GOROSPE, ROJ GIAN

Section 2.1

2.1.1. Nominal attributes

Nominal Attributes:

Nominal attributes are categorical variables that represent distinct categories or labels with no inherent order or ranking among them. These attributes classify data into various groups based on qualitative differences. Examples include colors, gender, types of animals, and more.

Practical Python Code for Handling Nominal Attributes:

Let's create a simple Python code snippet using the pandas library to work with a dataset containing nominal attributes. We'll load a sample dataset, explore nominal attributes, and perform one-hot encoding.

```
import pandas as pd

# Sample dataset with nominal attributes
data = {
    'Animal': ['Dog', 'Cat', 'Fish', 'Bird', 'Snake'],
    'Color': ['Brown', 'Black', 'Gold', 'Blue', 'Green'],
    'Habitat': ['Forest', 'Home', 'Aquarium', 'Sky', 'Jungle']
}

df = pd.DataFrame(data)

# Display the original dataset
print("Original Dataset:")
print(df)
print("\n")

# Explore the nominal attributes
nominal_attributes = ['Animal', 'Color', 'Habitat']

# Display the unique values in each nominal attribute
for attribute in nominal_attributes:
    print(f"Unique values in {attribute}: {df[attribute].unique()}")

# Perform one-hot encoding
#code here#
df_encoded = pd.get_dummies(df)


# Display the dataset after one-hot encoding
print("\nDataset after One-Hot Encoding:")
print(df_encoded)
```

```
Original Dataset:
  Animal  Color   Habitat
0    Dog  Brown    Forest
1    Cat  Black      Home
2   Fish   Gold  Aquarium
3   Bird   Blue       Sky
4  Snake  Green    Jungle


Unique values in Animal: ['Dog' 'Cat' 'Fish' 'Bird' 'Snake']
Unique values in Color: ['Brown' 'Black' 'Gold' 'Blue' 'Green']
Unique values in Habitat: ['Forest' 'Home' 'Aquarium' 'Sky' 'Jungle']

Dataset after One-Hot Encoding:
   Animal_Bird  Animal_Cat  Animal_Dog  Animal_Fish  Animal_Snake  \
0        False       False        True        False         False
1        False        True       False        False         False
2        False       False       False         True         False
3         True       False       False        False         False
4        False       False       False        False          True

   Color_Black  Color_Blue  Color_Brown  Color_Gold  Color_Green  \
0        False       False         True       False        False
1         True       False        False       False        False
2        False       False        False        True        False
3        False        True        False       False        False
4        False       False        False       False         True

   Habitat_Aquarium  Habitat_Forest  Habitat_Home  Habitat_Jungle
Habitat_Sky
0             False            True         False           False
False
1             False           False          True           False
False
2              True           False         False           False
False
3             False           False         False           False
True
4             False           False         False            True
False
```

Explanation:

```
We create a pandas DataFrame with three nominal attributes: 'Animal',
'Color', and 'Habitat'.
We explore the unique values in each nominal attribute.
We use pd.get_dummies() to perform one-hot encoding, converting each
nominal attribute into binary columns.
```

```
The resulting DataFrame (df_encoded) is displayed after one-hot
encoding.
```

2.1.2. Binary attributes

Binary Attributes:

Binary attributes are categorical variables that can take on one of two possible values, typically representing the presence or absence of a certain characteristic. These attributes are fundamental in many datasets, and examples include Yes/No, True/False, 1/0, or any other two distinct categories.

Practical Python Code for Handling Binary Attributes:

Let's create a Python code snippet using the pandas library to work with a dataset containing binary attributes. We'll load a sample dataset, explore binary attributes, and perform basic operations on them.

```
import pandas as pd

# Sample dataset with binary attributes
data = {
    'StudentID': [1, 2, 3, 4, 5],
    'PassedExam': [1, 0, 1, 1, 0],
    'EnrolledInCourse': [1, 1, 0, 1, 0]
}

df = pd.DataFrame(data)

# Display the original dataset
print("Original Dataset:")
print(df)
print("\n")

# Explore the binary attributes
binary_attributes = ['PassedExam', 'EnrolledInCourse']

# Display the count of each unique value in binary attributes
for attribute in binary_attributes:
    print(f"Counts for {attribute}:\n{df[attribute].value_counts()}\
n")

# Perform basic operations on binary attributes
#code here#

# Add a new column 'TotalAttributes' that sums the binary attributes
df['TotalAttributes'] = df[binary_attributes].sum(axis=1)


# Display the dataset after the operation
```

```
print("Dataset after performing an operation on binary attributes:")
print(df)
```

```
Original Dataset:
   StudentID  PassedExam  EnrolledInCourse
0          1           1                 1
1          2           0                 1
2          3           1                 0
3          4           1                 1
4          5           0                 0


Counts for PassedExam:
PassedExam
1    3
0    2
Name: count, dtype: int64

Counts for EnrolledInCourse:
EnrolledInCourse
1    3
0    2
Name: count, dtype: int64

Dataset after performing an operation on binary attributes:
   StudentID  PassedExam  EnrolledInCourse  TotalAttributes
0          1           1                 1                2
1          2           0                 1                1
2          3           1                 0                1
3          4           1                 1                2
4          5           0                 0                0
```

