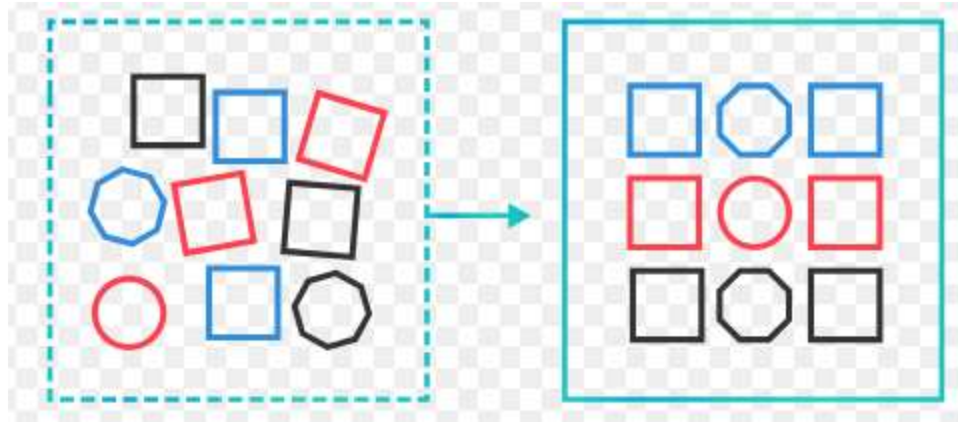
Data Transformation Techniques



1. Introduction
2. Transformation Activities
3. Merging Database
4. Reshaping and Pivoting
5. Transformation Techniques
6. Benefits & Challenges

1. Introduction

- Process of converting raw data into usable form
- Converts data into a intelligence format
- Convert from one format to another format

Data wrangling:

- The process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time.
- The process of removing errors and combining complex data sets to make them more accessible and easier to analyze.

2.Transformation Activities

1. Data Deduplication: involves the identification of duplicates and their removal.
2. Data cleansing: involves extracting words and deleting out-of-date, inaccurate, and incomplete information from the source to enhance the accuracy of the source data.
3. Data validation: is a process of formulating rules or algorithms that help in validating different types of data against some known issues.
4. Format revisioning: involves converting from one format to another.

2.Transformation Activities

5. Data derivation: consists of creating a set of rules to generate more information from the data source.
6. Data aggregation: involves searching, extracting, summarizing, and preserving important information in different types of reporting systems.
7. Data integration: involves converting different data types and merging them into a common structure or schema.
8. Data filtering involves identifying information relevant to any particular user.
9. Data joining: involves establishing a relationship between two or more tables.

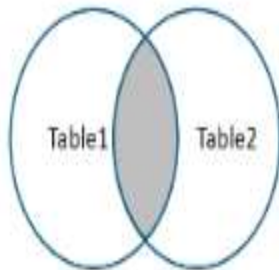
3.Merging Database

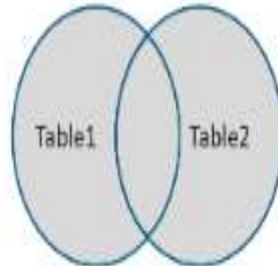
1. Concatenating along with an axis
2. Using df.merge with an **inner join - ∩ of two d.f**
3. Using the pd.merge() method with a **left join (uses the key from the lefthand data frame only)**
4. Using the pd.merge() method with a **right join (uses the key from the righthand data frame only)**
5. Using pd.merge() methods with **outer join -∪ of two d.f**
6. Merging on index

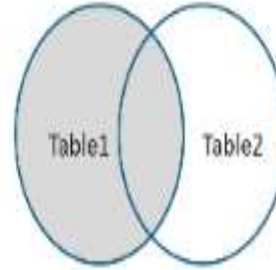
3.Merging Database

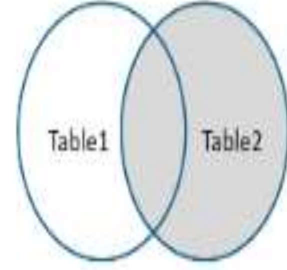
Combine data frames in 4 ways

1. If **combine Side by side (axis=1)** then we can use **merge or join**
2. If **combine Stacking (axis=0)** then we can use **concat or append**









3.Merging Database

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

The structure of the dataframes is the same in both cases. In this case, we would need to concatenate them. We can do that by using the pandas concat() method:

```
dataframe = pd.concat([dataFrame1, dataFrame2], ignore_index=True)
dataframe
```

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

If we combined the data frames along axis=0, then it combined them together in the same direction.
If axis = 1, then to combine them side by side.

