第3章

数据预处理

教学视频

原始数据或多或少会存在一些问题，比如可能会有缺失值、异常值和重复值，不同来源的原始数据可能以不同的格式和结构存在，原始数据可能以不同的形式和单位表示，这些问题会影响到数据挖掘与分析的效率和准确性。数据预处理通过对原始数据的清洗、集成、变换和归约等方法，提升数据的品质。这一过程对于挖掘那些隐藏在复杂和不完整数据集背后的珍贵信息至关重要。直接使用未经预处理的原始数据，就像是在没有指南针的情况下航海，可能会导致偏离真实的数据挖掘与分析结果模型预测。

本章将为读者呈现数据预处理的各个维度，介绍相关的技术和方法，旨在优化数据的结构和质量，为后续的数据挖掘工作奠定坚实的基础。通过本章的学习，读者将掌握如何将原始数据转换为真正可以用于深入分析和建模的宝贵资源，从而揭开数据背后隐藏的秘密。

3.1 数据预处理概述

本节将深度探讨数据预处理的基本原理、数据预处理的概念，数据预处理对于数据挖掘的重要性，以及数据预处理的相关技术。

3.1.1 数据预处理的概念

随着信息技术的飞速发展，我们目睹了从手工记录到电子化数据库再到今天的大数据时代的巨大变革。计算机的普及使得数据处理变得更加高效和快速，同时也为数据的存储和传输提供了更便捷的途径。这种技术的进步为各个行业带来了前所未有的数据量，也为数据科学和机器学习提供了丰富的素材。正是在这个背景下，数据预处理崭露头角。

数据预处理，简而言之，是对原始数据进行清洗和转换的过程，以便数据可以更有效地用于数据分析和挖掘。这个过程包括多个关键步骤：处理缺失值、消除噪声、识别和修正异常值、集成多个数据源、变换数据格式以及减少数据规模等。这些步骤确保了数据

的质量和一致性，为后续的分析和挖掘工作打下坚实的基础。

3.1.2 数据预处理在数据挖掘中的作用

数据预处理位于数据挖掘流程的前端，是连接原始数据收集与后续分析和模型构建之间的桥梁。这一步骤的主要目标是将原始数据转换为更适合分析和挖掘的形式，确保数据的质量和一致性，从而为发现有价值的信息和知识打下坚实的基础。在没有进行充分预处理的情况下，即使是最先进的数据挖掘技术也难以发挥其应有的效能，因为“垃圾进，垃圾出”的原则在此过程中同样适用。

数据预处理的重要性不仅仅体现在提升数据质量上，它还直接关系到后续模型的准确性、效率和可靠性。通过有效的预处理，可以显著减少数据中的噪声和异常值，提高模型训练的速度，并增强模型对新数据的泛化能力。此外，适当的特征工程——一个重要的数据预处理步骤——能够显著提高模型解释能力和预测性能，使得最终挖掘出的模式更加准确和有用。

下面通过具体例子来阐明未经预处理的数据如何影响数据挖掘结果。

例 3-1 一个典型的例子是在进行客户细分时，使用了包含大量缺失值和异常值的数据集。在这种情况下，如果直接将这些原始数据用于聚类分析，可能会导致以下问题。

（1）缺失值的影响。

缺失值会导致分析算法无法正确识别数据间的相似性或差异性，因为部分信息的缺失使得数据的比较变得不准确。例如，如果客户的某些重要属性数据缺失，将这些客户归入任何一个细分群体都可能是不恰当的，这直接影响了细分结果的准确性和可靠性。

（2）异常值的干扰。

异常值可能会严重扭曲聚类分析的结果。比如，一个通常消费水平非常低的客户，因为某次特殊情况下的高额消费被错误地归入高消费客户群体。这种情况下，异常值没有得到妥善处理，就可能导致错误的客户群体划分，进而影响到后续的营销策略和决策制定。

未经预处理的数据直接用于数据挖掘，会因为数据质量问题而导致分析结果不准确或误导。因此，在进行任何形式的数据挖掘之前，合理的数据预处理步骤是必不可少的。这包括但不限于对缺失值进行填充、删除或估算，对异常值进行检测和处理，以及对数据进行规范化或标准化等操作。只有这样，才能确保数据挖掘过程中使用的数据是准确和可靠的，从而提高最终结果的质量和有效性。

因此，在数据挖掘流程中，数据预处理不仅是必不可少的一环，更是影响整个项目成功与否的关键因素。通过深入理解和正确应用数据预处理技术，可以有效提升数据挖掘项目的整体质量和效果。

3.1.3 数据预处理的主要任务

数据预处理包括数据清洗、集成、变换和归约四个环节，每个环节都不可或缺。

1. 数据清洗

数据清洗包括识别并处理数据中的错误和不一致性，比如缺失值、异常值和重复数

据的处理。对于缺失值的处理，常见方法包括插值、使用全局常数填充或依据其他数据推算值。异常值的检测和处理则是通过各种统计学方法来识别那些偏离正常数据范围的值，并据此决定是删除、修正还是保留这些值。

2. 数据集成

将不同来源、格式、特点的数据，在逻辑上或物理上有机地集中，合并为一个统一的数据集，以方便后续的处理，这就是数据集成。

在处理具体问题时用到的数据常常来自多个数据库或文件，这些数据可能存在结构不同、格式不一致、信息冗余等问题。通过有效的数据集成，可以有效解决这些问题。

3. 数据变换

数据变换包括规范化、离散化、属性构造和聚合等操作，规范化操作通过调整不同尺度上的数据，确保没有单一属性因其数值范围大而对分析结果产生不成比例的影响。离散化和属性构造将连续属性数据转换为类别属性数据，或者创建新的属性以更好地反映数据中的模式。这些变换增强了数据对于特定分析任务的适应性和解释能力。

4. 数据归约

数据归约旨在减少数据集的大小，同时尽量保留重要信息。这可以通过技术如维度归约、数值归约和数据压缩来实现。维度归约通过识别最重要的属性来减少属性的数量，而数值归约则试图用较小的数据集代替原始数据集，以减少计算资源的需求。这些方法有助于提高数据挖掘算法的效率和效果，特别是在处理大规模数据集时。

数据预处理对于整个数据挖掘过程有着不可或缺的贡献，周到细致的数据预处理，确保了数据挖掘过程的有效性和准确性，提升了分析结果的质量和实用性。

3.2　数据清洗


实验视频

数据清洗包括处理缺失值、噪声数据和异常值等多方面。每一种处理技术都有其适用场景和可能的影响，了解并正确应用这些技术对于提高数据质量、揭示深层次的数据洞察具有重要意义。通过细致地处理缺失值、平滑噪声、识别和处理异常值以及检查数据一致性，构建一个更加准确和可靠的数据分析基础。以此提升模型性能，确保分析结果的真实性和可信度。

3.2.1　缺失值处理

在数据预处理过程中，处理缺失值是一个不可避免的挑战。缺失值的存在可能会导致数据分析或模型建立的准确性受到影响。因此，采取合适的策略来处理这些缺失值至关重要。

1. 删除含有缺失值的记录

当数据集很大，且含有缺失值的记录占总数据量的很小一部分时，删除这些记录可能是一个简单且有效的解决方案。此方法适用于那些缺失值不会对数据分析结果产生重大影响的情况。频繁地删除含有缺失值的记录可能会导致数据量显著减少，这可能会影响模型的训练效果。此外，如果缺失值不是随机出现的，那么这种方法可能会引入偏

差，从而影响分析结果的准确性。

下面介绍该方法可能用到的库与函数。对于删除含有缺失值的记录，可以使用Pandas库和DataFrame.dropna()函数。Pandas是Python的一个强大的数据处理库，它提供了方便的数据结构和数据分析工具。DataFrame.dropna()函数用于删除含有缺失值的行或列。

用法如下：

```
import pandas as pd
# 假设 df 是已经创建好的包含缺失值的 DataFrame
# 删除含有任何缺失值的行
df.dropna(axis = 0, how = 'any', inplace = True)
```

常用参数介绍如下。

axis：默认为0，表示删除含有缺失值的行；如果设置为1，表示删除含有缺失值的列。

how：如果设置为any，则删除包含任何NaN的行或列；如果设置为all，则只删除所有值都为NaN的行或列。

thresh：设置行或列中非NaN值的最小数量，少于该数量的行或列将被删除。

subset：在哪些列中查找缺失值，仅当这些指定列中有缺失值时才删除列。

2. 均值法填充

在处理数据集中的缺失值时，常用的填充方法之一是均值填充，特别适用于连续型变量。这种方法通过计算变量的平均值，并用该值填充所有缺失的数据点，从而保持了数据的整体趋势和分布特性。均值填充的简单性和直观性使其成为初步数据处理中的常见选择，尤其是当数据集的缺失数据量不大且缺失随机发生时。

```
import pandas as pd
import numpy as np
# 使用均值填充
df['column_name'].fillna(df['column_name'].mean(), inplace = True)
```

3. 中位数法填充

在某些情况下，数据分布可能会因为极端值或不均匀分布而扭曲。这时，使用中位数来填充缺失值可能更为合适。中位数作为数据集中间点的值，对异常值具有更好的抵抗力，能够提供一个更稳健的数据填充选项。这种方法在处理具有偏斜分布的连续型变量时尤其有效，能够确保填充后的数据更真实地反映原始数据集的中心趋势。

```
import pandas as pd
import numpy as np
# 使用中位数填充
df['column_name'].fillna(df['column_name'].median(), inplace = True)
```

4. 众数法填充

对于分类变量，众数填充则是一种常见的处理策略。它涉及使用数据集中最频繁出现的类别来填充所有缺失的分类值。这种方法尤其适合那些类别明确且最常见类别占

比较高的情况，因为它假设缺失值很可能属于最常见的类别。众数填充简单易行，特别适合处理那些缺失数据量不大且缺失完全随机的情况，但它可能不适用于具有复杂数据结构或特定缺失模式的数据集。

```
import pandas as pd
import numpy as np
# 使用众数填充(注意众数可能返回多个值,这里取第一个)
df['column_name'].fillna(df['column_name'].mode()[0], inplace = True)
```

5. KNN 法填充

K 最近邻(KNN)方法为处理数据集中的缺失值提供了一种更精细化的策略。该方法的核心思想是利用数据的局部结构信息，通过寻找缺失值记录在特征空间中的 K 个最近邻居——即那些在已知特征上与其最相似的记录——来进行缺失值的估计或填充。这些邻居的相应值，无论是平均值还是众数，都被用作填充缺失值的依据，从而在一定程度上重建了丢失的信息。

