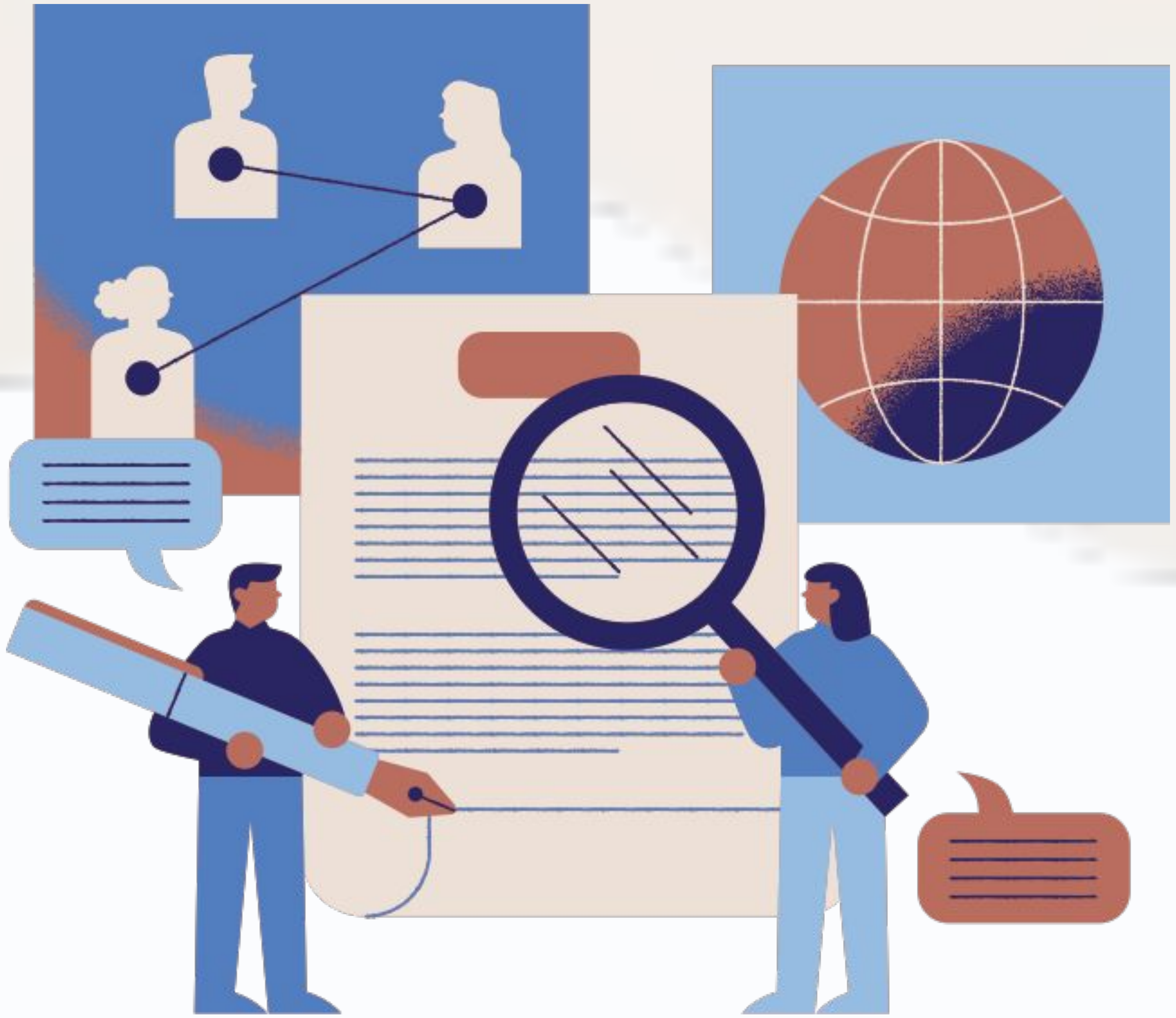
|| JAI SRI GURUDEV ||
Sri Adichunchanagiri Shikshana Trust®
SJB INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTE UNDER VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Course: Exploratory Data Analytics