Explanation:

```
We create a pandas DataFrame with two binary attributes: 'PassedExam'
and 'EnrolledInCourse'.
We explore the counts of each unique value in binary attributes using
value_counts().
We perform a basic operation (addition) on binary attributes to create
a new attribute, 'TotalAttributes'.
The resulting DataFrame (df) is displayed after these operations.
```

2.1.3. Ordinal attributes

Ordinal Attributes:

Ordinal attributes are categorical variables with a meaningful order or ranking among the categories. Unlike nominal attributes, ordinal attributes have a clear, meaningful sequence, but the intervals between them are not necessarily uniform or well-defined. Examples of ordinal

attributes include education levels (e.g., elementary, high school, college), customer satisfaction ratings, or socioeconomic classes.

Practical Python Code for Handling Ordinal Attributes:

Let's create a Python code snippet using the pandas library to work with a dataset containing ordinal attributes. We'll load a sample dataset, explore ordinal attributes, and demonstrate how to encode them to preserve the ordinal relationship.

```
import pandas as pd

# Sample dataset with ordinal attributes
data = {
    'StudentID': [1, 2, 3, 4, 5],
    'EducationLevel': ['High School', 'College', 'Elementary',
'College', 'High School'],
    'SatisfactionRating': [3, 5, 2, 4, 1]
}

df = pd.DataFrame(data)

# Display the original dataset
print("Original Dataset:")
print(df)
print("\n")

# Explore the ordinal attributes
ordinal_attributes = ['EducationLevel', 'SatisfactionRating']

# Display the unique values in each ordinal attribute
for attribute in ordinal_attributes:
    print(f"Unique values in {attribute}: {df[attribute].unique()}")

# Encode ordinal attributes with meaningful numerical values
education_level_mapping = {'Elementary': 1, 'High School': 2,
'College': 3}
#code here#

# Add a new column 'EducationLevelEncoded' that maps the ordinal
values
df['EducationLevelEncoded'] =
df['EducationLevel'].map(education_level_mapping)


# Display the dataset after encoding ordinal attributes
print("\nDataset after encoding ordinal attributes:")
print(df)

Original Dataset:
   StudentID EducationLevel  SatisfactionRating
0          1    High School                   3
```

```
1           2        College                    5
2           3     Elementary                    2
3           4        College                    4
4           5    High School                    1


Unique values in EducationLevel: ['High School' 'College'
'Elementary']
Unique values in SatisfactionRating: [3 5 2 4 1]

Dataset after encoding ordinal attributes:
   StudentID EducationLevel  SatisfactionRating  EducationLevelEncoded
0           1    High School                   3                      2
1           2        College                   5                      3
2           3     Elementary                   2                      1
3           4        College                   4                      3
4           5    High School                   1                      2
```

Explanation:

```
We create a pandas DataFrame with two ordinal attributes:
'EducationLevel' and 'SatisfactionRating'.
We explore the unique values in each ordinal attribute.
We encode ordinal attributes with meaningful numerical values using
the map() function to create a new attribute, 'EducationLevelEncoded'.
The resulting DataFrame (df) is displayed after these operations.
```

2.1.4. Numeric attributes

Numeric Attributes:

Numeric attributes represent quantities and can take on numerical values. There are two main types of numeric attributes: discrete and continuous. Discrete numeric attributes can only take on distinct, separate values (e.g., the number of bedrooms in a house), while continuous numeric attributes can take on any value within a range (e.g., height, weight).

Practical Python Code for Handling Numeric Attributes:

