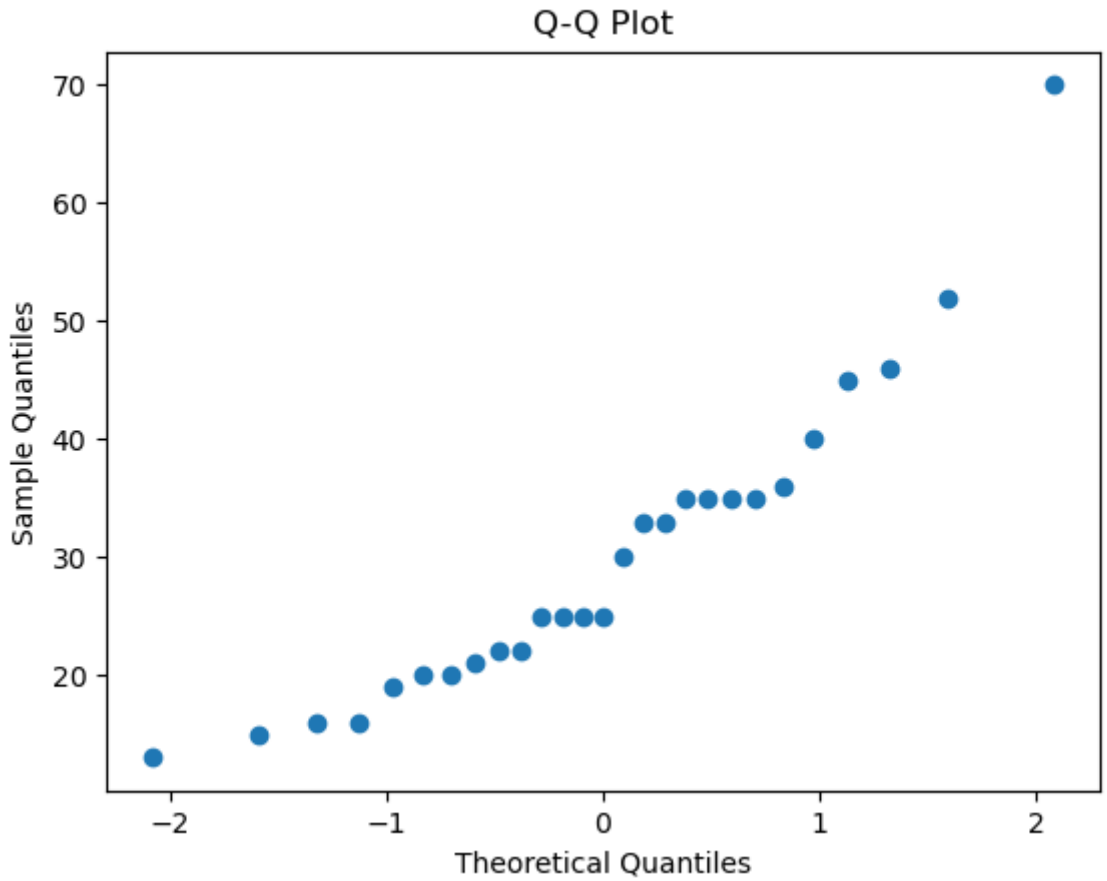
简单来说，
分位数图是一种观察单变量数据分布的简单有效分发（就是上面给出的箱型图）。首先它显示给定属性的所有数据的分布情况；其次它给出了分位数信息。
分位数-分位数图则是反映了同一个属性的不同样本的数据分布情况，使得用户可以很方便地比较这两个样本之间的区别或联系。

具体地说，
分位数图是一种用来展示数据值低于或等于在一个单变量分布中独立的变量的粗略百分比。这样,他可以展示所有数的分位数信息，而为独立变量测得的值(纵轴)相对于它们的分位数(横轴)被描绘出来。
分位数-分位数图用纵轴表示一种单变量分布的分位数，用横轴表示另一单变量分布的分位数。两个坐标轴显示它们的测量值相应分布的值域，且点按照两种分布分位数值展示。
举例来说，一条线(y=x)可画到图中+以增加图像的信息。落在该线以上的点表示在y轴上显示的值的分布比x轴的相应的等同分位数对应的值的分布高。反之，对落在该线以下的点来说，则低。

以下是简单实现观测值和正态分布对比的QQ图（仅仅作为练习使用）

```python
#python代码
import scipy.stats as stats
import matplotlib.pyplot as plt
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]
# 计算标准正态分布的分位数
theoretical_quantiles = stats.norm.ppf([(i - 0.5) / len(data) for i
in range(1, len(data) + 1)])
# 计算数据集的分位数
sample_quantiles = sorted(data)
plt.scatter(theoretical_quantiles, sample_quantiles)
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.title('Q-Q Plot')
plt.show()
```