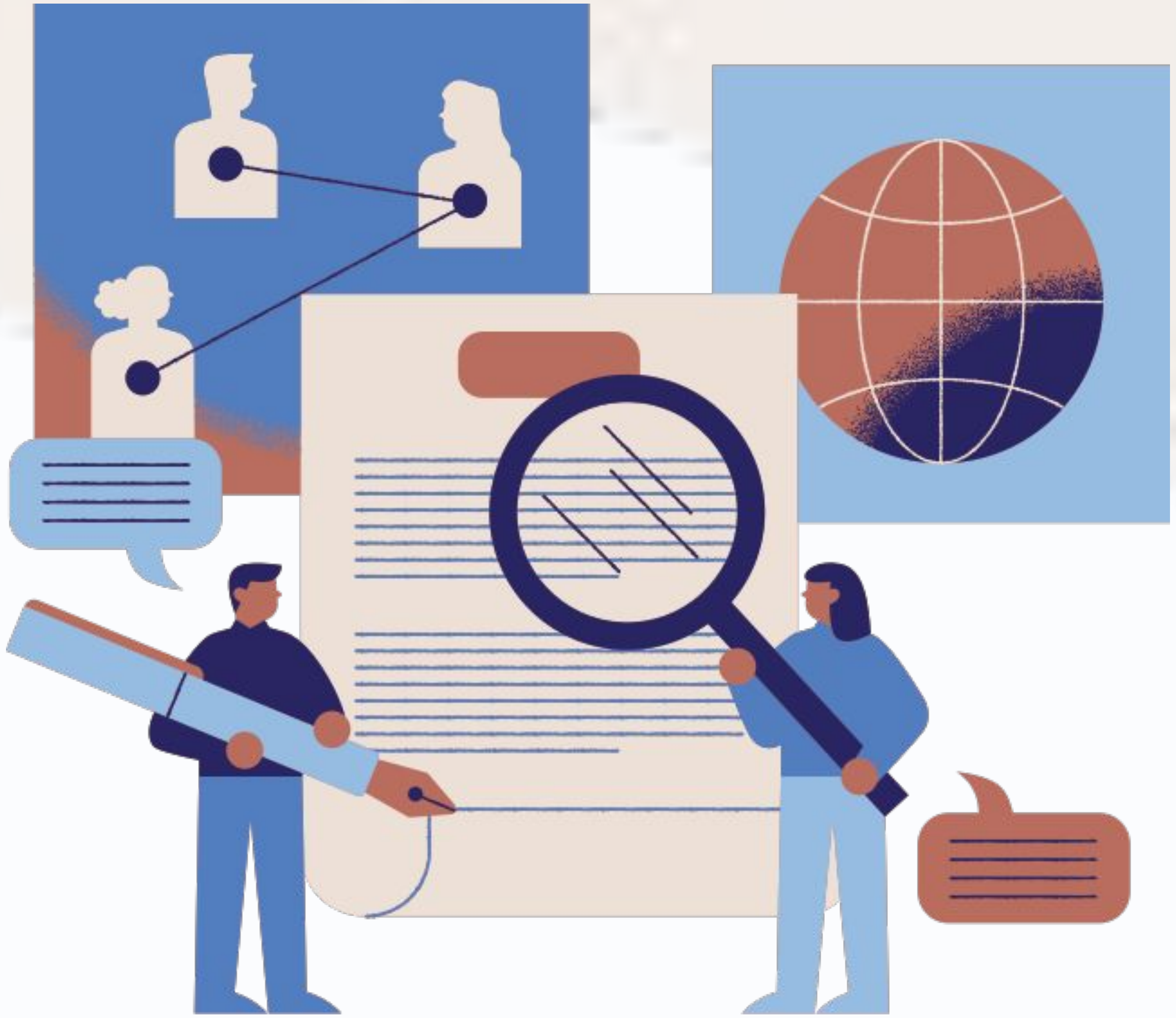
Course Code: 23CSE422



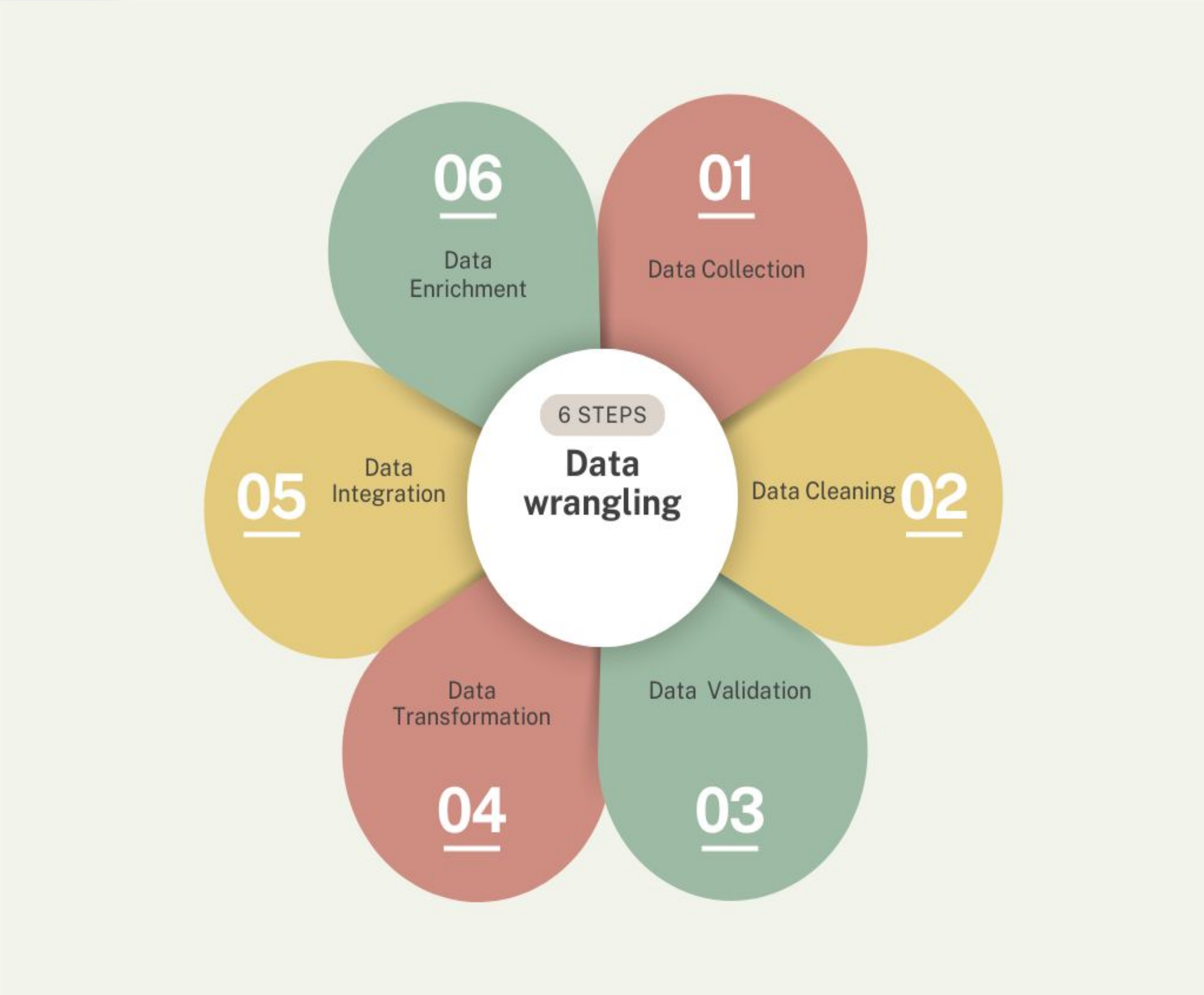
Mrs. Shilpashree S,
Assistant Professor
Department of CS & E

Data Transformation

Module 2



Data Wrangling



Data Wrangling

Data wrangling is the process of cleaning, transforming, and preparing raw data into a structured format suitable for analysis.

01

Data Collection

Gather raw data from various sources (databases, spreadsheets, APIs).

02

Data Cleaning

Identify and fix errors, such as missing values, duplicates, or inconsistencies.

03

Data validation

Ensure data adheres to predefined rules (correct formats, valid ranges, consistency).

04

Data Transformation

Convert data into a usable format (e.g., normalization, data type conversion).

05

Data Integration

Combine data from multiple sources into a unified dataset.

06

Data Enrichment

Enhance data with additional information (e.g., merging external datasets, calculated fields).

Technical Requirements

Pandas

Matplotlib

NumPy

Seaborn

Data Transformation

Data transformation is a set of techniques used to convert data from one format or structure to another, ensuring compatibility and enhancing interoperability.

01 **Data Deduplication**

Identify and remove duplicate data.

