**1. Text Data Cleaning Using Regular Expressions**

Regular Expressions are simple sequences of characters that one matches in a given text. Furthermore, these identified sequences can be removed or extracted from the text using a simple module called [Regex](https://www.geeksforgeeks.org/regex-regular-expression-in-c/) imported in the code as re.

Here’s a sample code for using regular expressions in :

|  |
| --- |
| import re    # Read text  text = "Read the data from <https://www.gfg.org/> posted by gfg@gmail.com"    # Remove all non-alphanumeric characters  clean\_text = re.sub(r'[^a-zA-Z0-9\s]', '', text)    # Find all email addresses in text  emails = re.findall(r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text)    # Replace all phone numbers with "PHONE NUMBER"  clean\_text = re.sub(r'\d{3}[-.\s]??\d{3}[-.\s]??\d{4}', 'PHONE NUMBER', clean\_text)    # Write cleaned text to output file  print(clean\_text)    print("Emails found:", emails) |

**Output**:

Read the data from httpswwwgfgorg posted by gfggmailcom

Emails found: ['gfg@gmail.com']

Here are some examples of the data cleaning tasks that can be done using regular expressions:

* Extract specific patterns from text data, such as email addresses, phone numbers, or dates.
* Remove unwanted characters or substrings from text data.
* Replace specific patterns or substrings with new values.
* Standardize text data by converting all characters to lowercase or uppercase.
* Identify and remove duplicates based on text data.
* Split text data into separate columns or rows based on delimiters or patterns.

**2. Read the Dataset Using Pandas**

[Pandas](https://www.geeksforgeeks.org/pandas-tutorial/) is an extremely popular, well-known, and one of the libraries used in almost all Machine Learning tasks. Pandas are essentially a package of used to deal majorly with data frames and manipulate them as per the need. While working on the data cleaning process of a data frame pandas can prove to be a very helpful library.

Below is a sample code for using Pandas in for data cleaning:

|  |
| --- |
| import pandas as pd    # Read in CSV file  df = pd.read\_csv('input.csv')    # Drop duplicates  df = df.drop\_duplicates()    # Fill missing values with mean  df = df.fillna(df.mean())    # Convert data types  df['date'] = pd.to\_datetime(df['date'])  df['amount'] = df['amount'].astype(float)    # Define custom cleaning function  def clean\_name(name):      return name.strip().title()    # Apply custom function to 'name' column  df['name'] = df['name'].apply(clean\_name)    # Write cleaned data to CSV file  df.to\_csv('output.csv', index=False) |

Here are some things you can do with Pandas to automate the data-cleaning process:

| **Task** | **Function Used** | **Description** |
| --- | --- | --- |
| Remove duplicates | drop\_duplicates() | Remove duplicate rows from a dataframe. |
| Drop missing values | dropna() | Remove rows or columns with missing values. |
| Impute missing values | fillna() | Fill in missing values in a dataframe with a specified value or method. |
| Convert data types | astype() | Convert the data type of a column in a dataframe. |
| Rename columns | rename() | Rename columns in a dataframe. |
| Group and aggregate data | groupby(), agg(), apply() | Group and aggregate data in a dataframe. |
| Filter data | query(), loc[], iloc[] | Filter data in a dataframe using various methods |
| Apply functions to data | apply() | Apply a function to a column or row in a dataframe |
| Merge data | merge(), join(), concat() | Merge data from multiple dataframes |
| Pivot data | pivot\_table() | The method allows for more advanced features such as multi-index and custom aggregation. |

By using these functions and methods, you can create a powerful data-cleaning pipeline in Pandas to automate the data-cleaning process.

**3. Mathematical Operations Using NumPy**

[NumPy](https://www.geeksforgeeks.org/python-numpy/) is another popular library in for numerical computing. As its name suggests, it stands for Numerical . It provides a powerful array data structure that can be used for efficient data processing and analysis. NumPy has several functions for cleaning data, such as filtering, sorting, and aggregating data.

Here is an example code for using NumPy to filter and sort data:

|  |
| --- |
| import numpy as np    # create a numpy array  data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])    # filter the array to keep only values greater than 5  filtered\_data = data[data > 5]    # sort the filtered data in descending order  sorted\_data = np.sort(filtered\_data)[::-1]    print(sorted\_data) |

**Output**:

[10 9 8 7 6]

Here are some things you can do with Pandas to automate the data-cleaning process:

| **Task** | **Function Used** | **Description** |
| --- | --- | --- |
| Replace missing values | numpy.nan\_to\_num() | Replaces NaN values with zeros or a specified value |
| Identify missing values | numpy.isnan() | Returns a boolean array indicating where NaN values are found |
| Replace outliers | numpy.clip() | Clips values to a specified range to remove outliers |
| Normalize data | numpy.linalg.norm() | Computes the norm (magnitude) of a vector or matrix |
| Standardize data | numpy.std(), numpy.mean() | Computes the standard deviation and mean of a dataset |
| Scale data | numpy.interp() | Scales a dataset to a specified range |
| Remove duplicate values | numpy.unique() | Removes duplicate values from an array |
| Filter data based on a condition | numpy.where() | Returns values from one array if a condition in another array is met |
| Split data into chunks | numpy.array\_split() | Splits an array into equally-sized subarrays |

**Creating a Basic Data Cleaning Pipeline in**

Now that we have discussed some of the popular libraries for automating data cleaning in , let’s dive into some of the techniques for using these libraries to clean data. Following is a structure of a basic data-cleaning pipeline that covers the most essential steps:

* **Loading the CSV file:**The CSV file is loaded as a data frame using the pandas module in .
* **Preprocessing the Data:**The data has multiple attributes and mostly these are not in a format that Machine Learning modules can understand. Hence following key preprocessing steps can be applied:
  + Removing duplicates: Duplicate rows in a dataset can cause errors or bias in analysis, so it’s important to remove them.
  + Correcting inconsistent data: Inconsistent data can arise due to errors in data entry or data integration.
  + Handling outliers: Outliers can skew analysis, so it’s important to handle them appropriately.
  + Formatting data: Data may need to be formatted to meet the requirements of the analysis.
* **Handling missing values:**Missing values can cause problems with analysis, so it’s important to handle them appropriately. Here’s an example of how to handle missing values using the pandas library in :

The above steps include some of the significant and key ones, but, as per the requirement, one can add or remove functions and clean the data using the updated pipeline.

**Implementing the Pipeline**

The final pipeline can be implemented as follows using . Here the implemented code is tested on a custom-generated dataset as well to see the effect of the data-cleaning process.

|  |
| --- |
| import pandas as pd  from sklearn.preprocessing import LabelEncoder  import numpy as np    def drop\_duplicates(df, subset\_name):      df.drop\_duplicates(subset=[subset\_name], inplace=True)      return df    def encode(df, column\_to\_encode):      le = LabelEncoder()      # fit and transform a column using the LabelEncoder      df[column\_to\_encode] = le.fit\_transform(df[column\_to\_encode])      return df    def outlier\_handling(df, column\_with\_outliers):      q1 = df[column\_with\_outliers].quantile(0.25)      q3 = df[column\_with\_outliers].quantile(0.75)      iqr = q3 - q1      # remove outliers      df = df[(df[column\_with\_outliers] > (q1 - 1.5 \* iqr))              & (df[column\_with\_outliers] < (q3 + 1.5 \* iqr))]      return df    def date\_formatting(df, column\_with\_date):      # format date column      df[column\_with\_date] = pd.to\_datetime(df[column\_with\_date],                                            format='%m/%d/%Y')      return df    def remove\_missing\_values(df):      # Find missing values      missing\_values = df.isnull().sum()      # Remove rows with missing values      df = df.dropna()      # Print number of missing values removed      print("Removed {} missing values".format(missing\_values.sum()))      return df      def data\_cleaning\_pipeline(df\_path,                             duplication\_subset,                             column\_to\_encode,                             column\_with\_outliers,                             column\_with\_date):      df = pd.read\_csv(df\_path)      df\_no\_duplicates = drop\_duplicates(df, duplication\_subset)      df\_encoded = encode(df\_no\_duplicates , column\_to\_encode)      df\_no\_outliers = outlier\_handling(df\_encoded, column\_with\_outliers)      df\_date\_formatted = date\_formatting(df\_no\_outliers, column\_with\_date)      df\_no\_nulls = remove\_missing\_values(df\_date\_formatted)      return df\_no\_nulls    # Create a sample DataFrame  data = {'Name': ['John', 'Jane', 'Bob', 'John', 'Alice'],          'Age': [30, 25, 40, 30, np.NaN],          'Gender': ['Male', 'Female', 'Male', 'Male', 'Female'],          'Income': [50000, 60000, 70000, 45000, 80000],          'Birthdate': ['01/01/1990', '02/14/1996', '03/15/1981',                        '01/01/1990', '06/30/1986'],          'Married': [True, False, True, False, True],          'Children': [2, 0, 1, 0, 3]}  df = pd.DataFrame(data)  print('Before Preprocessing:\n',df)  # Save DataFrame as CSV file  df.to\_csv('my\_data.csv', index=False)    clean\_df = data\_cleaning\_pipeline('my\_data.csv',                                    'Name',                                    'Gender',                                    'Income',                                    'Birthdate')    print('\nAfter preprocessing')  clean\_df.head() |

**Output**:

Before Preprocessing:

Name Age Gender Income Birthdate Married Children

0 John 30.0 Male 50000 01/01/1990 True 2

1 Jane 25.0 Female 60000 02/14/1996 False 0

2 Bob 40.0 Male 70000 03/15/1981 True 1

3 John 30.0 Male 45000 01/01/1990 False 0

4 Alice NaN Female 80000 06/30/1986 True 3

Removed 1 missing values

After preprocessing

Name Age Gender Income Birthdate Married Children

0 John 30.0 1 50000 1990-01-01 True 2

1 Jane 25.0 0 60000 1996-02-14 False 0

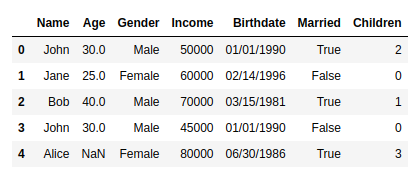
2 Bob 40.0 1 70000 1981-03-15 True 1

In the above code, a random dataset is created and then it is saved into a CSV file to pass it through the data cleaning pipeline. You can easily use the same pipeline by simply passing a different file path in the “data\_cleaning\_pipeline functions”.

Following is a snippet of the data before the cleaning process:

|  |
| --- |
| df.head() |

**Output**:

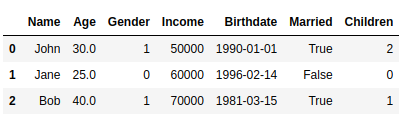


*Before Preprocessing*

 Following is a snippet of the data after the cleaning process:

|  |
| --- |
| clean\_df.head() |

**Output**:



*Cleaned dataset*