**Description about Script:**

The MBBS/BDS Extraction, Manipulation, and Closing Rank Finder Script is a Python-based tool designed to streamline the process of extracting and analyzing data from PDF files. Initially, the script extracts relevant information, such as student ranks and college details, and converts this data into a structured CSV file. This allows for easy handling and further processing of the data. By leveraging Python libraries, the script ensures that the extraction process is accurate, capturing all necessary details without omitting any critical information.

Once the data is extracted, the script performs various data manipulation tasks to correct any inconsistencies. For example, if certain values are mixed up or misplaced during extraction, the script reorganizes and sorts the data neatly. This ensures that the final dataset is clean and ready for analysis. After organizing the data, the script calculates and identifies the maximum and minimum ranks for each category, providing valuable insights into the closing ranks for MBBS and BDS admissions. This automated approach saves time and ensures precision in handling large volumes of data.

**Scripts with line by line explanation:**

**1.Script for extraction and Manipulation**

# pip install pdfplumber openpyxl: Installs the necessary Python packages: pdfplumber for extracting text from PDFs, and openpyxl for working with Excel files.

#import re: Imports the re module for working with regular expressions.

# import pdfplumber: Imports the pdfplumber module to extract text from PDF files.

# import pandas as pd: Imports the pandas library, commonly used for data manipulation and analysis.

# from collections import namedtuple: Imports namedtuple from the collections module to create a simple class-like structure for storing data.

# from openpyxl import Workbook: Imports Workbook from the openpyxl module to create and manipulate Excel files.

import re

import pdfplumber

import pandas as pd

from collections import namedtuple

from openpyxl import Workbook

# Adjusted named tuple to combine 'code', 'college', and everything else into 'code\_college'

Line = namedtuple('Line', 'sr\_no air neet\_roll\_no cet\_form\_no first\_name middle\_name last\_name gender category quota code\_college')

# Adjust the regular expressions based on your PDF structure

line\_re = re.compile(r'\d+\s+\d+\s+\d+\s+\d+\s+.+?\s+[MF]\s+.+?\s+.+?\s+\d+:.+')

file = 'example.pdf'

lines = []

with pdfplumber.open(file) as pdf:

    pages = pdf.pages

    for page in pages:

        text = page.extract\_text()

        for line in text.split('\n'):

            if line\_re.match(line):

                items = line.split()

                # Print the items and their length to inspect

                print(f"Length: {len(items)}, Items: {items}")

                # Identify the gender index to correctly capture the name

                gender\_index = 5

                for i in range(4, len(items)):

                    if items[i] in ('M', 'F'):

                        gender\_index = i

                        break

                sr\_no = items[0]

                air = items[1]

                neet\_roll\_no = items[2]

                cet\_form\_no = items[3]

                name\_parts = items[4:gender\_index]

                if len(name\_parts) == 2:

                    first\_name = name\_parts[0]

                    last\_name = name\_parts[1]

                    middle\_name = ''

                elif len(name\_parts) == 3:

                    first\_name = name\_parts[0]

                    middle\_name = name\_parts[1]

                    last\_name = name\_parts[2]

                else:

                    first\_name = name\_parts[0]

                    last\_name = ' '.join(name\_parts[1:])

                    middle\_name = ''

                gender = items[gender\_index]

                category = items[gender\_index + 1]

                quota = items[gender\_index + 2]

                code\_college = ' '.join(items[gender\_index + 3:])

                lines.append(Line(sr\_no, air, neet\_roll\_no, cet\_form\_no, first\_name, middle\_name, last\_name, gender, category, quota, code\_college))

# Create a pandas DataFrame

df = pd.DataFrame(lines)

