

Data Cleaning Summary

1. Introduction

This document details the steps taken to clean the dataset, including assumptions made and methodologies used. The primary goal was to ensure the data was accurate, consistent, and ready for analysis.

2. Initial Data Inspection

- **Data Overview:** Inspected the dataset to understand the structure, column types, and identify any obvious issues such as missing values or incorrect data formats.
- **Assumptions:** Identified that out of 11,000 IDs, only 10,000 were unique, indicating the presence of rows with duplicate IDs. Additionally, observed possible null values that need further examination.

3. Handling Missing Values

Handling missing values was crucial to ensure the dataset's quality. The steps taken were:

1. Initial Inspection:

- Identified the extent and patterns of missing values across all columns.

2. Removing Rows with Excessive Missing Values:

- **Criteria:** Removed rows with more than 4 missing values.

```
df_cleaned = df.dropna(thresh=len(df.columns) - 4)
```

- **Reason:** To maintain data integrity by excluding excessively incomplete rows.

3. Filling Remaining Missing Values:

- **Name:** Filled with 'Unknown'.
- **Age:** Filled with the median age.
- **Salary:** Filled with the median salary.
- **Department:** Filled with 'Unknown'.
- **Join Date:**
 - Forward fill to fill most missing dates.
 - Backward fill to handle any remaining null values.
- **Code:**

```
df['Name'] = df['Name'].fillna('Unknown')
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Salary'] = df['Salary'].fillna(df['Salary'].median())
df['Department'] = df['Department'].fillna('Unknown')
df['Join Date'].ffill(inplace=True)
df['Join Date'].bfill(inplace=True)
```

- **Reason:** Ensured completeness and consistency in the dataset.

4. Removing Duplicates

To ensure the dataset had unique entries and to eliminate any redundancies, duplicates were removed:

1. Remove Duplicate Rows:

- **Code:**

```
df.drop_duplicates(inplace=True)
```

- **Reason:** To ensure each row represents a unique record.

2. Verification:

- **Code:**

```
df.duplicated().sum()
```

- **Reason:** To confirm that no duplicate rows remain in the dataset, ensuring data integrity.

5. Standardise Date Formats

- **Objective:** Ensure all dates follow a consistent format (YYYY-MM-DD).
- **Method:** Used `pd.to_datetime` to parse dates and convert them to the desired format.
- **Code:**

```
df_cleaned['Join Date'] = pd.to_datetime(df_cleaned['Join Date'],
errors='coerce').dt.strftime('%Y-%m-%d')
```

6. Handling Duplicate IDs

Nature of Duplicate ID Rows

- **ID Consistency:** Duplicate ID rows have identical IDs across entries.
- **Variations:** Despite identical IDs, variations may exist in other attributes such as 'Join Date', 'Name', 'Department', 'Email', and 'Salary'.
 - **Join Date:** May differ between duplicates due to updates or corrections.
 - **Name and Department:** Could include additional words or variations.
 - **Email:** Some duplicates may have incorrect formats.
 - **Salary:** Values may differ depending on changes over time or errors in recording.

Methodology

- **Sorting:** Sorted duplicate ID rows by 'Join Date' to prioritize the most recent data.
- **Name:** Kept the name with the least number of characters.
- **Email:** Kept the email with the correct format.
- **Join Date:** Always kept the latest join date.
- **Department:** Kept the department with the least number of characters.
- **Salary:** Updated the salary corresponding to the latest join date.

- **Null Values:** If a column had a null value, the non-null value from another row was retained.