KNN 方法的优点在于其能够充分考虑数据中的局部结构特性，通常能够提供比简单的均值或中位数填充更加精确的估计结果。然而，这种方法也存在一定的局限性。首先，它的计算量相对较大，尤其是在处理大规模数据集时，可能会导致显著的性能下降。此外，对于特征众多的数据集，选择一个合适的距离度量和 K 值可能会变得相当困难，这些因素都限制了 KNN 方法在实际应用中的灵活性和效率。

操作 KNN 方法时，首先需要确定一个合适的 K 值，比如 $K=5$，这一步骤对于后续的估计准确性至关重要。接着，对于数据集中每一条含有缺失值的记录，算法会计算其与其他记录在已知特征上的距离，以此来找出距离最近的 K 个记录。最后，根据这些最近邻居在缺失特征上的平均值(对于连续变量)或众数(对于分类变量)，来填充目标记录的缺失值。

下面介绍该方法可能用到的库与函数。对于删除含有缺失值的记录，可以使用 scikit-learn 的 KNNImputer 函数。

DataFrame.fillna()函数的用法如下：

```
from sklearn.impute import KNNImputer

# 创建 KNNImputer 实例
imputer = KNNImputer(n_neighbors = 5, weights = 'uniform')

# 对 DataFrame 进行拟合和转换
df = pd.DataFrame(imputer.fit_transform(df), columns = df.columns)
```

常用参数说明如下：

n_neighbors：K 的值，即选择多少个最近邻居，默认为 5。

weights：权重函数，可以是 uniform(所有点在平均中的权重相同)或 distance(点之间的权重与距离成反比)，或其他自定义函数。

metric：距离度量方式，默认为 minkowski，与 $p=2$ 时等同于欧氏距离。

6. 案例分析

例 3-2 以下案例使用 Python 的 Pandas 库和 scikit-learn 库来处理一个简化的数据集。假设有一个关于房地产市场的数据集，其中包含房屋的价格、面积、卧室数量和地理位置等信息。下面展示如何应用上述三种方法来处理这个数据集中的缺失值。

首先，需要安装并导入必要的库，并创建一个简化的示例数据集：

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer, KNNImputer

# 创建示例数据集
data = {
    'Price': [250000, np.nan, 320000, 590000, 225000],
    'Area': [1400, 1600, np.nan, 1800, 1500],
    'Bedrooms': [3, np.nan, 2, 4, np.nan],
    'Location': ['Suburb', 'City', 'City', 'Suburb', 'Rural']
}

df = pd.DataFrame(data)
```

通过命令查看数据，如图 3-1 所示，图中显示的 NaN 即为缺失值。

方法一：删除含有缺失值的记录。

```
# 删除含有缺失值的记录
df_dropped = df.dropna()
print("Data after dropping rows with missing values:\n", df_dropped)
```

这段代码会删除包含任何缺失值的记录(行)。这是最直接的处理缺失值的方法，但可能会导致大量数据的丢失。代码结果如图 3-2 所示，可以看到拥有缺失的数据行被删除了。

	Price	Area	Bedrooms	Location
0	250000.0	1400.0	3.0	Suburb
1	NaN	1600.0	NaN	City
2	320000.0	NaN	2.0	City
3	590000.0	1800.0	4.0	Suburb
4	225000.0	1500.0	NaN	Rural

图 3-1 数据集信息查看

```
Data after dropping rows with missing values:
      Price    Area  Bedrooms Location
0  250000.0  1400.0       3.0   Suburb
3  590000.0  1800.0       4.0   Suburb
```

图 3-2 删除缺失行后的数据集

方法二：缺失值填充。

对于不同类型的变量，可以使用不同的策略进行填充。

```
# 对连续变量使用均值填充，对分类变量使用众数填充
imputer_cont = SimpleImputer(strategy='mean')
df['Area'] = imputer_cont.fit_transform(df[['Area']])

imputer_cat = SimpleImputer(strategy='most_frequent')
df['Bedrooms'] = imputer_cat.fit_transform(df[['Bedrooms']])

print("Data after imputation:\n", df)
```

这段代码分别对 Area(连续变量)和 Bedrooms(分类变量)应用了均值填充和众数填充,结果如图 3-3 所示。

方法三：采用更复杂的缺失值估计方法。

使用 K 最近邻(KNN)方法来估计缺失值。

```
# 使用 KNN 估计方法填充缺失值
knn_imputer = KNNImputer(n_neighbors = 2)
# 注意:KNNImputer 通常只适用于连续变量,这里仅作为示例
numeric_columns = df.select_dtypes(include = [np.number]).columns
df[numeric_columns] = knn_imputer.fit_transform(df[numeric_columns])

print("Data after KNN imputation:\n", df)
```

这段代码使用 KNNImputers 从 scikit-learn 库来填充 Area 和 Price 字段的缺失值。KNN 方法考虑到了数据点之间的相似性,为缺失值提供了可能更加准确的估计,结果如图 3-4 所示。

Data after imputation:

	Price	Area	Bedrooms	Location
0	250000.0	1400.0	3.0	Suburb
1	NaN	1600.0	2.0	City
2	320000.0	1575.0	2.0	City
3	590000.0	1800.0	4.0	Suburb
4	225000.0	1500.0	2.0	Rural

图 3-3　使用均值和众数填充后的数据集

Data after KNN imputation:

	Price	Area	Bedrooms	Location
0	250000.0	1400.0	3.0	Suburb
1	407500.0	1600.0	3.5	City
2	320000.0	1450.0	2.0	City
3	590000.0	1800.0	4.0	Suburb
4	225000.0	1500.0	2.5	Rural

图 3-4　使用 KNN 方法填充后的数据集

7. 其他可能用到的函数

除了以上处理缺失值的方法之外,可能还会用到以下函数来查看数据集的信息。

1) info()方法

通过调用数据集的 info()方法,可以查看每个特征的非空值数量和数据类型信息。如果某个特征的非空值数量小于总样本数,那么该特征可能包含缺失值。

例 3-3　info()的使用案例。

对数据集使用 info()方法,查看每个特征的非空值数量和数据类型信息。下面是使用 info()方法查看数据集的示例代码。

```
import pandas as pd
data = pd.read_csv('happiness_train_abbr.csv')
# 查看每个特征的非空值数量和数据类型信息
data.info()
```

这段代码将导入名为'happiness_train_abbr.csv'的数据集,然后使用 info()方法来查看数据集的信息。调用 info()方法后,它会显示数据集中每个特征的名称、非空值的数量以及数据类型。可以通过这些信息来了解数据集的结构和特征的缺失情况。

如图 3-5 所示,可以看出该数据集一共有 8000 条数据,共 42 列。

同时,在展示出的数据信息中,如图 3-6 所示特征“work_yr”“work_status”等的非空数据条数不足 8000,说明这些特征中是有缺失值的。

```
RangeIndex: 8000 entries, 0 to 7999
Data columns (total 42 columns):
```

图 3-5　数据集信息 1

```
27  work_exper       8000 non-null   int64
28  work_status      2951 non-null   float64
29  work_yr          2951 non-null   float64
30  work_type        2951 non-null   float64
31  work_manage      2951 non-null   float64
32  family_income    7999 non-null   float64
33  family_m         8000 non-null   int64
34  family_status    8000 non-null   int64
```

图 3-6　数据集信息 2

2) isnull().sum()方法

使用该方法可以计算每个特征中缺失值的数量。它返回一个 Series 对象,索引是特征名,值是该特征中缺失值的数量。可以通过检查返回的结果来确定哪些特征存在缺失值。

例 3-4　isnull().sum()的使用案例。

这段代码将读取名为'happiness_train_abbr.csv'的数据集,并使用 isnull().sum()方法计算每个特征中缺失值的数量。下面是示例代码。

```
import pandas as pd
data = pd.read_csv('happiness_train_abbr.csv')
# 计算每个特征中缺失值的数量
data.isnull().sum()
```

	data
equity	0
class	0
work_exper	0
work_status	5049
work_yr	5049
work_type	5049
work_manage	5049
family_income	1
family_m	0
family_status	0
house	0

图 3-7　特征中缺失值的数量

如图 3-7 所示,可以看出该数据集的特征“work_status”“work_yr”“work_type”“work_manage”有 5049 条空数据,特征“family_income”有 1 条空数据。

3) isnull().any()方法

使用该方法可以检查每个特征是否存在缺失值。它返回一个布尔型的 Series 对象,索引是特征名,值是该特征是否存在缺失值(True 表示存在,False 表示不存在)。

例 3-5　isnull().any()使用案例。

使用 isnull().any()方法检查每个特征中是否存在缺失值。

```
import pandas as pd
data = pd.read_csv('happiness_train_abbr.csv')
# 查看每个特征的非空值数量和数据类型信息
data.info()
```

运行这段代码后,将获得一个 Series 对象,其中索引是数据集中的每个特征,值是一个布尔值,表示该特征是否存在缺失值。如果特征的值为 True,表示该特征存在缺失值;如果值为 False,则表示该特征中没有缺失值。如图 3-8 所示,特征“work_status”“work_yr”“work_type”“work_manage”“family_income”返回的对象值为 True,表示这些特征是存在缺失值的。

	data
equity	False
class	False
work_exper	False
work_status	True
work_yr	True
work_type	True
work_manage	True
family_income	True
family_m	False
family_status	False
house	False

图 3-8　特征中是否存在缺失值

3.2.2　噪声数据处理

所谓噪声数据，指的是那些由于各种外部或内部因素导致偏离真实值的数据点；噪声数据的来源多样，可能是由于测量误差、数据录入错误、传输过程中的损失或者是异常值的出现。这些数据点的存在，不仅扭曲了数据的真实面貌，也为数据分析的准确性设置了障碍。有效地识别并处理这些噪声数据，对于确保数据分析结果的可信度至关重要。通过采用科学合理的方法来减少或消除噪声数据的影响，更准确地捕捉到数据背后的模式和趋势，从而为后续的数据分析和决策提供坚实的基础。接下来将详细介绍几种常用且有效的噪声数据处理技术，并通过实例展示它们在实际应用中如何提高数据处理的质量和效率。

1. 平滑技术

平滑技术用于减少数据集中的随机波动和异常值的影响，从而揭示数据的真实趋势。滑动平均是其中一种常见的方法，它通过计算一定范围内数据点的平均值来平滑数据，常用的还有加权滑动平均和指数平滑等方法。

在金融市场分析中，滑动平均技术被广泛应用于股票价格趋势分析中。想象一下股票市场的日价格图。原始数据可能因日常波动而起伏不定，使得分析趋势变得困难。应用滑动平均技术后，会看到一条平滑的曲线贯穿这些波动点，这条曲线显示了价格随时间变化的平均趋势。分析师可以更清晰地看到价格的长期趋势，从而做出更准确的投资决策。

平滑技术特别适用于时间序列数据的分析，能有效去除短期波动带来的噪声，揭示长期趋势。它简单直观，易于实现和理解。然而，这种方法可能会掩盖一些重要的数据变化点，不适用于所有类型的数据分析。

例 3-6　平滑技术——滑动平均案例。

在许多实际情况中，数据可能会随时间显示出波动性，直接观察原始数据可能不易于发现其趋势。因此，通过计算滑动平均来平滑数据，可以更清晰地展示数据的长期趋势，减少短期波动的干扰。

