CUSTOMER SEGMENTATION

Project Overview

Project Title: Customer Segmentation using Data Science Techniques.

Project Phase: Phase 3 – loading and preprocessing the customer data. Collect and preprocess the customer data for analysis.

Dataset Link:
https://www.kaggle.com/datasets/akram24/mall-customers

INTRODUCTION

In this Phase, We discussed about the Data Loading and the Data Preprocessing using the customer data and then also analysis the data.Loading and preprocessing customer data are fundamanetal steps in any data analysis.

Step 1: Data Loading

Data loading refers to the process of acquiring and importing data from external sources into a data storage system or a software application for further analysis, processing, or utilization. This data can come from various origins, including databases, files, web services, or APIs. The primary objective of data loading is to make the data available and accessible for use in tasks such as data analysis, reporting, machine learning, or any other data-driven processes.

Key characteristics and steps in data loading include:

1. Data Source Identification: Identifying the source of the data, which could be databases, flat files (e.g., CSV, Excel), web services, online platforms, or other data repositories.

2. Data Extraction: Extracting data from the source systems, which may involve querying databases, parsing files, or making web requests to retrieve data.

3. Data Transformation: Converting and restructuring the data as necessary to make it compatible with the target data storage or analysis platform. This may include data cleansing, data format conversion, and handling missing values.

4. Data Loading: Loading the transformed data into a data warehouse, data lake, database, or analysis tool for further processing. This often includes defining data schemas, mapping fields, and specifying data storage rules.

5. Data Validation: Ensuring the accuracy and integrity of the loaded data through various validation checks, such as data type verification, uniqueness checks, and referential integrity.

6. Data Indexing and Optimization: Creating indices and optimizing data storage structures to facilitate faster and more efficient data retrieval.

Data loading is a critical step in the data pipeline as it lays the foundation for data-driven decision-making and insights. Properly executed data loading ensures that data is readily available, accurate, and structured for analysis, reporting, and other applications, making it a cornerstone of data management and analytics processes.

Data loading and data preprocessing are essential steps in any data analysis or machine learning project. Here are some common methods and techniques used in these steps:

1. Using Libraries: Python provides several libraries for data loading, including:
   - `pandas`: Used for reading data from various file formats like CSV, Excel, SQL, and more.
   - `numpy`: Used for working with numerical data.
   - **`openpyxl`** (for Excel files), `sqlite3` (for SQL databases), and others for specific data sources.

2. File Formats: Data can be stored in various formats, such as CSV, Excel, SQL databases, JSON, or even web scraping from HTML. You should choose the appropriate method for the data source.

3. Web APIs: For data from web services, you can use libraries like `requests` to make HTTP requests and fetch data from APIs.

Step 2: Data Preprocessing

1. Handling Missing Data:

- `dropna()`: Remove rows or columns with missing values.
- `fillna()`: Fill missing values with a specific value, like the mean, median, or a constant.
- Interpolation methods: Use statistical or time-based methods to estimate missing values.

ss.py

```python
import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
df = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
# Load the Mall Customer dataset (replace 'file_path' with your actual dataset path)
# Check for missing data
missing_data = df.isnull().sum()
# Display the missing data count for each column
print("Missing Data:")
print(missing_data)
# Option 1: Remove rows with missing data
df_cleaned = df.dropna()
# Option 2: Fill missing data with the mean (or other strategy)
# Replace 'ColumnName' with the specific column name with missing data
# df['ColumnName'].fillna(df['ColumnName'].mean(), inplace=True)
# Display the cleaned dataset
print("\nCleaned Dataset:")
print(df_cleaned.head())
```

OUTPUT:

```
Run    ss ×

C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
Missing Data:
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
Unnamed: 5              200
Unnamed: 6              200
Total Spending          200
dtype: int64

Cleaned Dataset:
Empty DataFrame
Columns: [CustomerID, Genre, Age, Annual Income (k$), Spending Score (1-100), Unnamed: 5, Unnamed: 6, Total Spending]
Index: []

Process finished with exit code 0
```

EXPLANATION:

1. We load the Mall Customer dataset using pandas.

2. We check for missing data using the `isnull()` method, which returns a DataFrame of Boolean values (True for missing data, False for non-missing data).

3. We calculate and display the count of missing data for each column.

4. We provide two options for handling missing data:

   - Option 1: Removing rows with missing data using the `dropna()` method. This option is useful when you can afford to remove incomplete records.

   - Option 2: Filling missing data with the mean (or another strategy) using the `fillna()` method. You can uncomment this section and replace `'ColumnName'` with the specific column name with missing data.