绘制图像如下：



第二题

在数据分析中，重要的选择相似性度量。然而，不存在广泛接受的主观相似性度量，结果可能因所用的相似性度量而异。虽然如此，在进行某种变换后，看来似乎不同的相似性度量可能等价。

假设我们有如下二维数据集：

	A1	A2
X1	1.5	1.7
X2	2	1.9
X3	1.6	1.8
X4	1.2	1.5
X5	1.5	1.0

- a. 把该数据看做二维数据点。给定一个新的数据点x=(1.4,1.6) 作为查询点，使用欧几里得距离、曼哈顿距离、上确界距离和余弦相似性，基于查询点的相似性对数据库的点排位。
- b. 规格化该数据集，使得每个数据点的范数等于 1。在变换后的数据上使用欧几里得距离对诸数据点排位。

解：

a.计算四种距离并给出各自排名

首先需要了解这几个距离都是怎么计算的

- 欧几里得距离：d=sqrt[(x1-x2)^2+(y1-y2)^2]，即平面直角坐标系上两点间距离
- 曼哈顿距离：d=|x1-x2|+|y1-y2|
- 上确界距离：d=max(|x1-x2|,|y1-y2|)
- 余弦相似性：d=(A·B)/(||A||·||B||)，A,B分别为原点指向两个点的向量

使用以下python代码实现进行计算与排序

```
import numpy as np

# 数据集
data = np.array([[1.5, 1.7],
                 [2.0, 1.9],
                 [1.6, 1.8],
                 [1.2, 1.5],
```

```
                    [1.5, 1.0]])

# 查询点
query_point = np.array([1.4, 1.6])

# a. 使用不同相似性度量对数据点进行排名

# 欧几里得距离
euclidean_distances = np.sqrt(np.sum((data - query_point) ** 2,
axis=1))
euclidean_ranking = np.argsort(euclidean_distances)

# 曼哈顿距离
manhattan_distances = np.sum(np.abs(data - query_point), axis=1)
manhattan_ranking = np.argsort(manhattan_distances)

# 上确界距离
supremum_distances = np.max(np.abs(data - query_point), axis=1)
supremum_ranking = np.argsort(supremum_distances)

# 余弦相似性
cosine_similarities = np.dot(data, query_point) /
(np.linalg.norm(data, axis=1) * np.linalg.norm(query_point))
cosine_ranking = np.argsort(cosine_similarities)[::-1]  # 使用负值排
名，因为余弦相似性越大越相似

# 计算值
print("欧几里得距离计算值:", euclidean_distances)
print("曼哈顿距离计算值:", manhattan_distances)
print("上确界距离计算值:", supremum_distances)
print("余弦相似性计算值:", cosine_similarities)

# 打印排名结果
print("欧几里得距离排名:", euclidean_ranking + 1)  # 加1以匹配数据点的索引
print("曼哈顿距离排名:", manhattan_ranking + 1)
print("上确界距离排名:", supremum_ranking + 1)
print("余弦相似性排名:", cosine_ranking + 1)
```

结果如下：

```
E:\anaconda\envs\python3-11\python.exe E:\python_files\数据挖掘
\homework1\homework1-2.py
欧几里得距离计算值: [0.14142136 0.67082039 0.28284271 0.2236068
 0.60827625]
曼哈顿距离计算值: [0.2 0.9 0.4 0.3 0.7]
上确界距离计算值: [0.1 0.6 0.2 0.2 0.6]
余弦相似性计算值: [0.99999139 0.99575226 0.99996948 0.99902823
0.96536339]
欧几里得距离排名: [1 4 3 5 2]
曼哈顿距离排名: [1 4 3 5 2]
上确界距离排名: [1 4 3 2 5]
余弦相似性排名: [1 3 4 2 5]
```

整理与绘制表格

表一：相似性计算结果

	X1	X2	X3	X4	X5
欧几里得距离	0.14	0.67	0.28	0.22	0.61
曼哈顿距离	0.2	0.9	0.4	0.3	0.7
上确界距离	0.1	0.6	0.2	0.2	0.6
余弦相似性	0.99999139	0.99575226	0.99996948	0.99902823	0.99902823

表二：相似性排序结果

	排序结果
欧几里得距离	X1 < X4 < X3 < X5 < X2
曼哈顿距离	X1 < X4 < X3 < X5 < X2
上确界距离	X1 < X4 < X3 < X2 < X5
余弦相似性	X1 > X3 > X4 > X2 > X5