```
# 导入对应的包
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt

# 生成示例数据
np.random.seed(0)
data = np.random.randn(100).cumsum()

# 将数据转换为 Pandas Series 对象,方便处理
data_series = pd.Series(data)

# 计算滑动平均,窗口大小为 5
smoothed_data = data_series.rolling(window=5).mean()

# 绘制原始数据和平滑后的数据
plt.figure(figsize=(10, 6))

# 绘制原始数据(蓝色实线)
plt.plot(data_series, label='原始数据', color='blue', linestyle='solid')

# 绘制平滑后的数据(红色虚线)
plt.plot(smoothed_data, label='平滑数据', color='red', linestyle='dashed')

plt.legend()

# 汉字字体
plt.rcParams['font.sans-serif'] = ['SimSun','KaiTI','Microsoft YaHei','LiSU','Arial Unicode MS']
# 正常显示负号
plt.rcParams['axes.unicode_minus'] = False
plt.show()
```

这段代码使用 Python 的数据处理和绘图库,生成并绘制了原始数据和其平滑后的数据的曲线图。首先,它导入了 pandas、numpy 和 matplotlib.pyplot 库,分别用于数据处理、数值计算和图形绘制。使用 numpy 库生成了一组随机数据,并将其转换为 pandas 的 Series 对象,方便后续处理。使用 rolling 函数对数据进行滑动平均计算,得到平滑后的数据。最后,代码使用 matplotlib.pyplot 库绘制了原始数据和平滑后的数据的曲线图,并设置了图例、坐标轴标签和标题。为了确保图形中能够正确显示中文,代码还设置了 matplotlib 的默认字体,并指定了几个常用的中文字体,例如宋体、楷体等。最后,使用 plt.show()函数将图形显示出来。结果输出如图 3-9 所示,虚线为经过平滑后的数据,实线为原数据。

2. 聚类分析

聚类分析通过将数据点根据相似性分组来识别数据中的自然结构或模式。在聚类分析中,可以通过图形化方法展示数据点如何根据相似性被分组。使用散点图可以可视化聚类结果,不同的聚类可以用不同的颜色或形状标记。这种方法特别适合于识别哪些数据点是异常值(噪声),因为它们通常不属于任何主要聚类,而是孤立地位于图表的边缘位置。

聚类分析在无监督学习中非常有用,特别是在不知道数据中有哪些潜在模式时。它可以自动发现这些模式并识别噪声。但是,聚类的结果高度依赖于所选择的相似性度量

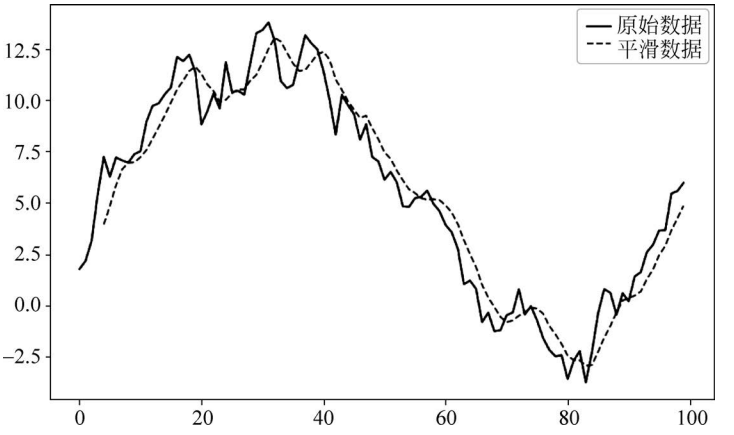


图 3-9　特征中是否存在缺失值

和聚类算法。更多有关聚类分析的介绍和案例请阅读第 6 章。

3. 概率与统计方法

利用概率统计的方法来检测异常值是一种有效的手段。可以通过计算数据集的均值和标准差来确定数据的正常波动范围。在此基础上，常见的做法是应用正态分布规则（如 3σ 原则），即认为位于均值三个标准差之外的数据点为异常值。此外，箱形图也是一种常用的工具，它通过四分位数和四分位距来识别异常值。对于更复杂的数据集，可以使用基于模型的方法，如高斯混合模型（GMM）或基于聚类的方法来鉴别异常点。这些方法通过建立数据的概率模型，并计算每个数据点属于该模型的概率，低概率的点被视为异常。

例 3-7　概率与统计方法——异常值检测。

概率与统计方法为处理和分析噪声数据提供了一种强有力的工具，帮助研究人员和分析师清洗数据、提高数据质量，并确保分析结果的可靠性和准确性，下面为具体案例。

```
import numpy as np
import matplotlib.pyplot as plt

# 生成示例数据
np.random.seed(0)
data = np.random.randn(100)

# 计算均值和标准差
mean = np.mean(data)
std_dev = np.std(data)

# 定义异常值判定标准(这里使用均值 ± 2 倍标准差)
outliers = []
normal_data = []

for i in data:
    if i < mean - 2 * std_dev or i > mean + 2 * std_dev:
        outliers.append(i)
```

```
        else:
            normal_data.append(i)

    print("Detected outliers:", outliers)

    # 绘制数据点
    plt.figure(figsize=(10, 6))
    plt.plot(normal_data, np.zeros_like(normal_data), 'o', label='Normal Data')
    plt.plot(outliers, np.zeros_like(outliers), 'ro', label='Outliers')

    # 标注均值和±2倍标准差线
    plt.axvline(x=mean, color='g', linestyle='--', label='Mean')
    plt.axvline(x=mean - 2 * std_dev, color='c', linestyle='--', label='Mean - 2 * Std Dev')
    plt.axvline(x=mean + 2 * std_dev, color='c', linestyle='--', label='Mean + 2 * Std Dev')

    plt.legend()
    plt.xlabel('Data Value')
    plt.yticks([])
    plt.title('Outlier Detection')
    plt.show()
```

这段代码是在数据挖掘的数据预处理阶段用于异常值检测的实例，使用了 Python 的 NumPy 和 matplotlib 库。首先，通过 NumPy 创建了一个包含 100 个从标准正态分布生成的随机数的数据集。为了确保结果的可复现性，设置了随机数种子为 0。

接下来，计算了数据集的均值和标准差，这两个统计量是后续确定数据点是否为异常值的基础。根据经验规则(均值±2 倍标准差)，对数据集中的每个数据点进行了遍历，以判断它们是否落在这个范围之外。落在范围之外的点被认为是异常值，并被收集到一个列表中，而其他的点则被视为正常数据并收集到另一个列表中。

在识别出异常值后，代码通过 matplotlib 绘制了一个图形来直观展示这些数据点，检测出的异常值列表如图 3-10 所示。

```
Detected outliers: [2.240893199201458, -2.5529898158340787, 2.2697546239876076, -1.980796468223927]
```

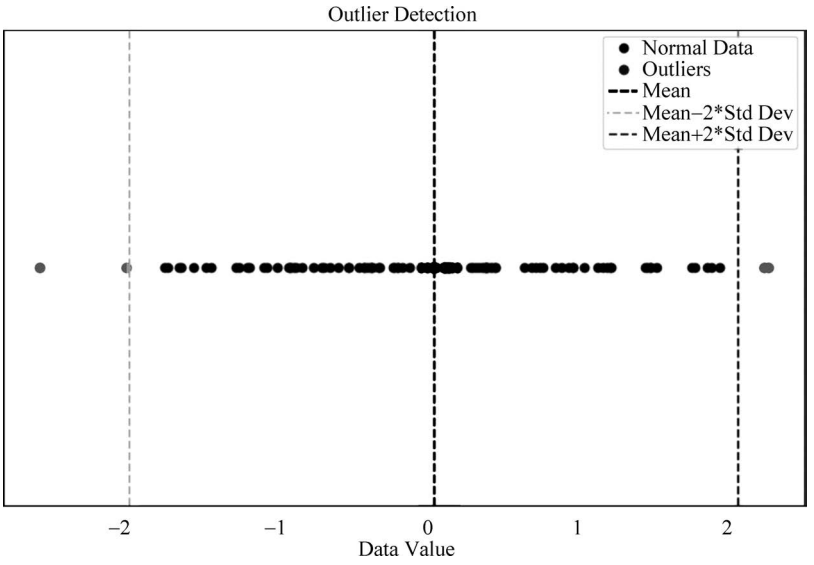
图 3-10　异常值列表

可视化结果如图 3-11 所示，蓝色点代表正常数据(Normal Data)，这些数据点在均值(Mean)的±2 倍标准差(Std Dev)范围内。红色点代表被检测为异常值(Outliers)，这些数据点超出了均值的±2 倍标准差范围。绿色虚线表示数据的均值。青色虚线表示均值加上或减去 2 倍标准差的位置，这是判定异常值的阈值。

在统计学中，如果数据是正态分布的，约有 95%的数据点会落在均值的±2 倍标准差范围内。超出这个范围的点可以被视为潜在的异常值，因为它们不太可能是随机变化的结果。

在这个具体的例子中，大部分数据点都集中在均值附近，形成了一条水平线。有几个点离群较远，它们被视为异常值。这种可视化有助于快速识别数据集中可能需要进一步调查的异常点。

上述三种方法不同情况下的适用性和优势不同。在实际应用中，选择最合适的方法



扫码看彩图

图 3-11　异常值检测可视化

需要根据具体的数据特性和分析目标来决定。理解并运用这些技术将大大提高数据预处理的效率和效果，为后续的数据挖掘工作打下坚实的基础。

3.2.3　异常值处理

异常值，也就是那些与大多数数据点显著不同的观测值，有时可能会扭曲整个数据分析的结果，导致误导性的结论。因此，识别并妥善处理这些异常值对于保证数据分析的准确性和可靠性至关重要。

数据清洗中的噪声数据处理和异常值处理虽然都旨在提高数据质量，但它们关注的问题和处理方法有所不同。噪声数据处理关注的是随机误差或无意义的数据变动，这些数据通常是由于测量误差、数据录入错误或其他随机因素造成的。噪声数据可能会掩盖数据的真实特征，影响数据分析的准确性。异常值处理则关注的是那些显著偏离其他观测值的数据点。这些数据点可能是由于测量错误、数据录入失误或者是真实的、非典型的观测结果造成的。异常值可能会对统计分析产生较大影响，如极端值可能会扭曲平均值和标准差。处理异常值的方法通常包括识别并评估这些值是否应该被保留、修改还是删除。

下面探讨两种广泛应用于异常值检测的方法：盒图分析与 Z 分数方法。

1. 盒图分析

盒图分析利用数据的五数概括——最小值、第一四分位数(Q1)、中位数、第三四分位数(Q3)以及最大值——来图形化地表示数据分布。这种方法的关键在于使用四分位距来评估数据点是否为异常值。四分位距的计算公式如式(3-1)所示。

$$\mathrm{IQR} = \mathrm{Q3} - \mathrm{Q1} \tag{3-1}$$

盒图不仅能够直观地展示数据的中心位置、分散程度和偏态，还能有效地揭示出异常值。例如，在财经领域，盒图经常被用于分析和比较不同公司或不同时间段内股票价

格的波动情况。盒图特别适用于小到中等规模的数据集。然而,对于具有复杂分布特性的大型数据集,其敏感度可能不足以准确识别所有异常值。

例 3-8 盒图分析。

下面是一个使用 Python 语言结合 Pandas 和 matplotlib 库来生成盒图的简单示例代码。这段代码展示了如何利用盒图分析技术来识别数据集中的异常值,为进一步的数据处理和分析提供依据。通过调整和应用这段代码,可以轻松地在自己的数据集上实现盒图分析,从而深入探究数据背后的故事。

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# 生成示例数据
np.random.seed(10)
data = np.random.normal(100, 20, size = 200)                    # 正态分布数据
outliers = np.random.uniform(low = 200, high = 300, size = 10)
data_with_outliers = np.concatenate((data, outliers))

df = pd.DataFrame(data_with_outliers, columns = ['Transaction volume box chart'])

# 使用 Seaborn 生成盒图
plt.figure(figsize = (10, 6))
sns.boxplot(x = df['Transaction volume box chart'])
plt.title('Transaction volume box chart')
plt.show()
```