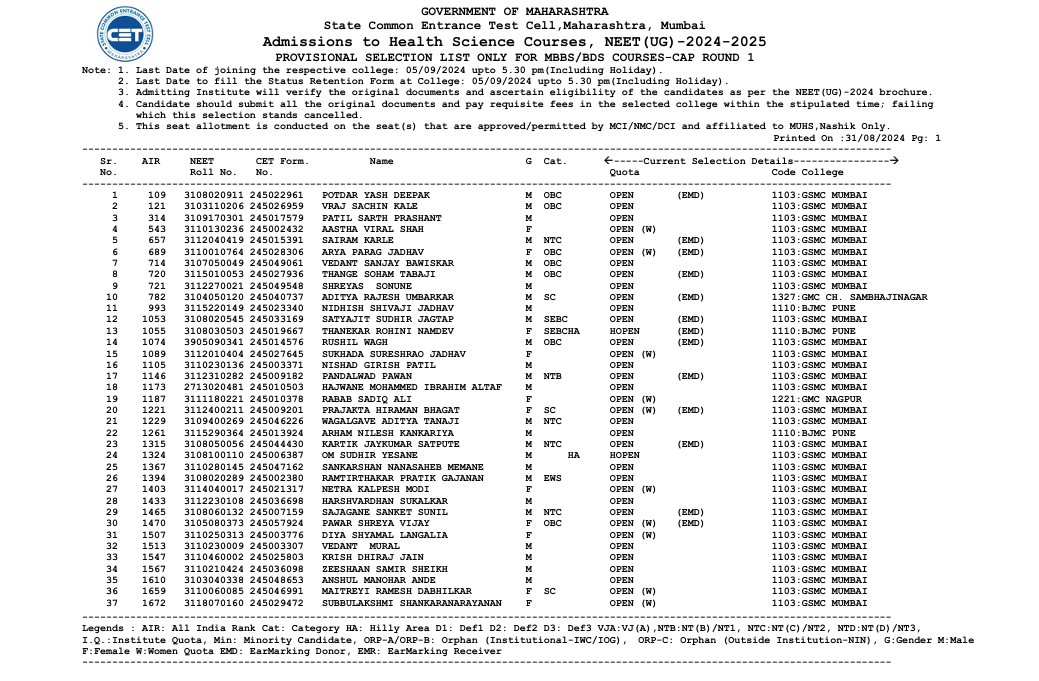
# Save the DataFrame to an Excel file

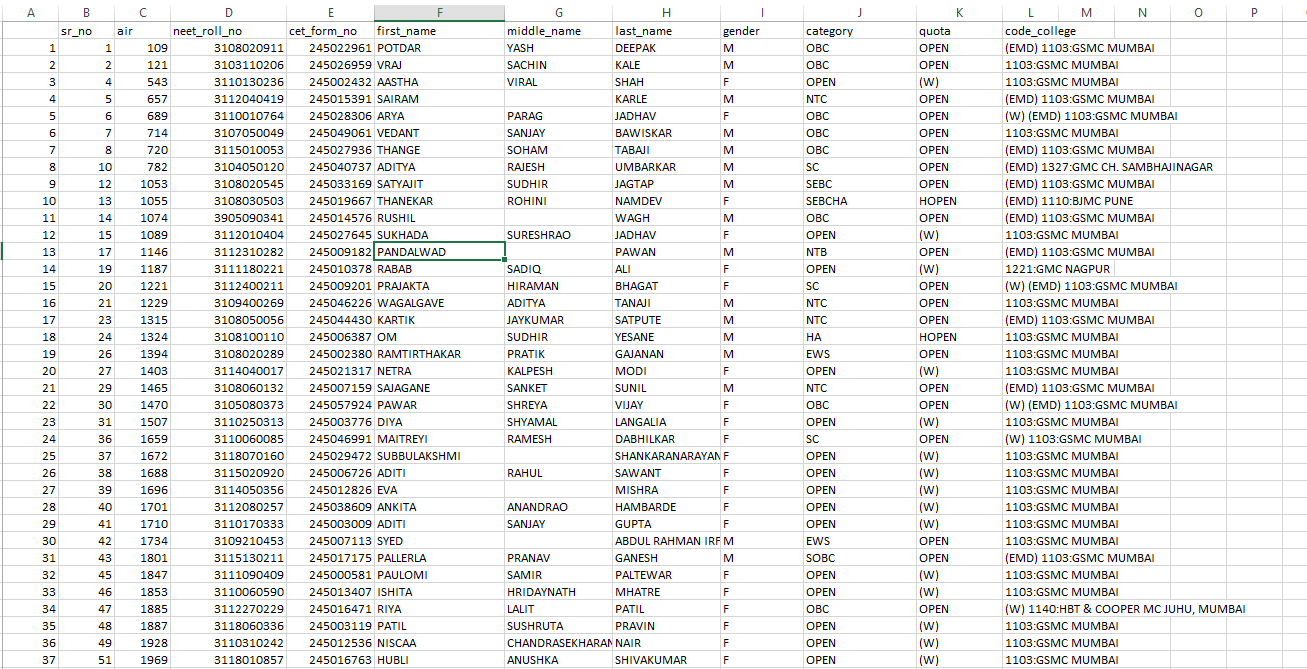
df.to\_csv('output.csv', index=False)