	StudentID	Score
0	1	89
1	3	39
2	5	50
3	7	97
4	9	22
5	11	66
6	13	31
7	15	51
8	17	71
9	19	91
10	21	56
11	23	32
12	25	52
13	27	73
14	29	92
0	2	98
1	4	93
2	6	44
3	8	77
4	10	69
5	12	56
6	14	31
7	16	53
8	18	78
9	20	93
10	22	56
11	24	77
12	26	33
13	28	56
14	30	27

pd.concat([dataFrame1, dataFrame2], axis=0)

	StudentID	Score	StudentID	Score
0	1	89	2	98
1	3	39	4	93
2	5	50	6	44
3	7	97	8	77
4	9	22	10	69
5	11	66	12	56
6	13	31	14	31
7	15	51	16	53
8	17	71	18	78
9	19	91	20	93
10	21	56	22	56
11	23	32	24	77
12	25	52	26	33
13	27	73	28	56
14	29	92	30	27

pd.concat([dataFrame1, dataFrame2], axis=1)

Now, consider another use case where you are teaching two courses: Software Engineering and Introduction to Machine Learning. You will get two dataframes from each subject: Two for the Software Engineering course Another two for the Introduction to Machine Learning course

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

1. There are some students who are not taking the software engineering exam.
2. There are some students who are not taking the machine learning exam.
3. There are students who appeared in both courses.
4. How many students appeared for the exams in total?
5. How many students only appeared for the Software Engineering course?
6. How many students only appeared for the Machine Learning course?

Ans:- df.merge (concatenate the individual dataframes from each of the subjects, and then use df.merge() methods)

option 1 : Concatenating along with an axis =1

	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

The StudentID field is repeated.

Option 2: Using df.merge with an inner join

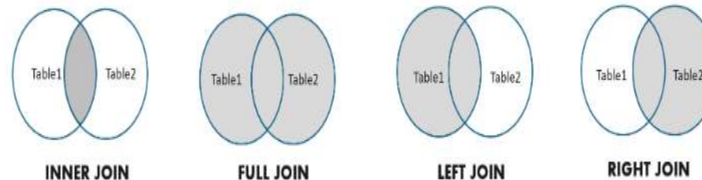
	StudentID	ScoreSE	ScoreML
0	9	22	52
1	11	66	86
2	13	31	41
3	15	51	77
4	17	71	73
5	19	91	51
6	21	56	86
7	23	32	82
8	25	52	92
9	27	73	23
10	29	92	49
11	2	98	93
12	4	93	44
13	6	44	78
14	8	77	97
15	10	69	87
16	12	56	89
17	14	31	39
18	16	53	43
19	18	78	88
20	20	93	78

← concatenate by ignore_index=True and merge **Inner join.**
Ans3: we now know there are 21 students who took both the courses.

Left join →
you can correctly answer how many students only appeared for the **Software Engineering course.** The total number would be 26. Note that these students did not appear for the Machine Learning exam and hence their scores are marked as NaN.

Right join : to get a list of all the students who appeared in the **Machine Learning course**

Outter join : the total number of students appearing for at least one course.



	StudentID	ScoreSE	ScoreML
0	9	22	52.0
1	11	66	86.0
2	13	31	41.0
3	15	51	77.0
4	17	71	73.0
5	19	91	51.0
6	21	56	86.0
7	23	32	82.0
8	25	52	92.0
9	27	73	23.0
10	29	92	49.0
11	2	98	93.0
12	4	93	44.0
13	6	44	78.0
14	8	77	97.0
15	10	69	87.0
16	12	56	89.0
17	14	31	39.0
18	16	53	43.0
19	18	78	88.0
20	20	93	78.0
21	22	56	NaN
22	24	77	NaN
23	26	33	NaN
24	28	56	NaN
25	30	27	NaN

Merging on index

- Sometimes the keys for merging dataframes are located in the dataframes index. In such a situation, we can pass left_index=True or right_index=True to indicate that the index should be accepted as the merge key
- Merging on index is done in the following steps:

Consider the following two dataframes:

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

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

	group_val
apple	33.4
ball	5.0

Inner join

Now, let's consider two different cases. Firstly, let's try merging using an inner join, which is the default type of merge. In this case, the default merge is the intersection of the keys. Check the following example 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

The output is the intersection of the keys from these dataframes. Since there is no cat key in the second dataframe, it is not included in the final table.