结果如图 3-12 所示,显示的是一个盒形图,用于展示数据集中的异常值。这个盒形图反映了数据集的分布情况。盒形图的组成部分包括盒子(Box)、触须(Whiskers)和异常值(Outliers)。

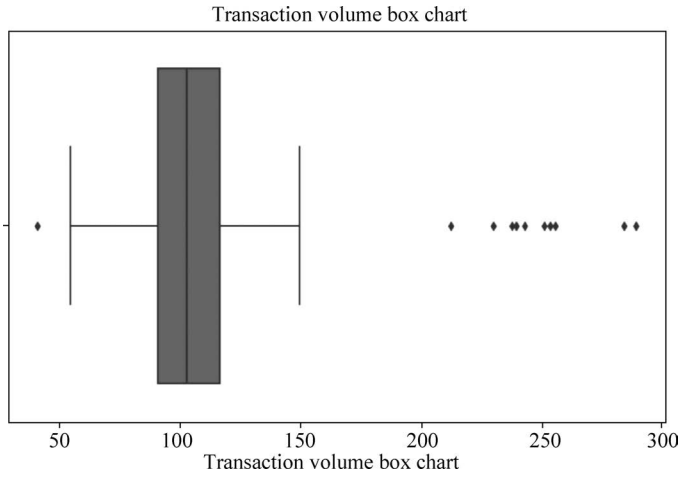


图 3-12 盒形图分析

盒子表示数据的四分位数范围，盒子的中间线代表中位数（第二四分位数，Q2），盒子的上边缘代表上四分位数（Q3），盒子的下边缘代表下四分位数（Q1）。触须是从盒子外伸展出来的线条，通常延伸至最小值和最大值，但不包括异常值。具体到这个例子中，触须可能延伸至 1.5 倍四分位距（IQR）之外。异常值为用小菱形表示的点，它们落在触须之外。在这个例子中，异常值明显高于正常数据的范围。

从图中可以看出，大多数数据点集中在 100 左右，反映了正态分布数据的特性。而超出触须的异常值则明显偏离了这个范围，表示它们与整体数据集有显著差异。

2. Z 分数方法

Z 分数方法将每个数据点与全体数据的平均值之间的差异通过标准差的倍数来度量。这意味着，如果一个数据点的 Z 分数绝对值非常高（例如大于 2 或 3），则该点可能被视为异常值。Z 分数方法的一个主要优点是它提供了一种量化数据点异常程度的方式，分析师可以根据 Z 分数的大小来决定是否需要进一步调查或处理某个数据点。在医学研究中，Z 分数方法常用于识别那些与总体健康指标显著不同的个体，从而进行早期干预。在防止信用卡欺诈方面，银行利用 Z 分数来监测异常交易活动。通过分析交易金额的 Z 分数，可以有效地识别出那些可能涉及欺诈行为的大额交易。

Z 分数方法适用于任何规模的数据集，并且对正态分布的数据特别有效。然而，这种方法依赖于数据集的平均值和标准差，对于本身就包含多个异常值的数据集而言，其稳定性和鲁棒性可能会受到影响。

例 3-9　Z 分数方法。

通过一个 Python 示例来展示如何使用 Z 分数方法来识别数据集中的异常值。这个例子将使用 Pandas 库来处理数据，并计算每个数据点的 Z 分数，从而识别和标记出异常值。这种方法的应用将帮助我们更好地理解数据的特性，并为进一步的数据分析打下坚实的基础。

```
import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
# 生成示例数据
np.random.seed(42) # 确保每次生成的随机数相同
normal_data = np.random.normal(200, 50, size = 990)
outliers = np.random.uniform(800, 1000, size = 10)
data = np.concatenate((normal_data, outliers)) # 合并数据
df = pd.DataFrame(data, columns = ['交易金额'])

# 计算 Z 分数
df['Z 分数'] = (df['交易金额'] - df['交易金额'].mean()) / df['交易金额'].std()

# 标识绝对值大于 3 的异常值
df['异常'] = df['Z 分数'].apply(lambda x: '是' if abs(x) > 3 else '否')

# 可视化
plt.figure(figsize = (10, 6))
plt.scatter(df.index, df['交易金额'], marker = 'o', color = 'blue', label = '正常值', s = 20)
# 使用圆圈
```

```
plt.scatter(df[df['异常'] == '是'].index, df[df['异常'] == '是']['交易金额'], s = 80,
marker = 's', color = 'red', label = '异常值') # 使用叉号
plt.axhline(y = df['交易金额'].mean(), color = 'g', linestyle = '--', label = '平均值')

plt.xlabel('索引')
plt.ylabel('交易金额')
plt.title('交易金额与 Z 分数异常值检测')
# 汉字字体
plt.rcParams['font.sans-serif'] = ['SimSun','KaiTI','Microsoft YaHei','LiSU','Arial Unicode MS']

# 正常显示负号
plt.rcParams['axes.unicode_minus'] = False
plt.legend()
plt.show()
```

这段代码首先生成了一个大型数据集。通过计算 Z 分数并设置阈值为 3，识别出这些异常值。最后，代码使用 matplotlib 库来可视化结果，如图 3-13 所示，其中正常的数据点在标准线上下，而被标记为异常的数据点则在离标准线较远的位置。这种可视化方法使得识别和理解数据中的异常值变得直观和容易。

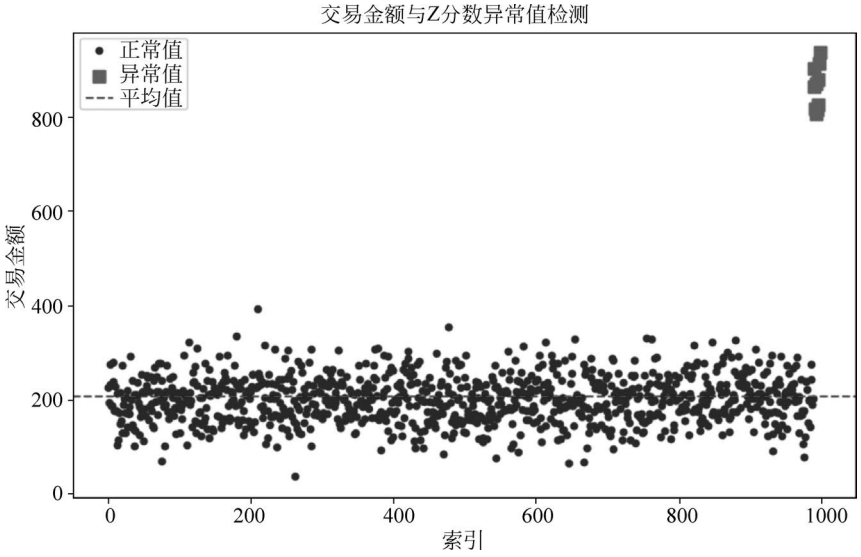


图 3-13　Z 分数方法分析

总之，噪声数据处理更多关注于减少数据中的随机误差，以提高数据集的整体质量，而异常值处理则专注于识别和处理那些偏离正常数据分布的个别数据点。两者都是数据预处理的重要组成部分，对提高数据分析项目的准确性和可靠性至关重要。


实验视频

3.3　数据集成

数据集成涉及将来自多个源的数据有效地合并和统一，以便进行全面分析。这一过程不仅要求识别和整合相关数据源，还需要解决数据格式不一致的问题，以确保数据集

的一致性和可用性。数据集成过程中，数据的冗余和相关性分析是必不可少的，它有助于提高数据质量，减少多重共线性问题，从而使得数据分析更加准确和高效。

3.3.1　数据源识别与整合

1. 概念介绍

在数据预处理的过程中，数据源识别与整合是至关重要的一步。它涉及从多个数据源中识别出相关数据，并将这些数据合并成一个统一的、可供分析的数据集。这一过程不仅要求技术上的精确操作，还需要对数据的业务逻辑有深刻的理解。数据源识别与整合的目标是确保数据的完整性和一致性，为后续的数据分析工作打下坚实的基础。

