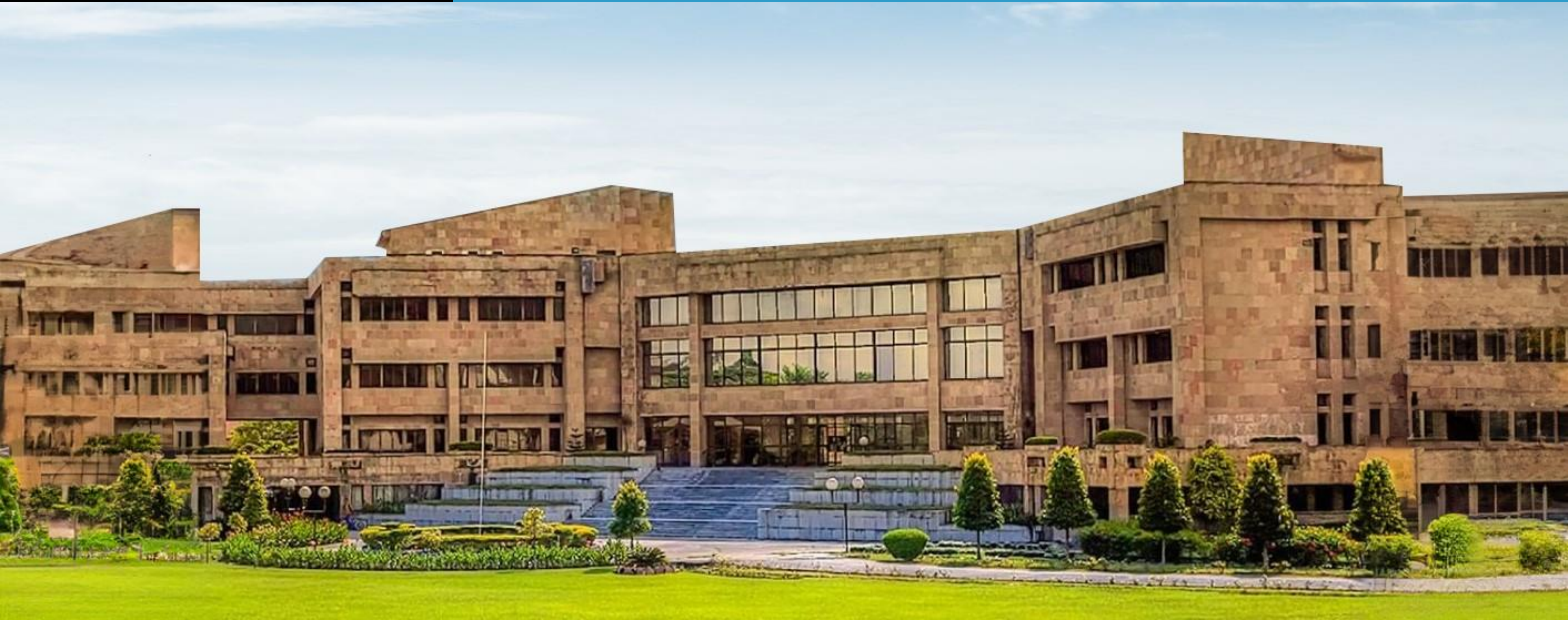
E & C I T

IIT Guwahati

Advanced Certification Programme in Data Science Business Analytics



Week 3
Data Cleaning

Topics Covered

- Introduction to Data Cleaning
- Handling Domain-Specific Data Cleaning Challenges
- Q & A

Introduction to Data Cleaning

Improve Data Quality for Better Decisions

What is data cleaning?

- Detects and corrects errors, inconsistencies and missing values in data
- Helps maintain accuracy and reliability in datasets used for analysis
- Ensures that data is structured, complete and ready for meaningful insights

Why is data cleaning important?