Outter join

```
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

Note that the last row includes the cat key. This is because of the outer join.

4.Reshaping and Pivoting

- During EDA, we often need to rearrange data in a dataframe in some consistent manner. This can be done with hierarchical indexing using two actions:
  - Stacking: Stack rotates from any particular column in the data to the rows.
  - Unstacking: Unstack rotates from the rows into the column.

4.Reshaping and Pivoting

```
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

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

The output of this stacking is as follows:

```
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
```

5.Transformation Techniques

1. Performing data deduplication
2. Replacing values
3. Handling missing data
    1. NaN values in pandas objects
    2. Dropping missing values
        1. Dropping by rows
        2. Dropping by columns
    3. Mathematical operations with NaN
    4. Filling missing values
    5. Backward and Forward filling
    6. Interpolating missing values

5.Transformation Techniques

4. Renaming axis indexes
5. Discretization and binning
6. Outlier detection and filtering
7. Permutation and random sampling
    1. Random sampling without replacement
    2. Random sampling with replacement
8. Computing indicators/dummy variables
9. String manipulation

1.Performing data deduplication

– If dataframe contains duplicate rows, removing them.

To check duplicates: frame3.duplicated()

	column 1	column 2
0	Looping	10
1	Looping	10
2	Looping	22
3	Functions	23
4	Functions	23
5	Functions	24
6	Functions	24

```
0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

drop these duplicates using the drop_duplicates()

	column 1	column 2
0	Looping	10
2	Looping	22
3	Functions	23
5	Functions	24

2. Replacing values

– it is essential to find and replace some values inside a dataframe.

```
replaceFrame =
pd.DataFrame({'column 1': [200.,
3000., -786., 3000., 234., 444., -
786., 332., 3332. ], 'column 2':
range(9)})
replaceFrame
```

	column 1	column 2
0	200.0	0
1	3000.0	1
2	-786.0	2
3	3000.0	3
4	234.0	4
5	444.0	5
6	-786.0	6
7	332.0	7
8	3332.0	8

```
replaceFrame.replace(to_replace =-
786, value= np.nan)
```

	column 1	column 2
0	200.0	0
1	3000.0	1
2	NaN	2
3	3000.0	3
4	234.0	4
5	444.0	5
6	NaN	6
7	332.0	7
8	3332.0	8

3. Handling missing data

– 3.1 Whenever there are missing values, a NaN value is used, which indicates Not A Number.

- It can happen when data is retrieved from an external source and there are some incomplete values in the dataset.
- It can also happen when we join two different datasets and some values are not matched.
- Missing values due to data collection errors.
- When the shape of data changes, there are new additional rows or columns that are not determined.
- Reindexing of data can result in incomplete data.

	store1	store2	store3	store4	store5
apple	False	False	False	False	True
banana	False	False	False	True	True
kiwi	False	False	False	True	True
grapes	False	False	False	True	True
mango	False	False	False	True	True
watermelon	False	False	False	False	True
oranges	True	True	True	True	True

dfx.isnull()

	store1	store2	store3	store4	store5
apple	True	True	True	True	False
banana	True	True	True	False	False
kiwi	True	True	True	False	False
grapes	True	True	True	False	False
mango	True	True	True	False	False
watermelon	True	True	True	True	False
oranges	False	False	False	False	False

dfx.notnull()

3. Handling missing data

- Whenever there are missing values, a NaN value is used, which indicates Not A Number.
  - It can happen when data is retrieved from an external source and there are some incomplete values in the dataset.
  - It can also happen when we join two different datasets and some values are not matched.
  - Missing values due to data collection errors.
  - When the shape of data changes, there are new additional rows or columns that are not determined.
  - Reindexing of data can result in incomplete data.
- dfx.isnull()
- dfx.notnull()
- dfx.isnull().sum() →

```
store1 1
store2 1
store3 1
store4 5
store5 7
dtype: int64
```

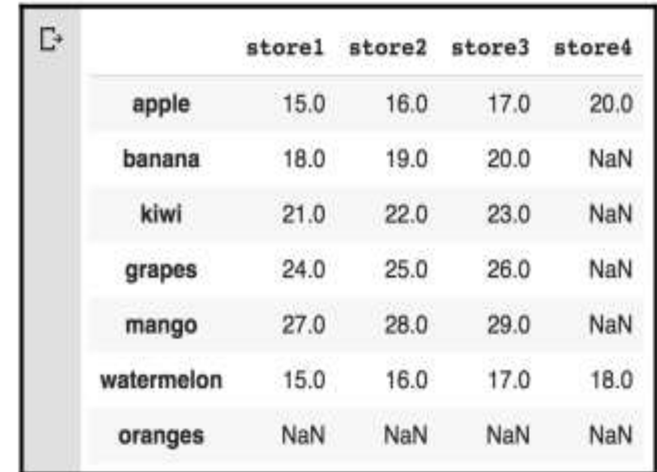
- dfx.isnull().sum().sum() → 15
- dfx.count() →

