**National University of Computer and Emerging Sciences**



**Lab Manual 07**

Department of Computer Science

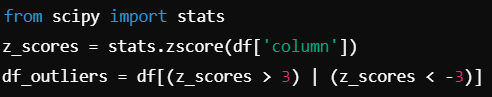
**Identifying and Handling Outliers**

**Z-Score Method**

**Z-Score**: The z-score is a statistical measure that describes a data point's relationship to the mean of the dataset. It indicates how many standard deviations a data point is from the mean. A high absolute z-score (typically greater than 3 or less than -3) indicates a potential outlier.

Formula:

where X is the data point, μ is the mean, and σ is the standard deviation.

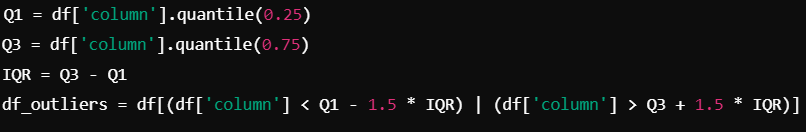


**Quantile Method (Interquartile Range - IQR)**

**Interquartile Range (IQR)**: This method defines outliers as values that fall below the 25th percentile (Q1) or above the 75th percentile (Q3) by more than 1.5 times the interquartile range.

Formula:

Outliers are defined as



Datatype Conversions

**String to Numeric**:

The pd.to\_numeric() function is used to convert string data to numeric data types (either integers or floats). This is necessary when dealing with numerical values stored as strings due to data import issues.



The errors='coerce' argument is useful for handling invalid parsing, converting them to NaN.

**Datetime Conversions**:

The pd.to\_datetime() function converts string representations of dates into pandas datetime objects. This is essential for time-series analysis and date-based operations.



**Float to Integer**:

The astype() function is used to change data types. When converting from floats to integers, it truncates the decimal points.



**Data Integration:**

**Concatenation**:

The concat() function is used to combine two or more DataFrames either along rows (axis=0) or columns (axis=1).



This adds rows from one DataFrame to another. Setting axis=1 adds columns instead.

**Joining**:

The join() function joins two DataFrames based on their indices. It is a more straightforward way to merge datasets when you have an index-based structure.

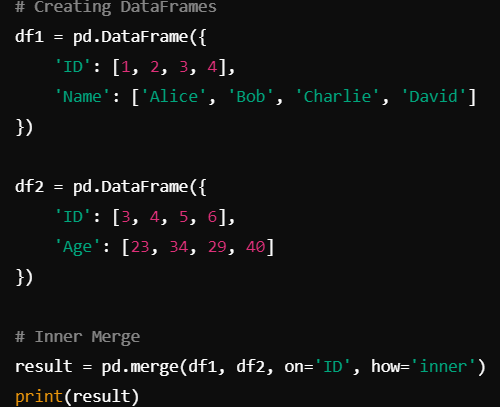


**Merging**:

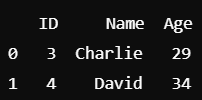
The merge() function in pandas is used to combine two DataFrames on a common column or index. It is similar to SQL joins and allows for combining data based on shared keys. You can perform different types of merges such as inner, outer, left, and right, depending on how you want to handle data from the two DataFrames. Let's explore each type of merge in detail.

**Inner Join**

An **inner join** returns only the rows that have matching values in both DataFrames based on the specified key. If a value in one DataFrame does not have a corresponding match in the other DataFrame, it will be excluded from the final result.



Output:

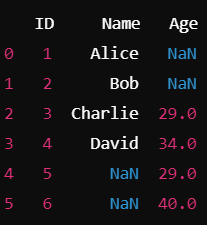


The two DataFrames df1 and df2 are merged based on the common column ID. Only rows with matching ID values in both DataFrames (ID = 3 and 4) are retained. Rows with ID = 1, 2, 5, and 6 are excluded because they do not appear in both DataFrames.

**Outer Join**

An **outer join** returns all rows from both DataFrames, filling in missing values (NaN) where there is no match. This is also known as a full outer join.