Above script is designed to extract specific data from a PDF file, organize it into a structured format, and then save that data as a CSV file. Here's a breakdown of its use:

1. **Data Extraction from PDF**:
   * The script uses pdfplumber to open and read the text from a PDF file named 'example.pdf'.
   * It processes each line of the PDF, applying a regular expression to match lines that contain the relevant data.
2. **Data Parsing**:
   * The script splits each matched line into individual data elements (like serial number, rank, roll number, etc.).
   * It identifies key elements like the student's name, gender, category, and college code by examining the position of these items in the line.
3. **Data Structuring**:
   * The script uses a named tuple (Line) to organize these elements into a structured format, with fields for serial number, rank, roll number, name components, gender, category, quota, and college code.
   * Each line of data from the PDF is converted into an instance of the Line named tuple and added to a list (lines).
4. **Data Conversion to DataFrame**:
   * After processing all pages of the PDF, the list of Line named tuples is converted into a pandas DataFrame (df).
   * This DataFrame provides a tabular structure that is easy to manipulate and analyze.
5. **Saving Data as CSV**:
   * Finally, the DataFrame is saved as a CSV file (output.csv), which can be used for further analysis, reporting, or integration into other systems.

**Use Case**:

* This script is particularly useful in scenarios where you need to extract structured data from a large number of PDF documents, such as processing student rank lists, admission data, or any other tabular data stored in PDFs.
* It automates the extraction and organization of data, saving time and reducing the potential for manual errors.

Output SS:

**2.Script for Sorting data neatly:**

import pandas as pd

import re

# Load the CSV file

df = pd.read\_csv('output.csv')

# Function to extract text before numbers

def extract\_text\_before\_numbers(value):

    match = re.match(r"([^\d]+)", value)

    return match.group(0).strip() if match else ''

# Apply the function to the 'cod\_college' column

df['new\_column'] = df['code\_college'].apply(extract\_text\_before\_numbers)

# Function to remove text before numbers in 'cod\_college'

def remove\_text\_before\_numbers(value):

    match = re.search(r"\d.\*", value)  # Find the first occurrence of a digit and everything after it

    return match.group(0).strip() if match else value  # Return the matched part or the original value if no match

# Apply the function to the 'cod\_college' column

df['code\_college'] = df['code\_college'].apply(remove\_text\_before\_numbers)

# Merge 'Quota' and 'new\_column' into a single column

df['Quota'] = df['category'].fillna('') + ' ' + df['quota'].fillna('') + df['new\_column'].fillna('')

# Save the updated DataFrame to a new CSV file

print(df)