Choose the option that best fits your data and analysis requirements. Handling missing data is essential to ensure that your analysis or machine learning models are based on complete and accurate data.

2. Encoding Categorical Data:

   - `One-Hot Encoding`: Convert categorical variables into binary (0/1) columns for machine learning algorithms.

   - `Label Encoding`: Assign unique integers to categorical values.

```
ss.py

import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
df = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")

# Let's assume 'Gender' is the categorical variable we want to encode
# Using one-hot encoding
data_encoded = pd.get_dummies(df, columns=['Genre'])

# Display the resulting DataFrame with one-hot encoding
print(data_encoded.head())
```

OUTPUT:

```
Run   ss

C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
   CustomerID  Age  ...  Genre_Female  Genre_Male
0           1   19  ...         False        True
1           2   21  ...         False        True
2           3   20  ...          True       False
3           4   23  ...          True       False
4           5   31  ...          True       False

[5 rows x 9 columns]

Process finished with exit code 0
```

3. Scaling and Normalization:

- `MinMax Scaling`: Scale numerical features to a specific range (e.g., [0, 1]).

- `Standardization (Z-score scaling)`: Scale numerical features to have a mean of 0 and a standard deviation of 1.

```python
from minisom import MiniSom
import matplotlib.pyplot as plt
df = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler


# Extract the numerical features to be scaled and normalized
numerical_features = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

# Standardization (Z-score scaling)
scaler_standard = StandardScaler()
scaled_data_standard = scaler_standard.fit_transform(numerical_features)
data_standard = pd.DataFrame(scaled_data_standard, columns=numerical_features.columns)

# Min-Max Scaling
scaler_minmax = MinMaxScaler()
scaled_data_minmax = scaler_minmax.fit_transform(numerical_features)
data_minmax = pd.DataFrame(scaled_data_minmax, columns=numerical_features.columns)

# Display the scaled and normalized data
print("Standardized Data:")
print(data_standard.head())

print("\nMin-Max Scaled Data:")
print(data_minmax.head())
```

OUTPUT:

```
Run    ss

C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
Standardized Data:
        Age  Annual Income (k$)  Spending Score (1-100)
0 -1.424569           -1.738999               -1.723412
1 -1.281035           -1.738999               -1.706091
2 -1.352802           -1.700830               -1.688771
3 -1.137502           -1.700830               -1.671450
4 -0.563369           -1.662660               -1.654129

Min-Max Scaled Data:
        Age  Annual Income (k$)  Spending Score (1-100)
0  0.019231            0.000000                0.000000
1  0.057692            0.000000                0.005025
2  0.038462            0.008197                0.010050
3  0.096154            0.008197                0.015075
4  0.250000            0.016393                0.020101

Process finished with exit code 0
```

4. Feature Selection and Engineering:

- Select relevant features and remove irrelevant ones.
- Create new features based on domain knowledge or data analysis.

ss.py

```python
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
df = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")

# Feature Engineering: Create a new feature "Total Spending" by adding "Annual Income" and "Spending Score"
df['Total Spending'] = df['Annual Income (k$)'] + df['Spending Score (1-100)']

# Display the first few rows of the dataset to verify the new feature
print(df.head())
```

OUTPUT:

Run ss

```
C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
   CustomerID   Genre  Age  ...  Unnamed: 5  Unnamed: 6  Total Spending
0           1    Male   19  ...         NaN         NaN              54
1           2    Male   21  ...         NaN         NaN              55
2           3  Female   20  ...         NaN         NaN              57
3           4  Female   23  ...         NaN         NaN              58
4           5  Female   31  ...         NaN         NaN              60

[5 rows x 8 columns]
```

5. Data Splitting:

- Split the dataset into training and testing sets for model evaluation.

ss.py

```python
import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
df = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
from sklearn.model_selection import train_test_split


# Split the data into features (X) and target (y)
X = df.drop('Spending Score (1-100)', axis=1)  # Features (excluding the target variable)
y = df['Spending Score (1-100)']  # Target variable

# Split the data into training and testing sets (adjust the test_size and random_state as needed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the shapes of the resulting sets to verify the split
print("Training set - X:", X_train.shape, " y:", y_train.shape)
print("Testing set - X:", X_test.shape, " y:", y_test.shape)
```

OUTPUT:

Run ss

```
C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
Training set - X: (160, 7)  y: (160,)
Testing set - X: (40, 7)  y: (40,)

Process finished with exit code 0
```

6. Date and Time Handling:

- Extract date and time features from datetime columns.
- Convert datetime to numerical values for modeling.
- In this Dataset there is no date and time

If your dataset doesn't contain any date and time information, you cannot directly perform date and time handling on it. Date and time handling functions are designed to work with columns that contain date and time data. If your dataset doesn't have such data, there's no meaningful date and time handling to be done.