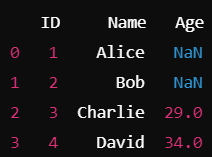
Output:



An outer join includes all rows from both DataFrames. For rows with no match in one DataFrame, the corresponding columns are filled with NaN. Here, IDs 1 and 2 from df1 and IDs 5 and 6 from df2 do not have corresponding matches, so their respective columns (Age or Name) are filled with NaN.

**Left Join**

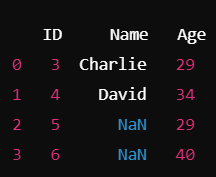
A **left join** returns all rows from the left DataFrame (the first one mentioned) and the matched rows from the right DataFrame. If there is no match in the right DataFrame, the result will have NaN values for the columns from the right DataFrame.



In the left join, all rows from df1 (left DataFrame) are included, even if there is no matching ID in df2. Here, IDs 1 and 2 do not have a corresponding match in df2, so their Age column has NaN values.

**Right Join**

A **right join** is the opposite of a left join. It returns all rows from the right DataFrame and the matched rows from the left DataFrame. If there is no match in the left DataFrame, the result will have NaN values for the columns from the left DataFrame.



In the right join, all rows from df2 (right DataFrame) are included, even if there is no matching ID in df1. Here, IDs 5 and 6 do not have a corresponding match in df1, so their Name column has NaN values.

The key difference between merge() and join() in pandas lies in how they handle the joining of DataFrames. The merge() function is used to combine DataFrames based on specific column(s), much like SQL-style joins, and provides full control over the joining process by allowing you to specify the join type (inner, outer, left, right) and the key column(s) involved in the merge. It's particularly useful when you need to merge data based on one or more common columns rather than the index.

On the other hand, join() is primarily designed for index-based joins. It merges two DataFrames using their indices by default, though it can be used with column keys by specifying the on parameter. It is a simpler method compared to merge() and is often used when the DataFrames already have aligned indices. While merge() offers more flexibility for multi-key joins, join() is ideal for quick and efficient merging when working with indexed DataFrames.

|  |  |  |
| --- | --- | --- |
| Function | Purpose | Syntax |
| dropna() | Removes missing values | df.dropna(axis=0) |
| fillna() | Fills missing values with a constant or method | df.fillna(value) |
| interpolate() | Fills missing values via interpolation | df.interpolate(method='linear') |
| drop\_duplicates() | Removes duplicate rows | df.drop\_duplicates() |
| pd.to\_numeric() | Converts string to numeric | pd.to\_numeric(df['string\_col'], errors='coerce') |
| pd.to\_datetime() | Converts string to datetime | pd.to\_datetime(df['string\_date\_col']) |
| astype() | Converts a column to another datatype | df['float\_col'].astype(int) |
| pd.concat() | Concatenates DataFrames | pd.concat([df1, df2], axis=0) |
| pd.merge() | Merges DataFrames based on key | pd.merge(df1, df2, on='key') |
| join() | Joins DataFrames based on index | df1.join(df2, how='left') |
| stats.zscore() | Calculates z-score for outlier detection | stats.zscore(df['column']) |
| quantile() | Computes quantiles for IQR-based outlier detection | df['column'].quantile(0.25) |

1. Merge the employee\_details.csv and employee\_salaries.csv datasets on the Employee\_ID column using an *inner join*. Display the resulting DataFrame. Explain why certain rows are excluded from the final result.
2. Perform an *outer join* between the employee\_details.csv and employee\_salaries.csv datasets. Ensure that all employees are included, even if they are missing in one dataset. Display the result and explain the handling of missing values in this case.
3. After merging the two datasets with an outer join, drop any rows where the Name or Salary columns have missing values. Display the cleaned DataFrame.
4. Fill missing values in the Department column with the string "Unknown" and missing values in the Salary and Bonus columns with the mean of their respective columns. Display the updated DataFrame.
5. Using the merged dataset, identify any outliers in the Salary column using the Interquartile Range (IQR) method. Remove rows with outliers and display the cleaned DataFrame.