02 **Key Restructuring**

- Transform keys with built-in meanings into generic keys.

03 **Data Cleansing**

- Remove outdated, inaccurate, or incomplete information without changing its meaning.

04 **Data Validation**

- Formulate rules or algorithms to validate data against known issues.

08 **Format Revisioning**

- Convert data from one format to another.

06 **Data Derivation**

- Create rules to generate additional information from the source data.

07 **Data Aggregation**

- Searching, extracting, summarize and preserve important information for reporting systems.

08 **Data Integration**

- Convert and merge different data types into a common structure or schema.

09 **Data Filtering**

- Identify and extract relevant information for specific users.

10 **Data Joining**

- Establish relationships between two or mor tables.

Merging Database-Style DataFrames

Case Study

Scenario:

- You are a professor teaching two courses:
  - Software Engineering (SE)
  - Introduction to Machine Learning (ML)

Scenario 1: Vertical Merging (Concatenation)

- Context:
  - You are teaching Software Engineering (SE) and have two different dataframes for the same course.
  - The first dataframe represents the first group of students who appeared for the exam, and the second represents the second group of students.
- Objective:
  - Combine the two dataframes vertically (stack them on top of each other), as both contain the same structure (same columns).

Vertical Concatenation

StudentID	ScoreSE
1	89
3	39
5	50
7	97
9	20
...	...
...	...
27	73
29	92

StudentID	ScoreSE
2	98
4	93
6	44
8	77
10	69
...	...
...	...
28	56
30	27

StudentID	ScoreSE
1	89
3	39
5	50
7	97
9	20
...	...
...	...
27	73
29	92
2	98
4	93
6	44
8	77
10	69
...	...
...	...
28	56
30	27

Merging Database-Style DataFrames

Case Study

Scenario:

- You are a professor teaching two courses:
  - Software Engineering (SE)
  - Introduction to Machine Learning (ML)

Scenario 2: Horizontal Merging (Side-by-Side Concatenation)

- Context:
  - You are teaching two different courses: Software Engineering (SE) and Introduction to Machine Learning (ML).
  - You have two dataframes for each course. Some students have taken both courses, and some have taken only one.
- Objective:
  - Combine the two dataframes horizontally (side-by-side), where the same students will be merged across both subjects.

Horizontal Concatenation

StudentID	ScoreSE
9	22
11	66
13	31
15	51
17	71
...	...
...	...
27	73
29	92

StudentID	ScoreSE
2	98
4	93
6	44
8	77
10	69
...	...
...	...
28	56
30	27

StudentID	ScoreML
1	39
3	49
5	55
7	77
9	52
...	...
...	...
27	23
29	49

StudentID	ScoreML
2	98
4	93
6	44
8	77
10	69
...	...
...	...
28	56
30	27

	StudentID	ScoreML	StudentID	ScoreSE
0	1.0	39.0	9	22
1	3.0	49.0	11	66
2	5.0	55.0	13	31
3	7.0	77.0	15	51
4	9.0	52.0	17	71
5	11.0	86.0	19	91
6	13.0	41.0	21	56
7	15.0	77.0	23	32
8	17.0	73.0	25	52
9	19.0	51.0	27	73
10	21.0	86.0	29	92
11	23.0	82.0	2	98
12	25.0	92.0	4	93
13	27.0	23.0	6	44
14	29.0	49.0	8	77
15	2.0	93.0	10	69
16	4.0	44.0	12	56

Merging Database-Style DataFrames

Key Differences Between Vertical & Horizontal Merging:

- Vertical Merging (Concatenation):
  - Stacks dataframes on top of each other.
  - Used when you have the same columns in both dataframes but more rows to combine.
  - Example: Combining students from different buildings for the same subject.
- Horizontal Merging (Side-by-Side Concatenation):
  - Joins dataframes by columns.
  - Used when you have data for the same students (rows) but in separate columns (like scores for different subjects).
  - Example: Combining students' scores from two different courses (SE and ML).

Concatenating along with an axis

pd.concat() Function

pd.concat() is used to concatenate pandas DataFrames along a particular axis (rows or columns).