2. 应用场景

一个典型的应用场景是在零售行业中，一个公司可能需要从线上商城、实体店销售、第三方合作伙伴以及社交媒体等多个渠道收集销售数据。这些数据源格式各异，内容繁杂。通过识别这些不同来源中与销售业绩相关的数据，并将它们整合到一个统一的数据集中，公司可以更准确地分析销售趋势，优化库存管理，提高营销活动的针对性和效果。

3. 方法对比与优势

1）手动整合 vs 自动化工具

手动整合依赖于人工识别和整理数据，这种方法在处理小规模数据集时可能是可行的，但随着数据量的增加，它变得不切实际，容易出错。自动化工具，如 ETL（提取、转换、加载）工具，可以大大提高效率，减少错误。它们能够自动识别数据源，并按照预设规则进行整合。

2）规则基础方法 vs 机器学习方法

规则基础方法依赖于预定义的规则来识别和整合数据。这种方法在规则明确、数据结构稳定的场景下效果很好。机器学习方法可以自动发现数据之间的关联和模式，适用于复杂和不断变化的数据环境。尽管它们的实现更复杂，但能够提供更高的灵活性和准确率。

3）集中式 vs 分布式

集中式整合方法将所有数据汇聚到一个中心位置进行处理。这种方法简化了管理，但可能会遇到性能瓶颈。分布式方法允许在多个位置并行处理数据，提高了处理速度和扩展性。尤其在处理大规模数据集时，分布式架构显示出其优势。

4. 函数介绍

1）pd.read_csv()

pd.read_csv(filepath_or_buffer, sep=',', …)函数用于从指定的文件路径或类文件对象中读取 CSV（逗号分隔值）格式的数据。它返回一个 Pandas DataFrame 对象，为数据提供了一个二维标签化结构。

参数说明如下。

filepath_or_buffer：字符串，表示文件系统上的路径，或类文件对象。

sep：字符串，默认为','，用于指定字段的分隔符。

2）pd.concat()

pd.concat(objs，axis=0，join='outer'，ignore_index=False，…)函数用于沿着一定的轴将多个 pandas 对象(如 DataFrame 或 Series)堆叠到一起。可以用于简单地合并两个数据集。

参数说明如下。

objs：序列或映射，表示要连接的 pandas 对象。

axis：{0/'index'，1/'columns'}，默认为 0，如果是 0，将在索引上进行连接；如果是 1，则在列上进行连接。

join：{'inner'，'outer'}，默认为'outer'，表示连接的方式。'outer'表示取并集，'inner'表示取交集。

ignore_index：布尔值，默认为 False。如果为 True，则不使用索引标签；如果为 False，则保持原有的索引标签。

3）DataFrame.head()

DataFrame.head(n=5)方法返回 DataFrame 的前 n 行数据，默认值为 5。这是一种快速查看数据集前几行的简便方法。

4）merge 函数

merge 函数用于合并两个或多个数据集，根据一个或多个键将它们连接在一起。在 Python 中，merge 函数通常是通过 Pandas 库中的 merge 方法来实现的。

merge 函数的基本语法如下：

```
merged_data = pd.merge(left, right, how = 'inner', on = 'key')
```

其中，left 和 right 是要合并的两个数据集。

参数说明如下。

how：指定了合并方式，常用的选项包括'inner'(取交集)、'outer'(取并集)、'left'(以左边数据集为准)和'right'(以右边数据集为准)；

on：指定了用来连接的键。

5）DataFrame.to_csv()

DataFrame.to_csv(path_or_buf，index=True，…)方法将 DataFrame 对象写入 CSV 文件。通过这种方式，可以轻松地将处理和整合后的数据保存起来。

参数说明如下。

path_or_buf：字符串或文件句柄，指定写入文件的路径。

index：布尔值，默认为 True。如果为 True，则列出索引；如果为 False，则不列出索引。

5. 实战分析

1）使用 merge 方法进行数据集合并

例 3-10 数据集合并。

以下案例讨论的是最简单的数据表格的集成，并且简化数据表的格式一致，以 merge 方法为例。

```
import pandas as pd
# 读取数据集 data1 和数据集 data2,且 adgroup_id 为 raw_sample 和 ad_feature 共有列标签
raw_df = pd.read_csv('raw_sample.csv')
raw_df.info()
```

合并前的 raw_sample 数据信息如图 3-14 所示。

```
ad_fea_df = pd.read_csv('ad_feature.csv')
ad_fea_df.info()
```

合并前的 ad_feature 数据信息如图 3-15 所示。

```
Data columns (total 6 columns):
 #   Column      Dtype
---  ------      -----
 0   user        int64
 1   time_stamp  int64
 2   adgroup_id  int64
 3   pid         object
 4   nonclk      int64
 5   clk         int64
dtypes: int64(5), object(1)
```

图 3-14　合并前的 raw_sample 数据信息

```
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   adgroup_id   846811 non-null  int64
 1   cate_id      846811 non-null  int64
 2   campaign_id  846811 non-null  int64
 3   customer     846811 non-null  int64
 4   brand        600481 non-null  float64
 5   price        846811 non-null  float64
dtypes: float64(2), int64(4)
```

图 3-15　合并前的 ad_feature 数据信息

```
merged_df = pd.merge(raw_df, ad_fea_df, on = 'adgroup_id')# 使用 merge 方法合并 DataFrame
# 打印合并后的 DataFrame
merged_df.info()
```

合并后的数据集信息如图 3-16 所示。

2）使用 concat 方法对数据表进行行拼接或列拼接

例 3-11　以下案例讨论的是最简单的数据表格的集成，并且简化数据表的格式一致，以 concat 方法为例。

```
df1 = pd.read_csv(data1.csv')
df1.info()
```

合并前的 data1 数据信息如图 3-17 所示，有 300 条数据。

```
Data columns (total 11 columns):
 #   Column       Dtype
---  ------       -----
 0   user         int64
 1   time_stamp   int64
 2   adgroup_id   int64
 3   pid          object
 4   nonclk       int64
 5   clk          int64
 6   cate_id      int64
 7   campaign_id  int64
 8   customer     int64
 9   brand        float64
```

图 3-16　合并后的 merged_df 数据集信息

```
Index: 300 entries, 15972118 to 5858336
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user        300 non-null    int64
 1   time_stamp  300 non-null    int64
 2   adgroup_id  300 non-null    int64
 3   pid         300 non-null    object
 4   nonclk      300 non-null    int64
 5   clk         300 non-null    int64
dtypes: int64(5), object(1)
```

图 3-17　合并前的 data1 数据信息

```
df1 = pd.read_csv(data2.csv')
df1.info()
```

合并前的 data2 数据信息如图 3-18 所示，有 200 条数据。

```
df = pd.concat([df2,df1])
df.info()
```

经过行拼接后，df 数据集信息如图 3-19 所示，有 500 条数据。

```
Index: 200 entries, 15972118 to 2218222
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user        200 non-null    int64
 1   time_stamp  200 non-null    int64
 2   adgroup_id  200 non-null    int64
 3   pid         200 non-null    object
 4   nonclk      200 non-null    int64
 5   clk         200 non-null    int64
dtypes: int64(5), object(1)
```

图 3-18　合并前的 data2 数据信息

```
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user        500 non-null    int64
 1   time_stamp  500 non-null    int64
 2   adgroup_id  500 non-null    int64
 3   pid         500 non-null    object
 4   nonclk      500 non-null    int64
 5   clk         500 non-null    int64
dtypes: int64(5), object(1)
```

图 3-19　合并后的 df 数据集信息

3.3.2　数据格式统一化

1. 概念介绍

数据格式统一化指的是将来自不同来源的数据转换成一个统一的格式，以便于后续的数据处理和分析。这一步骤确保了数据分析的准确性和可靠性，同时也大大提高了数据处理的效率。数据格式的不一致性可能源于多种因素，包括不同的数据收集工具、存储格式或者单位度量标准等。统一化处理后的数据将有助于消除这些差异，实现数据的标准化和规范化。

2. 应用场景

在进行全球气候变化研究时，研究人员可能需要收集来自世界各地的气温数据。这些数据可能以不同的温度单位（摄氏度、华氏度）提供，也可能存储在不同格式的文件中（如 CSV、Excel、JSON）。在这种情况下，数据格式统一化就包括了将所有温度单位转换为摄氏度，并将所有数据转换为统一的 CSV 格式，以便于后续的数据分析和可视化。

3. 方法对比与优势

1）手动转换 vs 自动化脚本

手动转换涉及人工检查每个数据源并逐个进行格式调整。这种方法在处理小规模或少量数据源时可能是可行的，但随着数据量和数据源数量的增加，它变得非常耗时且容易出错。自动化脚本，如使用 Python 或 R 编写的脚本，可以预先定义转换规则并自动应用到所有数据源上。这种方法大大提高了效率和准确性，特别适合处理大规模数据集。

2）直接转换 vs 中间格式转换

直接转换指的是直接从原始格式转换为目标格式。这种方法简单直接，但在某些情

况下可能不够灵活。中间格式转换涉及先将所有数据转换为一个通用的中间格式(如JSON),然后再从中间格式转换为目标格式。这种方法增加了一个额外的步骤,但提供了更高的灵活性和可扩展性,尤其是当涉及多种目标格式时。

3) 规则引擎 vs 机器学习模型

规则引擎通过定义一套明确的规则来实现数据格式的统一化。这种方法在规则明确、数据结构相对固定的场景下非常有效。机器学习模型可以学习数据之间的映射关系,自动进行格式转换。这种方法适用于复杂的、非结构化的数据转换任务,但需要足够的训练数据和计算资源。

4. 函数与库介绍

1) dateutil.parser 库

dateutil.parser 是一个强大的日期和时间解析库,能够智能地解析各种格式的日期和时间字符串。它是 dateutil 模块的一部分,这个模块提供了广泛的日期和时间处理功能。

2) parser.parse 函数

parser.parse()函数是 dateutil.parser 模块中用于解析日期和时间字符串的函数。它可以自动识别许多不同的日期和时间格式,并将其转换为 Python 的 datetime 对象。在这个例子中,通过 apply()方法将 parser.parse()应用于 DataFrame 中的每个日期字符串,以解析并转换日期格式。

3) apply()方法

apply()方法在 pandas 中用于将函数应用于 DataFrame 或 Series 的行或列。在这个例子中,它被用来遍历 date 列中的每个元素,对每个元素执行 parser.parse()函数。