# Save the updated DataFrame to a new CSV file

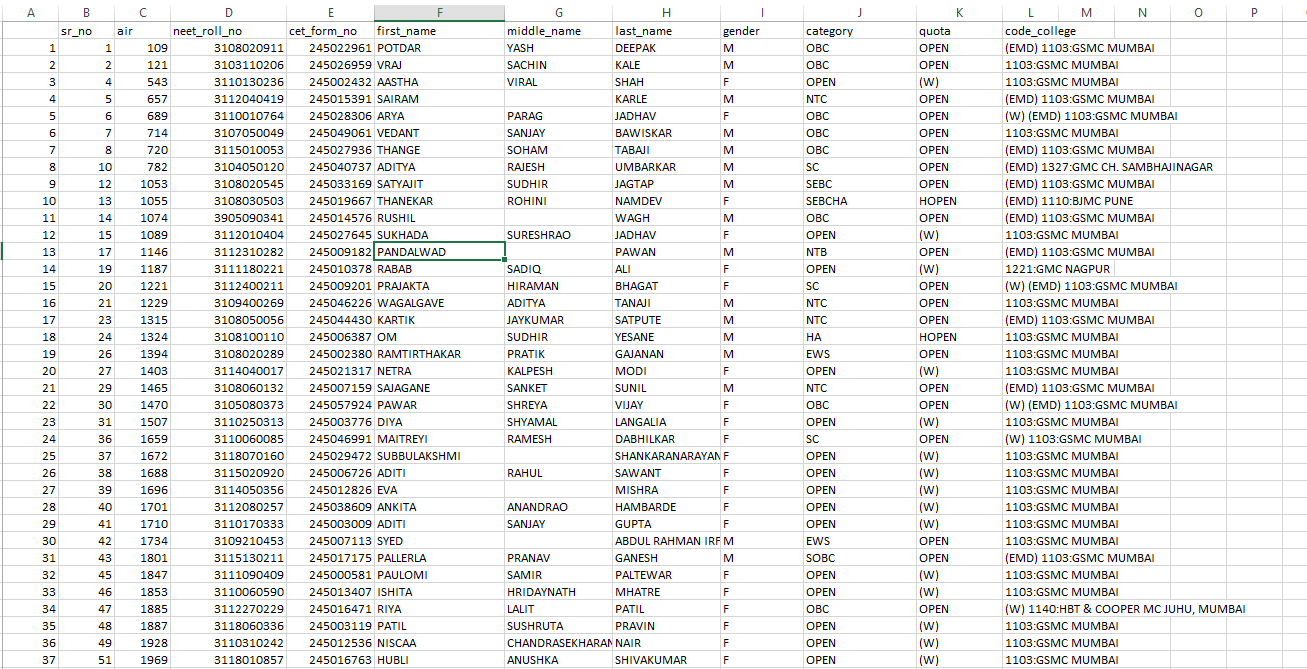
df.to\_csv('updated\_file.csv', index=False)