Syntax:

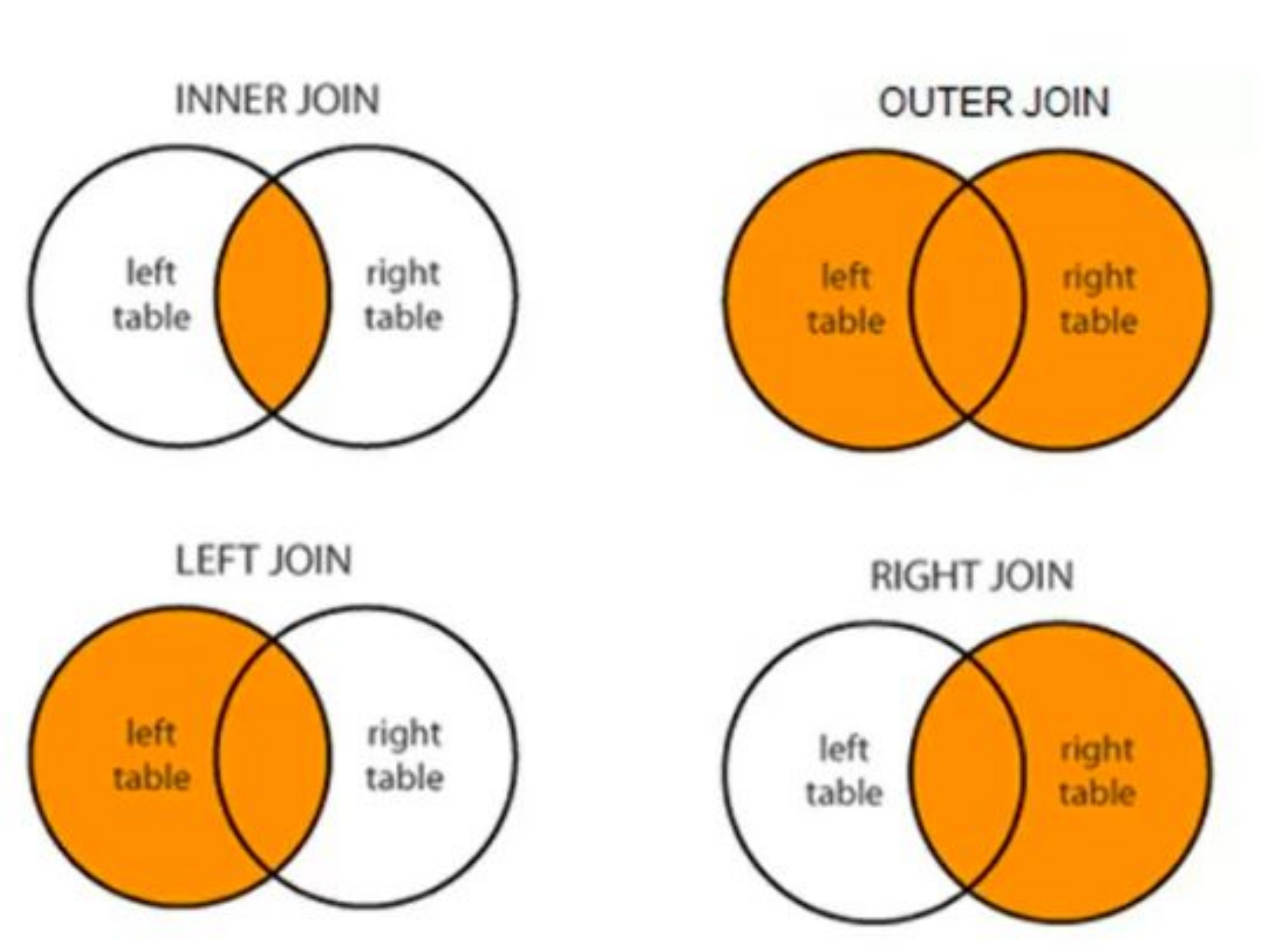
```
pd.concat([df1, df2], axis=0, ignore_index=False)
```

- axis=0 (default): Concatenates vertically (row-wise).
- axis=1: Concatenates horizontally (column-wise).
- ignore_index=True: Resets the index to a new integer sequence, avoiding duplication of indices.

When to Use pd.concat()?

- When you need to combine data from multiple sources with similar structure.
- When you want to combine data along rows or columns.
- When handling large datasets across multiple files or regions, concatenation helps combine data for further analysis.

Using df.merge



Using df.merge with an inner join

What is an Inner Join?