- Prevents inaccurate insights
- Enhances machine learning accuracy
- Strengthens decision-making

Example

- Distort customer segmentation due to missing age data

Common Data Quality Challenges

Ensuring Data Quality for Better Decision-Making

Issue	Description
Inconsistent data entry	Different spellings for the same entity
Irrelevant data	Unnecessary columns affecting analysis
Mismatched data types	Storing dates or numbers as text
Incorrect categorical encoding	Using one-hot instead of label encoding
Data leakage	Using future data in model training

Ensuring Consistency in Data Entry

Standardising Text Formatting for Accurate Analysis

Python code (standardising text data)

- Convert all text to lowercase or title case to ensure consistency across the dataset
- Remove unnecessary whitespace, characters and symbols that can cause data discrepancies

```python
import pandas as pd
df = pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['John', 'JANE', 'Mike'],
    'City': ['New York', 'new york', 'NYC']
})
df['City'] = df['City'].str.title().replace({'Nyc': 'New York'})
```

Ensuring Consistency in Data Entry

Standardising Text Formatting for Accurate Analysis

R code (standardising text data)

Use standard abbreviations uniformly to maintain clarity and prevent variations in data representation

```
library(dplyr)
df <- data.frame(
  ID = c(1, 2, 3),
  Name = c("John", "JANE", "Mike"),
  City = c("New York", "new york", "NYC")
)
df$City <- recode(df$City, "NYC" = "New York")
```

Eliminating Irrelevant Data

Streamlining Data for Better Insights

Python code

```
df.drop(columns=['Social_Security_Number'], inplace=True)
```

R code

```
df <- df %>% select(-Social_Security_Number)
```

- Remove irrelevant columns to prevent slow processing and reduce noise
- Keep only essential attributes to improve analysis efficiency
- Reduce storage needs while enhancing data accuracy
- **Example**: Removing social security numbers if they are not required

Converting Data Types

When Format Choices Go Wrong

Python code

```
df['Date'] = pd.to_datetime(df['Date'])
df['Age'] = df['Age'].astype(int)
```

- Avoid storing numerical values as text to prevent calculation errors

R code

```
df$Date <- as.Date(df$Date, format="%Y-%m-%d")
df$Age <- as.integer(df$Age)
```

- Convert date values from string format to a proper datetime format

Correcting Categorical Encoding

Improving Data Representation

Python code (one-hot encoding):

```
import pandas as pd
pd.get_dummies(df, columns=['Category'])
```

R code (one-hot encoding):

```
library(caret)
dummyVars(" ~ Category", data=df)
```

- **One-hot encoding:**
  Use one-hot encoding for categorical data without a natural ranking or order
- **Label encoding:**
  Apply label encoding when categories follow a meaningful sequence or hierarchy

Preventing Data Leakage

Maintaining Data Integrity

Python code (preventing data leakage)

```
from sklearn.model_selection import
train_test_split
train, test = train_test_split(df,
test_size=0.2, random_state=42)
```

What is data leakage?

- Occur when future data influences model training
- Lead to overfitting and unrealistic predictions

R code (preventing data leakage)

```
library(caret)
trainIndex <- createDataPartition(df$Target,
p=0.8, list=FALSE)
train <- df[trainIndex, ]
test <- df[-trainIndex, ]
```

How to avoid it?

- Exclude future values from training data
- Remove features linked to target variables

Handling Domain-Specific Data Cleaning Challenges

Handling Invalid Financial Data

Identifying and Resolving Data Issues Using Python

Python code: Handling negative financial values

```
import pandas as pd

# Creating a sample financial dataset
data = {'Employee': ['Alice', 'Bob', 'Charlie', 'David'],
        'Salary': [50000, -2000, 60000, -1500]}  #
Negative salaries

df = pd.DataFrame(data)
print("Original Data:\n", df)

df['Salary'] = df['Salary'].apply(lambda x: max(x, 0))  #
Replace negative values with 0
```

- **Issue:** Identify negative values in revenue, sales, or salary fields
- **Impact:** Lead to inaccurate analysis and flawed decisions
- **Solution:** Convert negatives to NA or set a lower bound at zero