4) dt.strftime()方法

strftime()方法是 Python datetime 对象的一个方法,用于将 datetime 对象格式化为指定格式的字符串。在这个例子中,dt 访问器允许能够在 pandas Series 的每个 datetime 对象上调用 strftime()方法,并将日期格式化为"%Y-%m-%d"格式的字符串。

5. 实战介绍

例 3-12　日期格式统一化。

假设有一个包含不同日期格式的数据集,需要将所有日期统一为 YYYY-MM-DD 的格式。

```
import pandas as pd
from dateutil import parser
# 示例数据
data = {
    'date': ['01/02/2020', '2020-03-04', '15 April 2020']
}
df = pd.DataFrame(data)
# 使用 dateutil.parser.parse 来解析日期
df['date'] = df['date'].apply(lambda x: parser.parse(x))
# 如果需要,可以将日期转换为特定字符串格式
df['formatted_date'] = df['date'].dt.strftime('%Y-%m-%d')
print(df[['formatted_date']])
```

这段代码的目的是将包含不同格式日期字符串的 pandas DataFrame 中的日期统一格式化为标准日期字符串。它首先利用 dateutil.parser 库自动识别并解析各种日期格式，然后将解析后的日期转换为统一的格式（'%Y-%m-%d'），以便于后续的数据处理和分析。简而言之，这是一个将混合日期格式统一化的过程，非常适用于处理实际数据分析中遇到的日期格式不一致问题。转换结果如图 3-20 所示。

	formatted_date
0	2020-01-02
1	2020-03-04
2	2020-04-15

图 3-20 日期转换结果示意图

例 3-13 温度单位转换。

接下来处理一个常见的单位转换问题——将华氏度转换为摄氏度。

```
# 示例数据
data = {
    'temperature_F': [32, 68, 104]              # 华氏度
}
df = pd.DataFrame(data)
# 华氏度转摄氏度的转换公式
df['temperature_C'] = (df['temperature_F'] - 32) * 5.0/9.0
print(df)
```

这段代码的目的是将 DataFrame 中的温度从华氏度转换为摄氏度。首先，它创建了一个包含华氏温度（temperature_F）的 pandas DataFrame。然后，应用了华氏度到摄氏度的转换公式，将每个华氏温度值转换为摄氏度，并将这些转换后的摄氏度值存储在新的列 temperature_C 中。最后，打印出包含原始华氏度和转换后摄氏度的完整 DataFrame。这是一个数据转换的实例，转换结果如图 3-21 所示。

	temperature_F	temperature_C
0	32	0.0
1	68	20.0
2	104	40.0

图 3-21 温度单位转换结果示意图

3.3.3 数据冗余与相关性分析

数据冗余不仅浪费存储空间，还可能导致分析结果的偏差。当属性间具有高度相关性时，特别是多重共线性时，会严重影响模型的预测能力和稳定性。

1. 冗余属性识别

在数据集中，冗余属性指的是那些不提供任何新信息的属性，因为它们可以通过其他一个或多个属性推导出来。识别和处理这些冗余属性对于提高数据分析的效率和准确性至关重要。

假设有一个电商平台的用户数据集，其中包含用户的生日和年龄两个属性。这两个属性是冗余的，因为年龄可以通过当前日期和生日计算得出。在这种情况下，可以选择保留生日属性，并在需要时计算年龄，以减少数据的冗余性。

2. 相关性分析与处理

相关性分析是衡量两个或多个变量之间关系密切程度的过程。在数据预处理中，通过识别高度相关的属性，避免多重共线性问题，这对于一些模型（如线性回归模型）的性能至关重要。同时，通过分析属性（特征）与标记（目标变量）之间的相关性，可以识别出对预测结果最有影响的特征。这有助于选择最相关的特征构建模型，从而提高模型的准

确性和效率。

考虑一个房地产数据集，其中包含了房屋的面积和房间数两个属性。这两个属性可能高度相关，因为面积越大，房间数通常也越多。在建立预测房价的模型时，选择其中一个属性（如面积）作为特征可能更为合适，以避免多重共线性问题。

3. 库与函数介绍

1）SciPy 库

SciPy 是基于 NumPy 的一个开源软件，用于数学、科学和工程领域。SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理等科学和工程中常用的库。

2）scipy.stats.pearsonr(x, y)函数

这个函数用于计算两个数据集的皮尔逊相关系数。x 和 y 是两个长度相同的数组或序列。函数返回一个元组，其中包含相关系数和两个随机变量独立性的 p-value。皮尔逊相关系数度量的是两个变量之间的线性相关程度，其值介于－1～1，其中 1 表示完全正相关，－1 表示完全负相关，0 表示没有线性相关。

4. 实战分析

例 3-14　相关性分析示例。

本示例将探究年龄、收入和消费评分三个变量之间的相关性。通过构建一个简单的数据集，并使用 Python 中的 Pandas 和 SciPy 库来计算这些变量之间的皮尔逊相关系数，揭示出它们之间的相互作用。

```
import pandas as pd
import numpy as np
import scipy.stats
# 创建一个示例数据集
data = {
    '年龄': [23, 25, 28, 32, 35, 40, 45, 50, 55, 60],
    '收入': [25000, 28000, 29000, 31000, 33000, 35000, 37000, 40000, 42000, 45000],
    '消费评分': [65, 70, 68, 72, 74, 73, 76, 78, 80, 82]
}
df = pd.DataFrame(data)
# 计算年龄和收入的皮尔逊相关系数
pearson_coef, p_value = scipy.stats.pearsonr(df['年龄'], df['收入'])
print(f"年龄和收入的皮尔逊相关系数为:{pearson_coef:.2f}, p-value 为:{p_value:.3f}")
# 计算年龄和消费评分的皮尔逊相关系数
pearson_coef, p_value = scipy.stats.pearsonr(df['年龄'], df['消费评分'])
print(f"年龄和消费评分的皮尔逊相关系数为:{pearson_coef:.2f}, p-value 为:{p_value:.
3f}")
# 使用 Pandas 计算所有变量间的相关系数矩阵
correlation_matrix = df.corr()
print("相关系数矩阵:")
print(correlation_matrix)
```

在上述代码中，首先创建了一个包含年龄、收入和消费评分的数据集。然后，使用 scipy.stats.pearsonr 函数计算了年龄与收入、年龄与消费评分之间的皮尔逊相关系数。

最后，使用 Pandas 的 corr 方法计算了所有变量之间的相关系数矩阵。

皮尔逊相关系数的值范围为－1～1。接近 1 或－1 的值表示变量之间存在强正相关或强负相关，而接近 0 的值表示没有线性关系。p-value 用于测试相关性是否具有统计学意义，较小的 p-value(＜0.05)通常表示相关性具有统计学意义。结果如图 3-22 所示。

```
年龄和收入的皮尔逊相关系数为: 1.00, p-value为: 0.000
年龄和消费评分的皮尔逊相关系数为: 0.97, p-value为: 0.000
相关系数矩阵:
             年龄        收入      消费评分
年龄     1.000000  0.996035  0.967141
收入     0.996035  1.000000  0.980591
消费评分  0.967141  0.980591  1.000000
```

图 3-22　相关性分析结果示意图


实验视频

3.4　数据变换

本节探讨如何通过各种数据变换技术，将原始数据转换为更适合分析和挖掘的形式。从规范化的精确操作，到聚合和泛化的高层次数据抽象，再到针对特定需求的数据离散化处理和数据编码处理。

3.4.1　数据变换概述

数据变换作为预处理的重要环节，其核心目的在于将原始数据转换成更适合模型分析的格式。本节将简要介绍规范化、聚合、泛化、数据离散化和数据编码这 5 方面的知识。

1. 规范化

规范化通过应用特定的规则来将数据缩放至一定的数值范围，以此减少不同量纲和数值范围对分析结果的影响。规范化处理对于确保来自不同来源和尺度的数据能够公平比较、有效分析至关重要，以下是几种常见的规范化方法。

1) 最小-最大规范化

最小-最大规范化将数据转换到 0～1 的区间。对于每一个特征，该方法会找到对应特征的最大值和最小值，并通过这两个值来调整特征的数值，使其落在[0，1]的区间内。其公式如式(3-2)所示。

$$X_{\text{norm}}=\frac{X-X_{\min}}{X_{\max}-X_{\min}} \tag{3-2}$$

其中，$X_{\min}$ 和 $X_{\max}$ 分别是数据在该特征上的最小值和最大值。

这种方法非常适合于那些需要维持数据中原有精确比例关系的场合，例如在图像处理领域中调整像素强度。然而，最小-最大规范化可能会受到异常值的影响，因为异常值会导致其他所有正常值被缩放到一个很小的区间内。

2) Z 分数规范化

Z 分数规范化是基于原始数据的均值(μ)和标准差(σ)进行的标准化。其公式如式(3-3)所示。

$$Z=\frac{X-\mu}{\sigma} \tag{3-3}$$

经过 Z 分数规范化后的数据具有均值为 0 和标准差为 1 的特性。

这种方法适用于那些特征之间存在较大标准差差异或数据分布接近正态分布的情况。Z 分数规范化通过减去均值并除以标准差来转换数据，从而有效地处理了异常值和偏斜数据。由于这种方法依赖于均值和标准差，因此它对于具有异常值的数据集表现良好。

3）小数标定规范化

小数标定规范化是一种通过移动小数点的位置来调整数值大小的方法。这种方法不依赖于最小值和最大值，而是通过考虑数据中的最大绝对值来决定小数点移动的位数。小数标定规范化适合处理数值跨度大且需要保留数值间相对关系的数据集。它操作简单，易于理解和实现。但是，对于那些数值极大或极小的数据集，这种方法可能不太适用。

2. 聚合

数据聚合技术将众多的数据项合并为一个整体，聚合不仅简化了数据的复杂度，而且在提升数据抽象层次、减少数据量以及加快数据分析速度方面也起到了不可或缺的作用。

1）数据汇总技术

数据汇总技术是聚合中最为常见的一种形式。它通过对数据进行摘要或汇总，从而提升了数据的抽象级别。在实际应用中，这种技术使得我们能够快速把握数据的整体趋势和模式，而无须深陷于繁杂的细节之中。

商业智能报告是数据汇总技术应用的一个典型例子。通过这种技术，企业能够将每日或每周的销售数据汇总成月度或季度报告，从而更加直观地观察到产品销售的趋势和周期性波动。

2）层次聚合方法

除了简单的数据汇总之外，层次聚合方法则是一种更加细致和结构化的聚合手段。这种方法依据数据本身所固有的层次结构进行操作，因此在组织结构分析或地理信息系统（Geographic Information System，GIS）中尤为常见。