Let's create a Python code snippet using the pandas library to work with a dataset containing numeric attributes. We'll load a sample dataset, explore numeric attributes, and perform basic operations on them.

```python
import pandas as pd

# Sample dataset with numeric attributes
data = {
    'StudentID': [1, 2, 3, 4, 5],
    'Age': [21, 19, 22, 20, 23],
    'Height (cm)': [175, 160, 180, 165, 185],
```

```
    'Score': [85, 92, 78, 89, 95]
}

df = pd.DataFrame(data)

# Display the original dataset
print("Original Dataset:")
print(df)
print("\n")

# Explore the numeric attributes
numeric_attributes = ['Age', 'Height (cm)', 'Score']

# Display summary statistics for numeric attributes
print("Summary Statistics for Numeric Attributes:")
#code here#
print(df[numeric_attributes].describe())

# Perform basic operations on numeric attributes
#code here#

# Add a new column 'NormalizedScore' that normalizes the 'Score'
attribute
# The formula used is: (x - min(x)) / (max(x) - min(x)) where x is the
'Score' attribute
# The normalization used is Min-Max normalization
df['NormalizedScore'] = (df['Score'] - df['Score'].min()) /
(df['Score'].max() - df['Score'].min())

# Display the dataset after the operation
print("\nDataset after performing an operation on numeric
attributes:")
print(df)
```

```
Original Dataset:
   StudentID  Age  Height (cm)  Score
0          1   21          175     85
1          2   19          160     92
2          3   22          180     78
3          4   20          165     89
4          5   23          185     95


Summary Statistics for Numeric Attributes:
             Age  Height (cm)      Score
count   5.000000     5.000000   5.000000
mean   21.000000   173.000000  87.800000
std     1.581139    10.368221   6.610598
min    19.000000   160.000000  78.000000
25%    20.000000   165.000000  85.000000
```

```
50%    21.000000   175.000000  89.000000
75%    22.000000   180.000000  92.000000
max    23.000000   185.000000  95.000000

Dataset after performing an operation on numeric attributes:
   StudentID  Age  Height (cm)  Score  NormalizedScore
0          1   21          175     85         0.411765
1          2   19          160     92         0.823529
2          3   22          180     78         0.000000
3          4   20          165     89         0.647059
4          5   23          185     95         1.000000
```

Explanation:

```
We create a pandas DataFrame with three numeric attributes: 'Age',
'Height (cm)', and 'Score'.
We explore summary statistics for numeric attributes using describe().
We perform a basic operation (normalization) on the 'Score' attribute
to create a new attribute, 'NormalizedScore'.
The resulting DataFrame (df) is displayed after these operations.
```

2.1.5. Discrete vs. continuous attributes

Discrete Attributes:

Definition: Discrete attributes can only take on distinct, separate values. Examples: The number of bedrooms in a house, the count of items in a shopping cart, the number of students in a class. Nature: These attributes are often counted in whole numbers and have clear boundaries between values.

Continuous Attributes:

Definition: Continuous attributes can take on any value within a range. Examples: Height, weight, temperature, and any measurement that can have decimal values. Nature: These attributes have a continuous and infinite set of possible values, making them suitable for measurement.

Practical Python Code for Discrete vs. Continuous Attributes:

Let's create a Python code snippet using the pandas library to work with a dataset containing both discrete and continuous attributes. We'll load a sample dataset, explore the nature of each type, and perform basic operations.

```python
import pandas as pd

# Sample dataset with discrete and continuous attributes
data = {
    'StudentID': [1, 2, 3, 4, 5],
    'NumCourses': [4, 5, 3, 6, 4],  # Discrete attribute (number of
```

```
courses)
    'GPA': [3.5, 4.0, 3.2, 3.8, 3.9],  # Continuous attribute (GPA)
    'Income': [25000, 30000, 20000, 35000, 32000]  # Continuous
attribute (income in dollars)
}

df = pd.DataFrame(data)

# Display the original dataset
print("Original Dataset:")
print(df)
print("\n")

# Explore the nature of attributes
print("Nature of Attributes:")
for column in df.columns:
    if df[column].dtype == 'int':
        print(f"{column} is a discrete attribute.")
    elif df[column].dtype == 'float':
        print(f"{column} is a continuous attribute.")

# Perform basic operations on continuous attributes
#code here#
# # Scale income for readability
df['ScaledIncome'] = (df['Income'] - df['Income'].min()) /
(df['Income'].max() - df['Income'].min())

# Display the dataset after the operation
print("\nDataset after performing an operation on continuous
attributes:")
print(df)
```

```
Original Dataset:
   StudentID  NumCourses  GPA  Income
0          1           4  3.5   25000
1          2           5  4.0   30000
2          3           3  3.2   20000
3          4           6  3.8   35000
4          5           4  3.9   32000


Nature of Attributes:
GPA is a continuous attribute.

Dataset after performing an operation on continuous attributes:
   StudentID  NumCourses  GPA  Income  ScaledIncome
0          1           4  3.5   25000      0.333333
1          2           5  4.0   30000      0.666667
2          3           3  3.2   20000      0.000000
```

```
3           4           6  3.8   35000      1.000000
4           5           4  3.9   32000      0.800000
```

Explanation:

```
We create a pandas DataFrame with three attributes: 'NumCourses'
(discrete), 'GPA' (continuous), and 'Income' (continuous).
We explore the nature of each attribute based on its data type.
We perform a basic operation (scaling) on a continuous attribute
('Income') to create a new attribute, 'ScaledIncome'.
The resulting DataFrame (df) is displayed after these operations.
```