Handling Invalid Financial Data

Identifying and Resolving Data Issues Using R

R code

```
# Creating a sample financial dataset
df <- data.frame(Employee = c("Alice", "Bob", "Charlie",
"David"),
                 Salary = c(50000, -2000, 60000, -1500))
# Negative salaries
print(df)

df$Salary[df$Salary < 0] <- NA  # Convert negative values
to NA
```

- Demonstrates how to replace negative values with NA
- Helps prevent errors in financial calculations and analysis

Healthcare Data: Standardising Medical Codes

Ensuring Consistency in ICD Codes Using Python

Python code

```
# Creating a sample healthcare dataset
data = {'Patient': ['P1', 'P2', 'P3', 'P4'],
        'ICD_Code': ['E11.90', 'E11.9', 'I10',
'E11.90']}  # Mixed ICD codes

df = pd.DataFrame(data)
print("Original Data:\n", df)

icd_mapping = {'E11.90': 'E11.9'}
df['ICD_Code'] =
df['ICD_Code'].replace(icd_mapping)
```

- **Issue:** Encounter variations in ICD codes (e.g., E11.9 vs. E11.90 for diabetes)
- **Solution:** Use a mapping dictionary to standardise codes
- Demonstrate how to standardise medical codes using a mapping dictionary

Healthcare Data: Standardising Medical Codes

Ensuring Consistency in ICD Codes Using R

R code

```r
# Creating a sample healthcare dataset
df <- data.frame(Patient = c("P1", "P2", "P3", "P4"),
                 ICD_Code = c("E11.90", "E11.9", "I10",
"E11.90"))  # Mixed ICD codes
print(df)

df$ICD_Code <- recode(df$ICD_Code, "E11.90" = "E11.9")
```

Helps standardise medical codes by applying a mapping dictionary

E-commerce Data: Currency Conversion

Standardising Different Currency Formats Using Python

Python code

```
# Creating a sample e-commerce dataset
data = {'Product': ['Laptop', 'Phone', 'Tablet',
'Headphones'],
        'Price': [1000, 800, 30000, 1500],
        'Currency': ['USD', 'EUR', 'INR', 'USD']}

df = pd.DataFrame(data)
print("Original Data:\n", df)

conversion_rates = {'EUR': 1.1, 'INR': 0.013}
df['Price_USD'] = df.apply(lambda row: row['Price'] *
conversion_rates.get(row['Currency'], 1), axis=1)
```

- **Issue:** Handle prices existing in multiple currencies (e.g., USD, EUR, INR)
- **Solution:** Convert all amounts to a common currency
- Help process currency conversion efficiently using pandas and lambda functions in Python

E-commerce Data: Currency Conversion

Standardising Different Currency Formats Using R

R code

```
# Creating a sample e-commerce dataset
df <- data.frame(Product = c("Laptop", "Phone",
"Tablet", "Headphones"),
                 Price = c(1000, 800, 30000, 1500),
                 Currency = c("USD", "EUR", "INR",
"USD"))
print(df)

df$Price_USD <- ifelse(df$Currency == "EUR", df$Price
* 1.1,
                   ifelse(df$Currency == "INR",
df$Price * 0.013, df$Price))
```

- Demonstrates currency conversion using data frames and apply functions

Cleaning Customer Reviews

Removing Unwanted Characters Using Python

Python code

```
# Creating a sample text dataset
data = {'Customer': ['John', 'Jane', 'Mike',
'Sara'],
        'Review': ['Great product! <br>', 'Loved
it 😊', 'Would buy again!', '<p>Amazing</p>']}

df = pd.DataFrame(data)
print("Original Data:\n", df)

import re
df['Review_Cleaned'] = df['Review'].apply(lambda
x: re.sub(r'<.*?>', '', x))  # Remove HTML tags
```

- **Issue:** Remove HTML tags, emojis, and special characters from reviews
- **Solution**: Use regular expressions (RegEx) to clean text