```
store1 6
store2 6
store3 6
store4 2
store5 0
dtype: int64
```

3.2 Dropping missing values

- To simply remove them from our dataset.
- dropna() method just returns a copy of the dataframe by dropping the rows with NaN.
- Dropping by rows------- dfx.dropna(how='all')
- Dropping by columns------------dfx.dropna(how='all', axis=1)

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	NaN	NaN
kiwi	21.0	22.0	23.0	NaN	NaN
grapes	24.0	25.0	26.0	NaN	NaN
mango	27.0	28.0	29.0	NaN	NaN
watermelon	15.0	16.0	17.0	18.0	NaN



	store1	store2	store3	store4
apple	15.0	16.0	17.0	20.0
banana	18.0	19.0	20.0	NaN
kiwi	21.0	22.0	23.0	NaN
grapes	24.0	25.0	26.0	NaN
mango	27.0	28.0	29.0	NaN
watermelon	15.0	16.0	17.0	18.0
oranges	NaN	NaN	NaN	NaN

3.3 Mathematical operations with NaN

– To simply remove them from our dataset.

```
ar1 = np.array([100, 200, np.nan, 300])
ser1 = pd.Series(ar1) ar1.mean(), ser1.mean()
(nan, 200.0)
```

3.3 Filling missing values

We can use the fillna() method to replace NaN values with any particular values

```
filledDf = dfx.fillna(0)
filledDf
```

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	0.0
banana	18.0	19.0	20.0	0.0	0.0
kiwi	21.0	22.0	23.0	0.0	0.0
grapes	24.0	25.0	26.0	0.0	0.0
mango	27.0	28.0	29.0	0.0	0.0
watermelon	15.0	16.0	17.0	18.0	0.0
oranges	0.0	0.0	0.0	0.0	0.0

- Backward and Forward filling
  - NaN values can be filled based on the last known values.
- Interpolating missing values
  - ser3 = pd.Series([100, np.nan, np.nan, np.nan, 292]) ser3.interpolate())
  - (292-100)/(5-1) = 48

```
0 100.0
1 148.0
2 196.0
3 244.0
4 292.0
dtype: float64
```

6. Benefits of Data Transformation

1. It promotes interoperability between several applications. The main reason for creating a similar format and structure in the dataset is that it becomes compatible with other systems.
2. Comprehensibility for both humans and computers is improved when using better-organized data compared to messier data (confused data)

6. Benefits of Data Transformation

3. It ensures a higher degree of data quality and protects applications from several computational challenges such as null values, unexpected duplicates, and incorrect indexings, as well as incompatible structures or formats..
4. It ensures higher performance and scalability for modern analytical databases and dataframes..

7. Challenges

1. It requires a qualified team of experts and state-of-the-art infrastructure. The cost of attaining such experts and infrastructure can increase the cost of the operation.
2. It requires data cleaning before data transformation and data migration. This process of cleansing can be expensively time-consuming.
3. Generally, the activities of data transformations involve batch processing. This means that sometimes, we might have to wait for a day before the next batch of data is ready for cleansing. This can be very slow.

df1 (left)

	ID	Col_1	Col_2	Col_3	Col_4
0	1	1	6	11	apple
1	2	2	7	12	orange
2	3	3	8	13	banana
3	5	4	9	14	strawberry
4	9	5	10	15	raspberry

df2 (right)

	ID	Col_A	Col_B	Col_4
0	1	8	12	apple
1	1	9	13	orange
2	3	10	15	banana
3	5	11	17	kiwi

df1 (left)

	ID	Col_1	Col_2	Col_3	Col_4
0	1	1	6	11	apple
1	2	2	7	12	orange
2	3	3	8	13	banana
3	5	4	9	14	strawberry
4	9	5	10	15	raspberry

df2 (right)

	ID	Col_A	Col_B	Col_4
0	1	8	12	apple
1	1	9	13	orange
2	3	10	15	banana
3	5	11	17	kiwi