- An inner join combines rows from two DataFrames where the key column(s) match in both.
- Only the rows that exist in both DataFrames are included in the result.

The df.merge() Method

Syntax:

```
df1.merge(df2, how='inner', on='common_column')
```

Parameters:

- how: Defines the type of join ('inner', 'left', 'right', 'outer').
- on: Specifies the column(s) to join on.
- how='inner': Keeps only the rows with matching values in both DataFrames.

Using df.merge with an inner join

Example Scenario

Consider two dataframes:

- dfSE: Data for Software Engineering students.
- dfML: Data for Machine Learning students.

You want to find students who have appeared in both courses.

1. Concatenating the DataFrames:
   - First, we concatenate two DataFrames for each course (Software Engineering and Machine Learning) using pd.concat().

```
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)
```

2. Merging the DataFrames with Inner Join:
   - Use the merge() function to find students who took both courses.

```
df = dfSE.merge(dfML, how='inner')
```

- The result will only contain students who are present in both dfSE (Software Engineering) and dfML (Machine Learning).
- Students who appear in both DataFrames will be merged into a single row.

Using df.merge with an inner join

The pd.merge() method in Pandas allows you to merge DataFrames in various ways using different types of joins. These joins are similar to SQL join operations, and each type serves a different purpose when combining data:

- Inner Join
  - Intersection of two or more DataFrames (only includes rows with matching keys).
  - It corresponds to the INNER JOIN in Structured Query Language (SQL).
- Outer Join
  - Union of two or more DataFrames (includes all rows, matching or not).
  - It corresponds to the FULL OUTER JOIN in SQL.
- Left Join
  - Uses keys from the left-hand DataFrame (only includes all rows from the left DataFrame, with matching data from the right DataFrame; non-matching rows from the right are filled with NaN).
  - It corresponds to the LEFT OUTER JOIN in SQL.
- Right Join
  - Uses keys from the right-hand DataFrame (only includes all rows from the right DataFrame, with matching data from the left DataFrame; non-matching rows from the left are filled with NaN).
  - It corresponds to the RIGHT OUTER JOIN in SQL.

pd.merge() with a Left Join:

In a left join, you merge two DataFrames, but only the keys from the left DataFrame are used. Any rows in the left DataFrame without corresponding matches in the right DataFrame will still appear in the result, but with NaN for the missing values.

Syntax:

```
df = dfSE.merge(dfML, how='left')
```

Here, dfSE represents the Software Engineering DataFrame and dfML represents the Machine Learning DataFrame. The how='left' argument tells Pandas to perform a left join, where all students who appeared for the Software Engineering exam will be included in the merged DataFrame, and those who did not appear for the Machine Learning exam will have their scores marked as NaN.

Code:

```
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)
df = dfSE.merge(dfML, how='left')
```

Using pd.merge() with a Left Join

```
import pandas as pd

df1SE = pd.DataFrame({ 'StudentID': [9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
29], 'ScoreSE' : [22, 66, 31, 51, 71, 91, 56, 32, 52, 73, 92]})
df2SE = pd.DataFrame({'StudentID': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22,
24, 26, 28, 30], 'ScoreSE': [98, 93, 44, 77, 69, 56, 31, 53, 78, 93, 56,
77, 33, 56, 27]})

df1ML = pd.DataFrame({ 'StudentID': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21,
23, 25, 27, 29], 'ScoreML' : [39, 49, 55, 77, 52, 86, 41, 77, 73, 51, 86,
82, 92, 23, 49]})
df2ML = pd.DataFrame({'StudentID': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
'ScoreML': [93, 44, 78, 97, 87, 89, 39, 43, 88, 78]})`
```

```
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='left')
df
```

	StudentID	ScoreSE
0	9	22
1	11	66
2	13	31
3	15	51
4	17	71
5	19	91
6	21	56
7	23	32
8	25	52
9	27	73
10	29	92
11	2	98
12	4	93
13	6	44
14	8	77
15	10	69
16	12	56
17	14	31
18	16	53
19	18	78
20	20	93
21	22	56
22	24	77
23	26	33
24	28	56
25	30	27

Using pd.merge() with a Right Join

we can use the right join to get a list of all the students who appeared in the Machine Learning course.