If you have other specific goals or analysis you want to perform on your dataset, please provide more details about your dataset and what you are trying to achieve. I'd be happy to help with alternative data processing or analysis tasks based on the actual data in your dataset.

7. Text Data Processing:

- Tokenization: Split text into words or phrases.
- Text cleaning: Removing special characters, stop words, and stemming or lemmatization.

ss.py

```python
import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
data = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Select the 'Genre' column for text data processing
genre_data = data['Genre']

# Initialize a CountVectorizer to tokenize the text data
count_vectorizer = CountVectorizer()

# Tokenize the 'Genre' data
genre_matrix = count_vectorizer.fit_transform(genre_data)

# Convert the tokenized data to a DataFrame for better visualization (optional)
genre_df = pd.DataFrame(genre_matrix.toarray(), columns=count_vectorizer.get_feature_names_out())

# Display the tokenized data
print(genre_df)
```

OUTPUT:

```
Run    ss ×

C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
     female  male
0         0     1
1         0     1
2         1     0
3         1     0
4         1     0
..      ...   ...
195       1     0
196       1     0
197       0     1
198       0     1
199       0     1

[200 rows x 2 columns]

Process finished with exit code 0
```

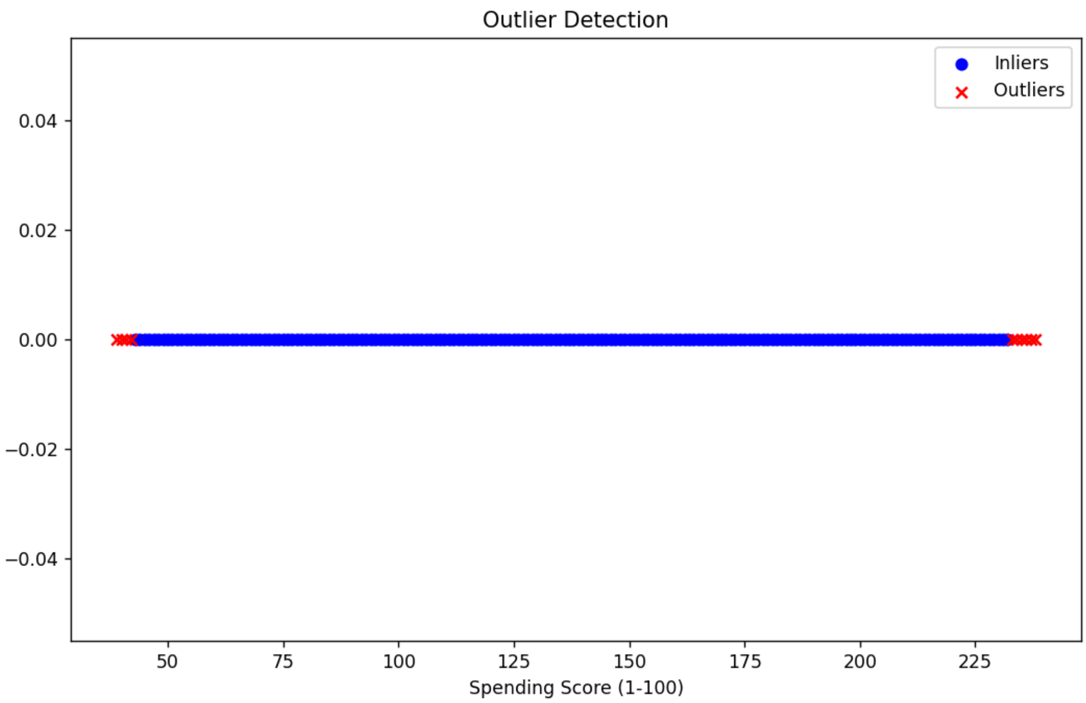
8. Outlier Detection and Handling:

- Identify and deal with outliers that may affect the model's performance.

