**# Step 1: Load the Data**

- I loaded the dataset from the file `messy\_data.csv` into a pandas DataFrame. The initial view of the data shows a mix of date formats, missing values, and potentially incorrect salary values.

<!-- import pandas as pd

import numpy as np -->

<!-- df=pd.read\_csv("messy\_data.csv") -->

**# Step 2: Handle Missing Values in the 'Join Date' Column**

- The 'Join Date' column had missing values, and the date format was inconsistent (both `-` and `/` used).

- I used `pd.to\_datetime()` to convert the dates to a standard format and `errors='coerce'` to convert invalid entries to `NaT`.

- Missing values were filled with `'Unknown'` to ensure that all rows have a valid entry.

<!--

df['Join Date']=df['Join Date'].fillna(df['Join Date'].mode()[0])

df['Join Date'] = pd.to\_datetime(df['Join Date'], dayfirst=True, errors='coerce')

# Format as 'DD/MM/YYYY' with '/'

df['Join Date'] = df['Join Date'].dt.strftime('%d-%m-%Y') -->

**# Step 3: Convert Salary Column to Fixed Range**

- The 'Salary' column was continuous and needed to be converted into discrete ranges for easier analysis.

- I defined the following salary ranges:

    - 0 to 50k

    - 51k to 100k

    - 101k to 150k

- I used `pd.cut()` to categorize the salary values into these ranges and added a new column `Salary Range` to the DataFrame.

<!-- bins = [0, 50000, 100000, 150000]  # Salary ranges: 0-50k, 51k-100k, 101k-150k

labels = ['0-50k', '51k-100k', '101k-150k']  # Labels for each range

# Convert salary to fixed ranges using pd.cut

df['Salary'] = pd.cut(df['Salary'], bins=bins, labels=labels, right=False)

df.head(30) -->

**# step  4: Handling Missing Values in the 'Name' Field**

   -remove the rows with NaN values

   <!-- df.dropna(subset=["Name"],axis=0, inplace=True) -->

**# 5 Handling Missing Values in the 'Date' Field**

   - The **\*\*'Date'\*\*** field had missing or invalid date values.

   - I converted all valid dates to a standard format using `pd.to\_datetime()`.

   - For invalid or missing dates, I filled the `NaT` (Not a Time) entries with `'Unknown'` as a placeholder.

   - This ensures consistency in the date field.

      <!-- df['Join Date']=df['Join Date'].fillna(df['Join Date'].mode()[0])

        df['Join Date'] = pd.to\_datetime(df['Join Date'], dayfirst=True, errors='coerce')

        # Format as 'DD/MM/YYYY' with '/'

        df['Join Date'] = df['Join Date'].dt.strftime('%d-%m-%Y') -->

**# step 7 Email Format Correction**

**# Problem:**

The email addresses in the dataset had inconsistent formatting. Some emails were missing the domain (e.g., `user@domain` instead of `user@domain.com`), and others had extra spaces, typos, or incorrect characters.

**## Steps Taken:**

1. **\*\*Remove Leading and Trailing Spaces\*\***: I used the `strip()` function to remove any spaces before or after the email addresses.

2. **\*\*Validate Email Format\*\***: I used a regular expression (regex) to check for proper email format. The correct format should include:

    - A string before the `@` symbol (e.g., `user`)

    - The `@` symbol

    - A valid domain name (e.g., `domain.com`)

3. **\*\*Fix Invalid Emails\*\***: Invalid emails were replaced with `invalid\_email@domain.com` as a placeholder.

    import re

**# Strip leading/trailing spaces and convert to lowercase**

df['Email'] = df['Email'].str.strip().str.lower()

**# Define a function to check email format, including missing domain**

<!-- def validate\_email(email):

    # Regular expression for valid email format

    email\_regex = r'^[a-z0-9]+[.-\_]\*[a-z0-9]\*@[a-z0-9.-]+\.[a-z]{2,}$'

    # Check if email matches the regex

    if re.match(email\_regex, email):

        return email

    # Check if email has a missing domain (e.g., 'user@' or 'user@domain.')

    elif '.' in email and '@' not in email and not email.endswith("hotmail.com") and not email.endswith ("gmail.com"):

        username = email[:7]

        domain = email[7:]

        email = f"{username}@{domain}"

        return email

    elif not '@' in email and not re.search(r'@[a-z0-9.-]+\.[a-z]{2,}', email):

        if email.endswith("hotmail.com"):

            email= email.replace("hotmail.com","@hotmail.com")

            return email

        elif  email.endswith("gmail.com"):

          email= email.replace("gmail.com","@gmail.com")

          return email

        else:

            email=email.replace(" ","")

            return email+"@gmail.com"

    # else:

    #     return 'Invalid Email'

# Apply the email validation function -->

<!-- df['Email'] = df['Email'].apply(validate\_email) -->

**# step 8: \*\*Standardize Department Name\*\*:**

 corrected sales,engineering,marketing,support,HR, without any trailing characters.

 <!-- df.loc[df['Department'].str.startswith('Sales'), 'Department'] = 'Sales'

df.loc[df['Department'].str.startswith('Engineering'), 'Department'] = 'Engineering'

df.loc[df['Department'].str.startswith('Support'), 'Department'] = 'Support'

df.loc[df['Department'].str.startswith('HR'), 'Department'] = 'HR'

df.loc[df['Department'].str.startswith('Marketing'), 'Department'] = 'Marketing' -->

2. **\*\*Handle Other Departments\*\***:

   - For departments that do not start with "Sales," no changes were made. These department names were left intact.

3. **\*\*Ensure Consistency\*\***:

   - The dataset now has a consistent department name format for all records that start with "Sales.","Engineering","support",'Markting'.

**# Step 7 : Save the Cleaned Data**

- Once the cleaning steps were completed, I saved the cleaned data into a new file called `cleaned\_dataset.csv`.

- This file can now be used for further analysis or reporting.

<!-- df.to\_csv('cleaned\_dataset.csv', index=False) -->