Cleaning Customer Reviews

Removing Unwanted Characters Using R

R code

```
# Creating a sample text dataset
df <- data.frame(Customer = c("John", "Jane",
"Mike", "Sara"),
                 Review = c("Great product!
<br>", "Loved it 😊", "Would buy again!",
"<p>Amazing</p>"))
print(df)

df$Review_Cleaned <- gsub("<.*?>", "",
df$Review)  # Remove HTML tags
```

- Removes unwanted elements using ‘dplyr’ and ‘stringr’
- Removes inconsistencies to ensure clean and structured customer feedback

Cleaning Invalid Timestamps in IoT Data

Ensuring Reliable Sensor Readings Using Python

Python code

```
import pandas as pd

# Creating a sample IoT sensor dataset
data = {'SensorID': [101, 102, 103, 104],
        'Temperature': [22.5, 24.0, 23.1, 25.6],
        'Timestamp': ['2021-12-30', '2023-06-15', '2025-01-01',
'2024-05-10']}  # Includes a future timestamp

df = pd.DataFrame(data)
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
print("Original Data:\n", df)

df = df[(df['Timestamp'] >= '2022-01-01') & (df['Timestamp'] <=
'2024-01-01')]
```

- **Issue:** Detect incorrect or future timestamps caused by system errors
- **Solution:** Remove timestamps outside a valid range
- Convert timestamps to datetime format and filters out invalid timestamps

Cleaning Invalid Timestamps in IoT Data

Ensuring Reliable Sensor Readings Using R

R code

```
# Creating a sample IoT sensor dataset
df <- data.frame(SensorID = c(101, 102, 103, 104),
                 Temperature = c(22.5, 24.0, 23.1, 25.6),
                 Timestamp = as.Date(c("2021-12-30", "2023-06-15", "2025-01-01", "2024-05-10")))
print(df)

df <- df %>% filter(Timestamp >= "2022-01-01" & Timestamp <= "2024-01-01")
```

- Converts timestamps into a standard date format for consistency
- Filters out invalid or incorrectly recorded timestamps to improve data quality

Survey Data: Fixing Inconsistent Responses

Standardising Open-Ended Survey Answers Using Python

Python code:

```
# Creating a sample survey dataset
data = {'Respondent': ['R1', 'R2', 'R3', 'R4'],
        'Satisfaction': ['very satisfied', 'happy',
'neutral', 'very satisfied']}  # Inconsistent
responses

df = pd.DataFrame(data)
print("Original Data:\n", df)

rating_mapping = {'very satisfied': 'Satisfied',
'happy': 'Satisfied', 'neutral': 'Neutral'}
df['Satisfaction'] =
df['Satisfaction'].replace(rating_mapping)
```

- **Issue:** Identify inconsistent wording in survey responses for similar answers
- **Solution:** Map varied responses to predefined categories
- Convert survey responses into a structured dataset
- Replace inconsistent responses with standardised values

Survey Data: Fixing Inconsistent Responses

Standardising Open-Ended Survey Answers Using R

R code:

```
# Creating a sample survey dataset
df <- data.frame(Respondent = c("R1", "R2",
"R3", "R4"),
                 Satisfaction = c("very
satisfied", "happy", "neutral", "very
satisfied"))  # Inconsistent responses
print(df)

df$Satisfaction <- recode(df$Satisfaction,
"very satisfied" = "Satisfied", "happy" =
"Satisfied")
```

- Load the survey dataset and identify inconsistencies in responses
- Use a mapping function to categorise similar responses into standard labels
- Ensure consistency in categorical data for better analysis and visualisation

Best Practices for Data Handling

Key Steps for Clean Data

1. Identify and manage inconsistent entries
2. Remove irrelevant or redundant data
3. Ensure correct data types for processing
4. Apply suitable categorical encoding techniques
5. Prevent unintended data leakage
6. Validate data integrity before analysis

Q & A

Thank you