注意余弦相似性计算结果越大表示越相似。

b.规格化数据集并在变换后重新用欧几里得距离排序

操作如下：

1. 计算每个数据点的范数（欧几里得距离）
2. 将每个数据点除以其范数，以规格化数据点
3. 使用规格化后的数据集计算欧几里得距离并对数据点进行排名

可以使用python实现如上过程

```python
import numpy as np

# 数据集
data = np.array([[1.5, 1.7],
                 [2.0, 1.9],
                 [1.6, 1.8],
                 [1.2, 1.5],
                 [1.5, 1.0]])

# 计算每个数据点的范数
norms = np.linalg.norm(data, axis=1)

# 规格化数据集
normalized_data = data / norms[:, np.newaxis]

# 查询点，需要进行规格化
query_point = np.array([1.4, 1.6])

# 规格化查询点
query_point_norm = np.linalg.norm(query_point)
normalized_query_point = query_point / query_point_norm

# 使用欧几里得距离对规格化后的数据点进行排名
euclidean_distances = np.linalg.norm(normalized_data -
normalized_query_point, axis=1)
euclidean_ranking = np.argsort(euclidean_distances)

# 打印欧几里得距离计算值和排名
print("欧几里得距离计算值:", euclidean_distances)
print("欧几里得距离排名:", euclidean_ranking + 1)  # 加1以匹配数据点的索引
```

运行结果如下：

```
E:\anaconda\envs\python3-11\python.exe E:\python_files\数据挖掘
\homework1\homework1-2b.py
欧几里得距离计算值: [0.00414935 0.09217091 0.00781232 0.04408549
0.26319805]
欧几里得距离排名: [1 3 4 2 5]
```

表三：规格化后的欧几里得距离排序

	X1	X2	X3	X4	X5
规格化后的欧几里得距离	0.0041	0.0922	0.0078	0.0441	0.2632

排序结果：X1 < X3 < X4 < X2 < X5

第三题

使用如下方法规范化如下数组：

200，300，400，600，1000

- a. 令 min=0，max=1，最小—最大规范化。
- b. z 分数规范化。
- c. z 分数规范化，使用均值绝对偏差而不是标准差。
- d. 小数定标规范化。

解：

首先了解这四种数据规范化的方法的操作步骤

1. 最小-最大规范化：
    - 解释：将数据缩放到一个指定的范围，通常是[0, 1]。
    - 计算方法：对于每个数据点X，使用以下公式进行规范化： X=[X-min(X)]/[max(X)-min(X)]

- 意义：这种方法确保了所有数据都位于指定的范围内，其中最小值映射为0，最大值映射为1。

2. z 分数规范化：

- 解释：将数据映射为均值为0，标准差为1的正态分布（z 分数分布）。
- 计算方法：对于每个数据点X，使用以下公式进行规范化： X=(X-μ)/σ，其中μ为均值，σ为标准差
- 意义：这种方法适用于数据分布近似正态分布的情况，可以使数据更容易进行比较和分析。

3. z 分数规范化（使用均值绝对偏差而不是标准差）：

- 解释：将数据映射为均值为0，均值绝对偏差为1的分布。
- 计算方法：： X=(X-μ)/MAD，其中μ为均值，MAD为均值绝对偏差，MAD=(1/n)[Σ|xi-μ|]
- 意义：均值绝对偏差是数据点到均值的绝对距离的均值，与标准差不同。这种方法在数据中存在离群值（异常值）的情况下更稳健。

4. 小数定标规范化：

- 解释：通过移动小数点，将数据映射到[-1, 1]或其他合适的范围。
- 计算方法：找到数据中的最大绝对值，然后计算一个缩放因子，通常是10的幂，以便将最大绝对值缩放到1之下。然后，将所有数据点除以这个缩放因子。
- 意义：这种方法将数据点缩放到[-1, 1]或[-0.1, 0.1]等范围内，使数据易于理解和比较。这里我选取的是[-1,1]

不同的规范化方法适用于不同的数据和分析场景。您可以根据数据的性质和分析要求选择适当的规范化方法。

使用python代码实现

```
#给定数据
data = [200, 300, 400, 600, 1000]

# 最小-最大规范化
min_val = min(data)
max_val = max(data)
normalized_data = [(x - min_val) / (max_val - min_val) for x in
data]
#print(normalized_data)
print([round(val, 2) for val in normalized_data])  # 保留两位小数

#  z 分数规范化，使用标准差
import statistics
mean = statistics.mean(data)
```

```
std_dev = statistics.stdev(data)
z_scores = [(x - mean) / std_dev for x in data]
#print(z_scores)
print([round(val, 2) for val in z_scores])  # 保留两位小数


# z 分数规范化，使用均值绝对偏差
def mean_absolute_deviation(data):
    mean = sum(data) / len(data)
    deviation = [abs(x - mean) for x in data]
    return sum(deviation) / len(deviation)

mad = mean_absolute_deviation(data)
normalized_data = [(x - statistics.mean(data)) / mad for x in data]
#print(normalized_data)
print([round(val, 2) for val in normalized_data])

# 小数定标规范化
max_val = max(data)
num_digits = len(str(max_val))
scaled_data = [x / (max_val) for x in data]
#print(scaled_data)
print([round(val, 2) for val in scaled_data]) # 保留两位小数
```