Syntax:

```
df = dfSE.merge(dfML, how='right')
```

Code:

```
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)
df = dfSE.merge(dfML, how='right')
```

In this right join, all the students from the Machine Learning course (dfML) are retained in the merged DataFrame, while students who did not appear for the Software Engineering course have NaN for the columns from dfSE.

Using pd.merge() methods with outer join

pd.merge() with an Outer Join:

Syntax:

```
df = dfSE.merge(dfML, how='outer')
```

- The dfSE DataFrame represents the Software Engineering course data.
- The dfML DataFrame represents the Machine Learning course data.
- how='outer' specifies an outer join, which will include all students from both courses, filling with NaN where there is no match in either DataFrame.

Code:

```
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)
df = dfSE.merge(dfML, how='outer')
```

Merging on Index

Merging on index occurs when the keys for merging DataFrames are located in the index of one or both DataFrames.

We use:

- left_index=True
- right_index=True

to indicate the index should act as the merge key.

DataFrames for Merging on Index

Code:

```
left1 = pd.DataFrame({'key': ['apple', 'ball', 'apple', 'apple', 'ball', 'cat'], 'value': range(6) })
right1 = pd.DataFrame({'group_val': [33.4, 5] }, index=['apple', 'ball'])
```

Merging on Index

Output of `left1`:

	key	value
0	apple	0
1	ball	1
2	apple	2
3	apple	3
4	ball	4
5	cat	5

Output of `right1`:

	group_val
apple	33.4
ball	5

Merging on Index

Inner Join on Index

- The default merge operation is an inner join, which results in the intersection of the keys.

Code:

```
df = pd.merge(left1, right1, left_on='key', right_index=True)
df
```

	key	value	group_val
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0

Merging on Index

Outer Join on Index

- An outer join includes all keys from both DataFrames, filling missing values with NaN.

Code:

```
df = pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
df
```

	key	value	group_val
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0
5	cat	5	NaN

Reshaping and Pivoting

- Reshaping and pivoting are essential during Exploratory Data Analysis (EDA) to rearrange data for better insights.

Two key operations for hierarchical indexing:

- Stacking: Moves data from columns to rows.
- Unstacking: Moves data from rows to columns.

Example

Create a DataFrame to record rainfall, humidity, and wind conditions across five counties in Norway

```
data = np.arange(15).reshape((3, 5))
indexers = ['Rainfall', 'Humidity', 'Wind']
dframe1 = pd.DataFrame(data, index=indexers, columns=['Bergen', 'Oslo', 'Trondheim', 'Stavanger', 'Kristiansand'])
dframe1
```

	Bergen	Oslo	Trondheim	Stavanger	Kristiansand
Rainfall	0	1	2	3	4
Humidity	5	6	7	8	9
Wind	10	11	12	13	14

Reshaping and Pivoting

Stacking Columns into Rows

Code:

```
stacked = dframe1.stack()
stacked
```

```
Rainfall  Bergen          0
          Oslo            1
          Trondheim       2
          Stavanger       3
          Kristiansand    4
Humidity  Bergen          5
          Oslo            6
          Trondheim       7
          Stavanger       8
          Kristiansand    9
Wind      Bergen         10
          Oslo           11
          Trondheim      12
          Stavanger      13
          Kristiansand   14
dtype: int64
```

Reshaping and Pivoting

Unstacking Rows into Columns

- This reverts the stacked data back to its original DataFrame structure.

Code:

```
stacked.unstack()
```

```
Rainfall  Bergen          0
          Oslo            1
          Trondheim       2
          Stavanger       3
          Kristiansand    4
Humidity  Bergen          5
          Oslo            6
          Trondheim       7
          Stavanger       8
          Kristiansand    9
Wind      Bergen         10
          Oslo           11
          Trondheim      12
          Stavanger      13
          Kristiansand   14
dtype: int64
```

Reshaping and Pivoting

Handling Missing Data in Unstacking

Code:

```
series1 = pd.Series([0, 111, 222, 333], index=['zeros', 'ones', 'twos', 'threes'])
series2 = pd.Series([444, 555, 666], index=['fours', 'fives', 'sixes'])
frame2 = pd.concat([series1, series2], keys=['Number1', 'Number2'])
frame2
frame2.unstack()
```

	fives	fours	ones	sixs	threes	twos	zeros
Number1	NaN	NaN	111.0	NaN	333.0	222.0	0.0
Number2	555.0	444.0	NaN	666.0	NaN	NaN	NaN