举例来说，在一个多层次的销售组织中，层次聚合方法可以帮助管理者按照地区、省份、国家等不同层级来汇总销售数据。这样不仅可以从宏观上理解市场分布情况，还能够针对特定区域进行深入分析，以发现潜在的市场机会或问题所在。

在地理信息系统中，层次聚合方法同样发挥着重要作用。例如，在环境监测或城市规划中，科学家和规划师可以通过将数据按照不同地理区域（如流域、城市、街区）进行聚合，来研究环境变化对不同区域的影响，或是评估城市发展政策的效果。

3. 泛化

数据泛化是一种将详细、敏感数据转换为抽象表示的技术手段。它通过替换、归纳或合并数据中的细节信息，减少个人信息的泄露风险，从而提高数据的隐私性和安全性。这种技术广泛应用于数据挖掘和隐私保护领域，尤其适用于需要对外共享但又不愿暴露

具体细节的场合。

1）抽象层次的提升

这种方法通过将具体的数据值替换为更加抽象的类别或概念来实现泛化。例如，在处理个人地址信息时，可以将详细的街道地址泛化为所在的城市或州名；同样，出生日期可以泛化为出生年份或所属年代，从而在不透露具体出生日期的情况下，依然可以进行年龄相关的统计分析。

2）数值属性的泛化

对于数值型数据，泛化通常是将具体的数值分组到更大的区间内。以年龄为例，原始数据中可能包含每个人的具体岁数，而通过泛化处理后，这些岁数可以被划分到如“20～29 岁”“30～39 岁”这样的年龄段中。这样不仅减少了数据的细节程度，而且便于进行群体特征的分析。

3）分类属性的泛化

对于分类数据，泛化是通过使用更广泛的类别来替代具体的类别值。例如，在职业分类中，“软件工程师”这一具体职业可以被泛化为更宽泛的“技术人员”类别。这种方法有助于在不损失过多信息的前提下，对数据进行有效的概括和简化。

4. 数据离散化

数据离散化的目的是将连续属性的值转换为一系列有限的区间，以此来简化数据模型的复杂性，增强数据的可理解性和可解释性。在数据挖掘和机器学习领域，离散化技术常被用于提高算法的运行效率和改善模型性能。

1）等宽离散化

等宽离散化是一种简单直观的离散化方法，它将属性的值域平均划分成若干个具有相同宽度的区间。每个区间包含的值范围是固定的，无论这些区间内实际包含多少数据点。这种方法适用于那些属性值分布较为均匀的情况。例如，如果我们有一组年龄数据，可以将年龄范围（0～100 岁）等分为若干 10 岁一个区间的段，如 0～9 岁、10～19 岁等。

等宽离散化方法的主要优点是实现简单，计算效率高。然而，其缺点也很明显：由于不考虑数据分布的特点，可能会因为异常值或者分布不均匀的存在而导致信息损失，使得某些区间内数据过于稀疏，而另一些区间则数据过于密集。

2）等频离散化

等频离散化方法则是将属性划分为含有相同数量值的区间。这意味着每个区间内包含的数据点数量是相同的，而区间的宽度则可能不同。这种方法适用于处理那些有偏分布（如长尾分布）的属性值。通过等频离散化，可以确保每个区间中数据点的数量均匀，从而更好地反映出数据的分布特性。

等频离散化方法能够较好地保留数据的分布信息，但也有其局限性。例如，它可能会将一些具有重要特征的数据点划分到不同的区间中，从而隐藏这些数据点之间的关联性或相似性。此外，在某些极端情况下，等频离散化可能会导致某些区间宽度过大，从而影响模型对数据细节的捕捉能力。

5. 数据编码

数据编码是数据预处理的核心任务，特别是当处理的数据集含有分类（类别）变量时。分类变量的编码对于大多数机器学习算法的性能有着重要影响。

1）标签编码（Label Encoding）

标签编码是一种常见的方法，它将分类数据转换为整数形式，使其适合在算法中使用。这种方法适用于类别之间存在某种有序关系时。

2）独热编码（One-Hot Encoding）

独热编码为每个类别创建一个新的二进制列，对于每个样本，其所属类别的列被标记为1，其余为0。这种方法适用于类别之间没有有序关系时。

3）频率编码（Frequency Encoding）

频率编码是一种用类别出现的频率来代替类别本身的编码方法。在频率编码中，每个类别都被替换为它在数据集中出现的频率，这样可以将分类变量转换为连续变量。频率编码保留了类别出现的频率信息，有助于模型更好地理解数据集中不同类别之间的重要性和关联性。

4）有序编码（Ordinal Encoding）

有序编码是一种将具有自然顺序的类别变量映射到有序整数的编码方法。当类别变量具有明显的顺序关系时，有序编码可以更好地表达这种顺序关系。有序编码能够保留类别之间的自然顺序关系，使得模型能够更好地理解和利用这种顺序信息。这对于评级、等级等有序类别的变量特别有用。

5）不同编码的优势

标签编码和有序编码都适用于有明显顺序的分类数据。不过，标签编码可能会误导一些模型，让它们错误地解读数字大小之间的关系。独热编码避免了模型误解类别之间关系的问题，但可能会导致数据维度急剧增加。二进制编码提供了一种折中方案，减少了独热编码可能导致的维度灾难，同时保持了一定程度的类别间的区分。频率编码通过反映类别的普遍性，为模型提供了额外的信息，但可能会丢失类别本身的一些信息。

在选择合适的编码方法时，需要考虑数据的特性、模型的类型以及最终的分析目标。每种编码方法都有其适用场景和限制，理解这些差异有助于更有效地处理分类数据。

通过对上述变换方法的深入理解和应用，可以有效地提升数据预处理的质量，为后续的数据挖掘任务打下坚实基础。每种方法都有其特定的应用场景和优缺点，因此，在实际应用中选择合适的数据变换策略对于确保数据挖掘项目的成功至关重要。

3.4.2　数据编码应用示例

本节将介绍数据编码中标签编码、独热编码、二进制编码、频率编码和有序编码的代码应用示例。

1. 标签编码

例 3-15　标签编码示例。

当处理包含服装尺寸（XS，S，M，L，XL）的数据集时，常常需要对这些尺寸进行编码以便在机器学习模型中使用。在这种情况下，将服装尺寸转换为整数可以帮助模型更

好地理解和处理这些数据。标签编码通常会按照顺序给每个类别分配一个整数，例如，XS可能被编码为0，S为1，以此类推。这种编码方式不会引入新的特征，而是简单地将分类数据转换为模型可以理解的形式，从而提高模型的性能和准确性。运行结果如图3-23所示。

```
from sklearn.preprocessing import LabelEncoder
# 示例数据
sizes = ['XS', 'S', 'M', 'L', 'XL']
# 初始化标签编码器
label_encoder = LabelEncoder()
# 拟合并转换数据
encoded_sizes = label_encoder.fit_transform(sizes)
print(encoded_sizes)
```

```
from sklearn.preprocessing import LabelEncoder

# 示例数据
sizes = ['XS', 'S', 'M', 'L', 'XL']

# 初始化标签编码器
label_encoder = LabelEncoder()

# 拟合并转换数据
encoded_sizes = label_encoder.fit_transform(sizes)

print(encoded_sizes)
```

```
[4 2 1 0 3]
```

图3-23　标签编码结果示意图

2. 独热编码

例3-16　独热编码示例。

当处理包含不同颜色(比如红、蓝、绿)的数据集时，通常会遇到需要将这些分类数据转换为模型可以理解的形式的情况。如果有一个包含颜色信息的数据集，其中包括红、蓝、绿三种颜色，使用独热编码可以将每种颜色表示为一个二进制向量。例如，红色可以表示为[1，0，0]，蓝色可以表示为[0，1，0]，绿色可以表示为[0，0，1]。这样，就可以将原始的分类数据转换为机器学习模型可以直接处理的数值形式，从而更好地进行数据分析和建模。

```
from sklearn.preprocessing import LabelEncoder
# 示例数据
sizes = ['XS', 'S', 'M', 'L', 'XL']
# 初始化标签编码器
label_encoder = LabelEncoder()
# 拟合并转换数据
encoded_sizes = label_encoder.fit_transform(sizes)
print(encoded_sizes)
```

如图3-24中的“False”即为0，“True”则为1。

3. 频率编码

例3-17　频率编码示例。

这段代码演示了如何使用频率编码来处理分类数据。在这个示例中，有一个包含手

```
import pandas as pd

# 示例数据
colors = ['Red', 'Blue', 'Green']

# 使用Pandas进行独热编码
encoded_colors = pd.get_dummies(colors)

print(encoded_colors)
```

```
    Blue  Green    Red
0  False  False   True
1   True  False  False
2  False   True  False
```

图 3-24　独热编码结果示意图

机品牌信息的数据集，其中包括了几个不同的品牌。通过计算每个品牌出现的频率，并将这些频率映射回原始数据集，实现频率编码的效果。

```
import pandas as pd
# 示例数据
data = pd.DataFrame({'Brand': ['Apple', 'Samsung', 'Apple', 'Xiaomi', 'Samsung']})
# 计算频率并映射到原始数据
frequency_encoding = data['Brand'].value_counts(normalize = True)
data['Brand_Freq'] = data['Brand'].map(frequency_encoding)
print(data)
```

首先，创建了一个 DataFrame，其中包含了手机品牌的信息。然后，使用 value_counts(normalize＝True)方法计算了每个品牌出现的频率，将其存储在 frequency_encoding 中。接下来，通过 map()函数将这些频率值映射回原始数据集，并将新的编码结果存储在名为 Brand_Freq 的新列中。

这样的处理方式可以将分类数据转换为连续变量，从而更好地在机器学习模型中使用这些数据。结果如图 3-25 所示，Brand_Freq 代表变量的频率。

```
     Brand  Brand_Freq
0    Apple         0.4
1  Samsung         0.4
2    Apple         0.4
3   Xiaomi         0.2
4  Samsung         0.4
```

图 3-25　频率编码结果示意图

4. 有序编码

例 3-18　有序编码示例。

对于具有自然排序顺序的类别变量(如评级“低”“中”“高”)，有序编码是一个合适的选择。

```
from sklearn.preprocessing import OrdinalEncoder
# 示例数据
ratings = [['Low'], ['Medium'], ['High'],['Low'],['High']]
# 初始化有序编码器
ordinal_encoder = OrdinalEncoder(categories = [['Low', 'Medium', 'High']])
# 拟合并转换数据
encoded_ratings = ordinal_encoder.fit_transform(ratings)
print(encoded_ratings)
```