结果如下：

```
E:\anaconda\envs\python3-11\python.exe E:\python_files\数据挖掘
\homework1\homework1-3.py
[0.0, 0.12, 0.25, 0.5, 1.0]
[-0.95, -0.63, -0.32, 0.32, 1.58]
[-1.25, -0.83, -0.42, 0.42, 2.08]
[0.2, 0.3, 0.4, 0.6, 1.0]
```

绘制表格如下

	X1	X2	X3	X4	X5
最小-最大规范化	0.0	0.12	0.25	0.5	1.0
z 分数规范化（标准差）	-0.95	-0.63	-0.32	-0.32	1.58

	X1	X2	X3	X4	X5
z 分数规范化（均值绝对偏差）	-1.25	-0.83	-0.42	0.42	2.08
小数定标规范化	0.2	0.3	0.4	0.6	1.0

第四题

假设 12 个销售价格记录已经排序，如下所示：

5，10，11，13，15，35，50，55，72，92，204，215

使用如下各方法将它们划分成三个箱。

- a. 等频（等深）划分。
- b. 等宽划分。
- c. 聚类。

解：

a. 等频（等深）划分：

等频划分将数据集分成相等数量的箱，每个箱中包含近似相等数量的数据点。

1. 首先，计算数据集的总数，即 12。
2. 然后，计算每个箱的大小，即 12 / 3 = 4。
3. 从最小值5开始，将数据点按顺序放入箱中，直到每个箱包含4个数据点为止。

划分后的三个箱分别是：

- 箱1: [5, 10, 11, 13]
- 箱2: [15, 35, 50, 55]
- 箱3: [72, 92, 204, 215]

b. 等宽划分：

等宽划分将数据集分成包含相等数值范围的箱。

1. 首先，找到数据集的最小值（5）和最大值（215）。
2. 计算数值范围，即 215 - 5 = 210。
3. 将数值范围除以3，以确定每个箱的宽度，即 210 / 3 = 70。
4. 从最小值开始，创建三个箱，每个箱的宽度为70。
5. 第一个箱，5-75；第二个箱，75-145；第三个箱，145-215

划分后的三个箱分别是：

- 箱1: [5, 10, 11, 13, 15, 35, 50, 55, 72]
- 箱2: [92]
- 箱3: [204, 215]

c. 聚类：

聚类方法：使用聚类算法来将数据点分成组。使用 k-means 聚类方法，将数据点划分成三个簇。这个过程需要计算簇的中心点，然后将每个数据点分配到离它最近的中心点所属的簇。

本题由于数据只有一个维度，故没有必要进行标准化，也可以进行标准化，但是实际测试没有改变结果（符合预期）。

在Python中，可以使用`sklearn`库来执行 k-means 聚类。使用以下代码实现。

```
from sklearn.cluster import KMeans
import numpy as np
from sklearn.preprocessing import StandardScaler

data = np.array([5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215])
data = data.reshape(-1, 1)  # 将数据转换为一列

# 标准化数据
# scaler = StandardScaler()
# data = scaler.fit_transform(data)

kmeans = KMeans(n_clusters=3, algorithm='lloyd').fit(data)
print(kmeans)
labels = kmeans.labels_
```

```
# 根据标签将数据点分为三个簇
cluster1 = data[labels == 0]
cluster2 = data[labels == 1]
cluster3 = data[labels == 2]

print("簇1:", cluster1)
print("簇2:", cluster2)
print("簇3:", cluster3)

# 打印每个簇的中心点位置
centers = kmeans.cluster_centers_
print("簇1 中心点:", centers[0])
print("簇2 中心点:", centers[1])
print("簇3 中心点:", centers[2])
```

这将使用 k-means 聚类将数据点划分成三个簇。在示例代码中，`cluster1`、`cluster2` 和 `cluster3` 包含了每个簇的数据点。

划分后的三个箱分别是：

- 箱1: [5, 10, 11, 13, 15, 35]
- 箱2: [50, 55, 72, 92]
- 箱3: [204, 215]

这三个簇的中心点分别为

- 14.833
- 67.25
- 209.5