Code

```
import pandas as pd
import re

# Function to validate email format
def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))

# Function to merge duplicate rows based on specified criteria
def merge_duplicates(group):
    # Sort group by Join Date to prioritize latest data
    group_sorted = group.sort_values(by='Join Date', ascending=False)

    # Start with the latest sorted row
    merged = group_sorted.iloc[0].copy()

    # Iterate through the sorted group to fill missing values and correct
    # information
    for _, row in group_sorted.iterrows():
        # Name
        if pd.notnull(row['Name']) and (pd.isnull(merged['Name']) or
len(row['Name']) < len(merged['Name'])):
            merged['Name'] = row['Name']

        # Email
        if pd.isnull(merged['Email']) or (not pd.isnull(row['Email']) and
is_valid_email(row['Email'])):
            merged['Email'] = row['Email']

        # Join Date
        merged['Join Date'] = row['Join Date']

        # Department
        if pd.isnull(merged['Department']) or (not pd.isnull(row['Department'])
and len(row['Department']) < len(merged['Department'])):
            merged['Department'] = row['Department']

        # Salary
        merged['Salary'] = row['Salary']

    return merged

# Apply merge_duplicates function on rows with duplicate IDs
cleaned_rows = []
for _, group in df.groupby('ID'):
    if group['ID'].duplicated().any():
        cleaned_row = merge_duplicates(group)
        cleaned_rows.append(cleaned_row)
    else:
        cleaned_rows.append(group.iloc[0])

df_cleaned = pd.DataFrame(cleaned_rows)
```

7. Standardizing Email Addresses

- **Objective:** Standardize email addresses in the DataFrame, replacing invalid formats with a default placeholder ('invalid@example.com').

- **Method:** Implemented a validation function (`is_valid_email`) using regular expressions to verify email formats. Applied this function to the 'Email' column using the `apply` method and lambda function.

- **Code**

```
import re

def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email)

df['Email'] = df['Email'].apply(lambda x: x if is_valid_email(x) else
                              'invalid@example.com')
```

- **Reason:** To confirm that each email address in the 'Email' column adheres to a standard format and to maintain data integrity by replacing invalid email addresses with 'invalid@example.com'.

8. Cleaning Specific Columns

Name

- **Objective:** Standardize names by removing extraneous words or characters to enhance consistency.

- **Method:** Implemented a function `clean_name` to iterate through each name:

- Removed uppercase letters that are not at the beginning or after a space.
- Trimmed trailing spaces from names.
- Processed names to handle cases:
 - If more than two words exist, removed the last word.
 - Checked if the second word was longer than 8 characters; if so, retained only the first word.

Code

```
# Function to clean names
def clean_name(name):
    # Remove uppercase letters at the end
    while name and name[-1].isupper():
        name = name[:-1]

    # Remove trailing spaces
    name = name.strip()

    # Split the cleaned name into words
    words = name.split()

    # If there are more than two words, remove the last word and return
    if len(words) > 2:
```

```

        cleaned_name = ' '.join(words[:-1])
        return cleaned_name.strip()

    # If exactly two words, check if the second word is longer than 8
    characters
    if len(words) == 2 and len(words[1]) > 8:
        cleaned_name = words[0]
    else:
        cleaned_name = name

    return cleaned_name.strip()

# apply function to Name column
df['Name'] = df['Name'].apply(clean_name)

```

Department

- **Objective:** Remove trailing characters not in the standard department names.
- **Method:** Used a function to standardize department names by stripping trailing characters.

Code

```

# Function to clean department names
def clean_department(department):
    standard_departments = ['Unknown', 'Marketing', 'Support', 'Sales',
                             'Engineering', 'HR']
    for std in standard_departments:
        if department.startswith(std):
            return std
    return department

# Apply department cleaning
df_cleaned['Department'] = df_cleaned['Department'].apply(clean_department)

```

Salary

- **Objective:** Ensure salary values are reasonable and free from random fluctuations.
- **Method:** Rounded salary values to remove decimal points. Additionally, checked for outliers, confirming no outliers were present.

Code

```

# Round salary values
df_cleaned['Salary'] = df_cleaned['Salary'].round()

```

9. Finalizing and Saving the Cleaned Data

- **Objective:** Save the sorted DataFrame to a CSV file for future use or analysis.

Code

```

# Sort the DataFrame by 'Unnamed: 0' column
df_sorted = df.sort_values(by='Unnamed: 0').reset_index(drop=True)
# Save the sorted DataFrame to a CSV file
df_sorted.to_csv('cleaned_dataset.csv', index=False)

```