ss.py

```python
import numpy as np
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt


# Extract a numerical feature (e.g., 'Spending Score (1-100)' for this example)
feature_to_check = 'Spending Score (1-100)'
X = data[feature_to_check].values.reshape(-1, 1)

# Initialize and fit the Isolation Forest model for outlier detection
model = IsolationForest(contamination=0.05, random_state=42)
outlier_mask = model.fit_predict(X)

# Create a mask to identify outliers (1 for inliers, -1 for outliers)
is_inlier = outlier_mask == 1

# Separate outliers and inliers
outliers = X[~is_inlier]
inliers = X[is_inlier]

# Visualize the data with outliers highlighted
plt.figure(figsize=(10, 6))
plt.scatter(inliers, np.full_like(inliers, fill_value: 0), c='b', marker='o', label='Inliers')
plt.scatter(outliers, np.full_like(outliers, fill_value: 0), c='r', marker='x', label='Outliers')
plt.xlabel(feature_to_check)
plt.title('Outlier Detection')
plt.legend()
plt.show()
```

OUTPUT:



9. Normalization and Transformation:

- Apply mathematical transformations like log transformations to make data more suitable for modeling.

ss.py

```python
import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
data = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
from sklearn.preprocessing import StandardScaler
import numpy as np
# Assuming 'Annual Income (k$)' and 'Spending Score (1-100)' columns need normalization
columns_to_normalize = ['Annual Income (k$)', 'Spending Score (1-100)']
# Perform Standardization (Normalization) on the selected columns
scaler = StandardScaler()
data[columns_to_normalize] = scaler.fit_transform(data[columns_to_normalize])
# Perform Transformation (e.g., log transformation) on a column
data['Log_Transformed_Age'] = np.log(data['Age'])

# Display the first few rows of the transformed dataset
print(data.head())
```

OUTPUT:

Run ss

```
C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
   CustomerID   Genre  Age  ...  Unnamed: 6  Total Spending  Log_Transformed_Age
0           1    Male   19  ...         NaN             NaN             2.944439
1           2    Male   21  ...         NaN             NaN             3.044522
2           3  Female   20  ...         NaN             NaN             2.995732
3           4  Female   23  ...         NaN             NaN             3.135494
4           5  Female   31  ...         NaN             NaN             3.433987

[5 rows x 9 columns]

Process finished with exit code 0
```

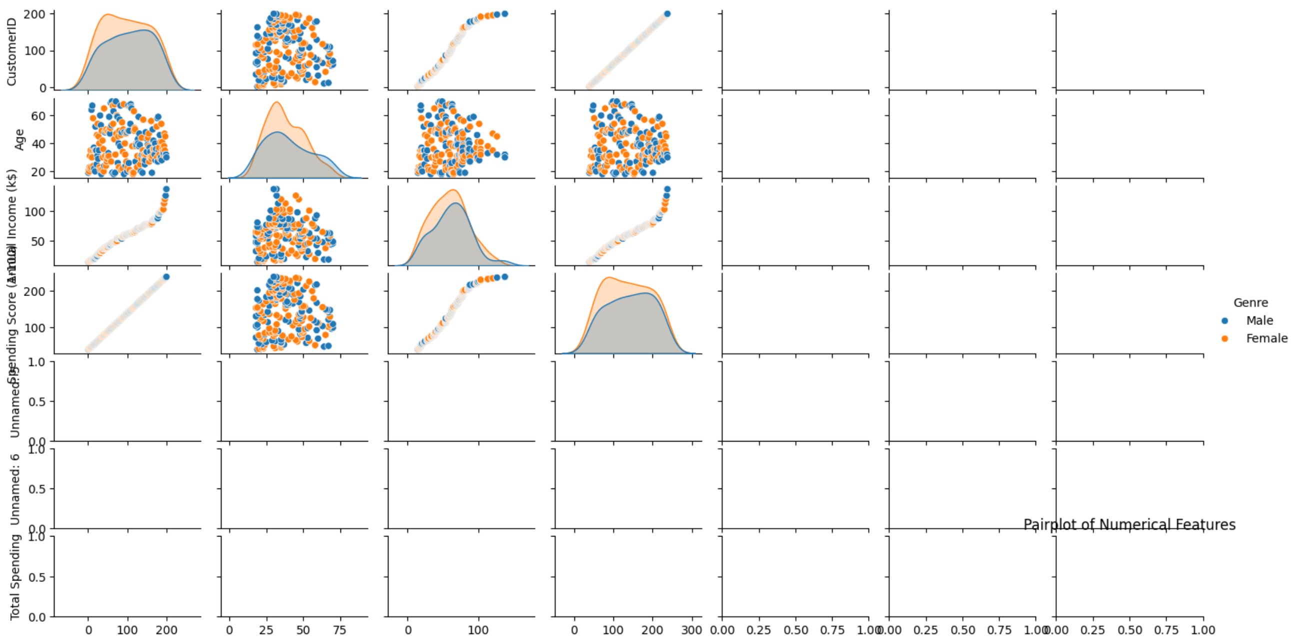
10. Data Visualization:

-Visualize the data to gain insights and detect patterns.

ss.py

```python
import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
data = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Display a pairplot to visualize pairwise relationships between numerical variables
sns.pairplot(data, diag_kind='kde', hue='Genre')
plt.title('Pairplot of Numerical Features')
plt.show()

# Create a histogram of Age distribution
plt.figure(figsize=(8, 6))
sns.histplot(data['Age'], bins=20, kde=True)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

# Create a bar plot to visualize the Gender distribution
plt.figure(figsize=(6, 4))
sns.countplot(data['Genre'])
plt.title('Genre Distribution')
plt.xlabel('Genre')
plt.ylabel('Count')
plt.show()
```

OUTPUT:



11. Scaling for Imbalanced Data:

- Techniques like oversampling or undersampling to address class imbalance in classification tasks.

```
ss.py ×    generic.py
import numpy as np
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
data = pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Select the 'Annual Income' column for scaling
income_data = data[['Annual Income (k$)']]
# Initialize the Min-Max scaler
scaler = MinMaxScaler()
# Fit and transform the data using Min-Max scaling
scaled_income = scaler.fit_transform(income_data)
# Create a new DataFrame with the scaled data
scaled_df = pd.DataFrame(scaled_income, columns=['Scaled Annual Income'])
# Concatenate the scaled data with the original DataFrame
data = pd.concat( objs: [data, scaled_df], axis=1)
# Display the first few rows of the updated DataFrame
print(data.head())
# Save the updated dataset if needed
data.to_csv( path_or_buf: 'scaled_mall_customer_dataset.csv', index=False)  # Replace with your desired output filename
```

OUTPUT:

```
Run    ss ×

C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
   CustomerID   Genre  Age  ...  Unnamed: 6  Total Spending  Scaled Annual Income
0           1    Male   19  ...         NaN             NaN              0.000000
1           2    Male   21  ...         NaN             NaN              0.000000
2           3  Female   20  ...         NaN             NaN              0.008197
3           4  Female   23  ...         NaN             NaN              0.008197
4           5  Female   31  ...         NaN             NaN              0.016393

[5 rows x 9 columns]

Process finished with exit code 0
```

12. Aggregation and Grouping:

-Summarize data by grouping it based on specific features.

ss.py × generic.py

```python
import pandas as pd
from minisom import MiniSom
import matplotlib.pyplot as plt
df= pd.read_csv(r"C:\Users\dhana\OneDrive\Documents\IBM NM\Project Phase 1\Mall_Customers.csv")
import pandas as pd
# Grouping by Gender and finding the average spending for each group
gender_grouped = df.groupby('Genre')['Spending Score (1-100)'].mean()
# Grouping by Age and finding the total spending for each age group
age_grouped = df.groupby('Age')['Spending Score (1-100)'].sum()
# Aggregating data for each customer
agg_data = df.groupby('CustomerID').agg({
    'Age': 'mean',
    'Spending Score (1-100)': 'sum'
})
# Display the results
print("Average Spending by Gender:")
print(gender_grouped)
print("\nTotal Spending by Age:")
print(age_grouped)
print("\nAggregated Data for Each Customer:")
print(agg_data)
```

OUTPUT:

Run   ss

```
C:\python3\phase2\ven\Scripts\python.exe C:\python3\phase2\ss.py
Average Spending by Gender:
Genre
Female    135.562500
Male      142.238636
Name: Spending Score (1-100), dtype: float64

Total Spending by Age:
Age
18     459
19    1080
20     486
21     451
22     224
23     575
24     350
25     414
26     256
27     888
28     801
29     747
30    1194
31     904
32    2096
33     527
34     928
35     963
36    1139
37     467
```

Run ss

```
63     258
64      47
65     228
66     293
67     412
68     382
69      96
70     208
Name: Spending Score (1-100), dtype: int64

Aggregated Data for Each Customer:
             Age  Spending Score (1-100)
CustomerID
1           19.0                      39
2           21.0                      40
3           20.0                      41
4           23.0                      42
5           31.0                      43
...          ...                     ...
196         35.0                     234
197         45.0                     235
198         32.0                     236
199         32.0                     237
200         30.0                     238

[200 rows x 2 columns]

Process finished with exit code 0
```