这段代码通过 OrdinalEncoder 来实现有序编码。首先，指定类别的顺序，然后使用 fit_transform 方法来转换数据。这样，每个文本标签就被转换为一个有序的整数，反映

了其在指定顺序中的位置。结果如图 3-26 所示。

```
from sklearn.preprocessing import OrdinalEncoder

# 示例数据
ratings = [['Low'], ['Medium'], ['High'],['Low'],['High']]

# 初始化有序编码器
ordinal_encoder = OrdinalEncoder(categories=[['Low', 'Medium', 'High']]

# 拟合并转换数据
encoded_ratings = ordinal_encoder.fit_transform(ratings)

print(encoded_ratings)
```

```
[[0.]
 [1.]
 [2.]
 [0.]
 [2.]]
```

图 3-26　有序编码结果示意图

3.4.3　规范化应用示例

本节将介绍规范化中最小-最大规范化和 Z 分数规范化代码应用示例。

1. 最小-最大规范化

例 3-19　最小-最大规范化示例。

假设有一个关于房价的数据集，其中包含房屋面积和价格两个特征。面积的范围是 30～300 平方米，而价格的范围是 10 万～1000 万元。这两个特征的量级差异非常大，直接使用这些数据进行分析可能会导致模型对面积特征的影响被忽视。通过最小-最大规范化，将这两个特征都缩放到[0，1]范围内，从而避免量级差异带来的问题。

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
# 生成随机数据集(5 个数据点)
np.random.seed(0)
data = np.random.rand(5, 2)          # 5 个数据点,每个数据点包含两个特征
# 创建 MinMaxScaler 对象
scaler = MinMaxScaler()
# 使用 MinMaxScaler 进行最小 - 最大规范化
normalized_data = scaler.fit_transform(data)
print("Normalized Data:")
print(normalized_data)
```

```
Normalized Data:
[[0.23177196 0.65262109]
 [0.33167766 0.31759132]
 [0.         0.51630207]
 [0.02580038 1.        ]
 [1.         0.        ]]
```

图 3-27　最小-最大规范化结果示意图

这段代码演示了如何使用 Python 中的 MinMaxScaler 类来对数据集进行最小-最大规范化。MinMaxScaler()创建一个 MinMaxScaler 对象，用于进行最小-最大规范化操作。使用 fit_transform 方法对数据集进行最小-最大规范化，将数据缩放到[0，1]范围内，并将规范化后的数据保存在 normalized_data 中。结果如图 3-27 所示。

2. Z 分数规范化

例 3-20　Z 分数规范化示例。

在处理股票市场数据时，不同股票的价格和交易量可能差异巨大。如果想要比较不同股票价格的波动情况，直接使用原始数据是不合适的。通过应用 Z 分数规范化，将所有股票的价格和交易量转换为具有相同均值和标准差的数据，从而使比较更加公平和合理。

```
from sklearn.preprocessing import StandardScaler
import numpy as np
# 股票市场数据示例
data = np.array([[150, 1000000], [200, 1200000], [180, 800000], [220, 900000], [250,
1100000]])                                                    # 股票价格和交易量
# 创建 StandardScaler 对象
scaler = StandardScaler()
# 使用 StandardScaler 进行 Z 分数规范化
normalized_data = scaler.fit_transform(data)
print("Normalized Data:")
print(normalized_data)
```

结果如图 3-28 所示。

```
Normalized Data:
[[-1.46805055  0.        ]
 [ 0.          1.41421356]
 [-0.58722022 -1.41421356]
 [ 0.58722022 -0.70710678]
 [ 1.46805055  0.70710678]]
```

图 3-28　Z 分数规范化结果示意图

3.5　数据预处理应用案例


实验视频

淘宝购物是互联网电子商务蓬勃发展的时期，淘宝作为中国最大的电子商务平台之一，为全球用户提供了丰富的购物体验和广告展示机会。在这个时代，越来越多的消费者选择通过在线平台进行购物，而广告点击率成为衡量广告效果和用户兴趣的重要指标之一。本节主要对淘宝广告数据集进行数据预处理。

1. 数据集

本案例的数据由两个数据集组成，其中一个数据集中的每一行代表了一个广告的交互记录，包括用户 ID、时间戳、广告组 ID、PID、非单击(nonclk)以及单击(clk)的信息。另外一个数据集包含广告组 ID、类别 ID、广告活动 ID、客户、品牌和价格。

2. 具体代码

1）导入库

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import tree
from sklearn.metrics import roc_auc_score,log_loss
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

import pandas as pd：这行代码导入了 Pandas 库，并将其命名为 pd。Pandas 是一个用于数据操作和分析的强大库，通常用于处理结构化数据。

import seaborn as sns：这行代码导入了 Seaborn 库，并将其命名为 sns。Seaborn 是一个基于 matplotlib 的数据可视化库，它提供了更高级别的接口以创建各种有吸引力的统计图表。

import matplotlib.pyplot as plt：这行代码导入了 matplotlib 库中的 pyplot 模块，并将其命名为 plt。matplotlib 是一个用于绘制图表和可视化数据的库。

from sklearn import tree：这行代码从 scikit-learn 库中导入了决策树模型。

from sklearn.metrics import roc_auc_score,log_loss：这行代码从 scikit-learn 库中导入了用于评估分类模型性能的指标，包括 ROC 曲线下面积(ROC AUC)和对数损失(log loss)。

from sklearn.tree import DecisionTreeClassifier：这行代码从 scikit-learn 库中导入了决策树分类器模型。

from sklearn.model_selection import train_test_split：这行代码从 scikit-learn 库中导入了用于划分训练集和测试集的函数。

from sklearn.preprocessing import StandardScaler：这行代码从 scikit-learn 库中导入了数据标准化的功能，标准化是数据预处理中常用的一种方法，可以将数据转换为均值为 0，方差为 1 的标准正态分布。

这些导入的库和模块为接下来的数据挖掘和机器学习任务提供了基础设施和工具。

2) 读取数据集

```
raw_df = pd.read_csv('/Users/shiff/Desktop/广告点击率数据/raw_sample.csv')
ad_fea_df = pd.read_csv('/Users/shiff/Desktop/广告点击率数据/ad_feature.csv')
```

使用 pandas 的 read_csv()函数读取名为 raw_sample.csv 的数据集，并将其存储在 raw_df 中；读取名为 ad_feature.csv.csv 的数据集，并将其存储在 ad_fea_df 中。

3) 合并两个数据集

```
merged_df = pd.merge(raw_df, ad_fea_df, on = 'adgroup_id')
df = merged_df.sample(n = 3000, random_state = 666)
```

使用 Pandas 库中的 merge()函数将两个数据框(DataFrame)合并在一起。pd.merge()是 Pandas 库中用于合并数据框的函数。它会将两个数据框基于一个或多个键(key)进行连接。这里 merged_df 是一个新的数据框，它是通过合并 raw_df 和 ad_fea_df 得到的结果。在这个新的数据框中，raw_df 和 ad_fea_df 中的数据将根据 adgroup_id 列中的值进行匹配和合并。通过这样的合并操作，可以将两个数据框中的信息整合在一起，以便进行后续的分析、处理和建模。

4) 查看数据集合并后的一个信息

结果如图 3-29 所示。

```
df.head()
```

user	time_stamp	adgroup_id	pid	nonclk	clk	cate_id	campaign_id	cust
543582	1494254849	702628	430548_1007	1	0	6491	318435	
329145	1494233587	153446	430548_1007	1	0	6261	66086	
1076598	1494513320	683356	430539_1007	1	0	6736	381657	
551669	1494148986	480699	430539_1007	1	0	4282	373721	
672026	1494667219	465804	430548_1007	1	0	6426	341428	

图 3-29　合并后的数据集信息

5）舍弃不需要的特征

```
# 丢掉不用的特征
column_to_drop = ['user','time_stamp','nonclk']
df = df.drop(column_to_drop, axis = 1)

# 查看数据列有没有被丢弃
df.head()
```

这段代码的作用是从 DataFrame df 中删除名为 user、time_stamp 和 nonclk 的列。axis＝1 表示删除列，而 axis＝0 表示删除行。结果如图 3-30 所示。

adgroup_id	pid	clk	cate_id	campaign_id	customer	brand	price
702628	430548_1007	0	6491	318435	77998	63507.0	12.00
153446	430548_1007	0	6261	66086	104339	407835.0	714.00
683356	430539_1007	0	6736	381657	254073	429943.0	198.00
480699	430539_1007	0	4282	373721	37008	NaN	129.00
465804	430548_1007	0	6426	341428	227129	NaN	120.00

图 3-30　数据集信息查看

6）数据编码

```
# 数据标准化:
scaler = StandardScaler()
df['price'] = scaler.fit_transform(df['price'].values.reshape(-1,1))

# 缺失值处理和独热编码:
df['brand'] = df['brand'].fillna('-1')
df = pd.get_dummies(df,columns = ['pid','cate_id','campaign_id','customer','brand'],dtype =
int)
```

上述代码进行了一些数据预处理和特征工程的操作。

数据标准化处使用了 StandardScaler 来对 price 列进行标准化处理。标准化可以使数据符合标准正态分布，有助于一些机器学习算法的表现。

缺失值处理和独热编码部分，对 brand 列中的缺失值进行了填充，将缺失值替换为－1。然后使用 pd.get_dummies 进行独热编码，将 pid、cate_id、campaign_id、customer 和 brand 这几列进行独热编码处理，将其转换成哑变量，并设置数据类型为整数型。

习题 3

1. 从数据挖掘的角度，为什么数据预处理是一个至关重要的步骤？请解释其在整个数据挖掘流程中的作用。

2. 从一个实际数据集中选择一个包含缺失值的列，使用 Python 或其他适合的工具填补这些缺失值。可以选择使用均值、中位数、众数等方法进行填充。

3. 选择一个数据集，使用适当的异常值检测技术（如 Z 分数、箱线图等），识别和处理这些异常值。

4. 从一个实际数据集中选择一个分类变量，然后展示如何进行数据编码，例如使用独热编码或标签编码将分类变量转换为模型可以处理的形式。

5. 选择一个数值型特征，并展示如何进行归一化处理，比如使用最小-最大规范化或标准化方法将数据缩放到指定的范围。

6. 选择一个数据预处理的实际案例，描述该案例中涉及的数据预处理步骤，比如如何处理缺失值、异常值以及如何进行数据变换和数据集成。