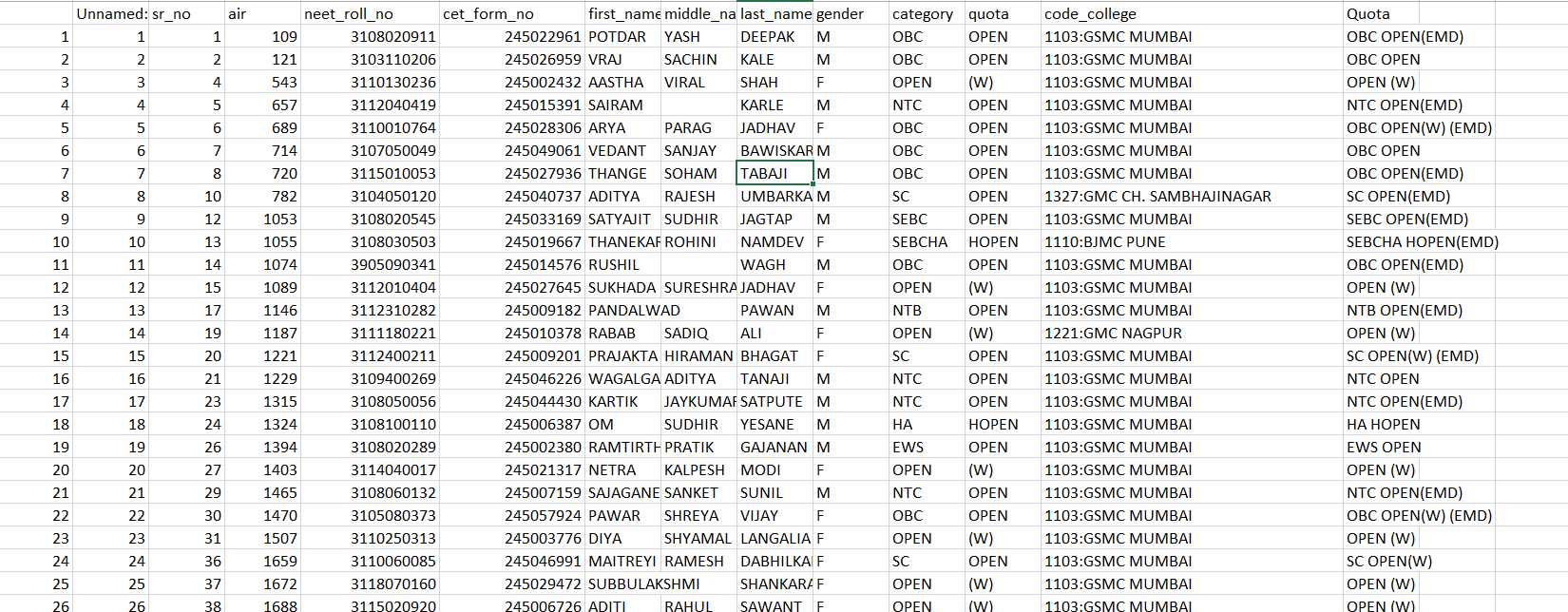
**Above** script is used to clean and manipulate data in a CSV file, specifically focusing on the code\_college column. Here's the purpose and use of each part of the script:

1. **Import necessary libraries:**
   * pandas: For data manipulation and analysis.
   * re: For working with regular expressions to handle text patterns.
2. **Load the CSV file:**
   * df = pd.read\_csv('output.csv'): Reads the data from the output.csv file into a pandas DataFrame for further processing.
3. **Function to extract text before numbers:**
   * extract\_text\_before\_numbers(value): This function takes a string and uses a regular expression to extract and return the text that appears before any digits. If no match is found, it returns an empty string.
   * df['new\_column'] = df['code\_college'].apply(extract\_text\_before\_numbers): Applies the function to the code\_college column in the DataFrame and stores the result in a new column called new\_column. This column will contain the text part of the code\_college values (i.e., everything before the numbers).
4. **Function to remove text before numbers in code\_college:**
   * remove\_text\_before\_numbers(value): This function removes the text before the first number in the code\_college values and returns the remaining string (i.e., the numeric part along with any following text). If no numbers are found, it returns the original value.
   * df['code\_college'] = df['code\_college'].apply(remove\_text\_before\_numbers): Applies the function to the code\_college column, updating it to only contain the numeric part (and any text after the number).
5. **Merge Quota and new\_column into a single column:**
   * df['Quota'] = df['category'].fillna('') + ' ' + df['quota'].fillna('') + df['new\_column'].fillna(''): Combines the values of the category, quota, and new\_column columns into a new Quota column. The fillna('') ensures that any missing values are replaced with an empty string before concatenation.
6. **Save the updated DataFrame:**
   * print(df): Prints the updated DataFrame to the console for verification.
   * df.to\_csv('updated\_file.csv', index=False): Saves the modified DataFrame to a new CSV file named updated\_file.csv, without including the index column.

**Overall Purpose:**  
This script processes the code\_college column by splitting and cleaning its content, then merges the cleaned data with other columns to create a more organized and useful structure. The result is saved in a new CSV file for further analysis or use.

**Input SS:**



**Output SS:**

**3.Script for finding Max AIR (Closing Rank) for Extracted sorted CSV file**

import pandas as pd

# Load the CSV file

df = pd.read\_csv('updated\_file.csv')

# Group by college, category, and gender to find max AIR

max\_air\_df = df.loc[df.groupby(['code\_college', 'category', 'gender', 'Quota'])['air'].idxmax()]

# Rename columns to indicate max

max\_air\_df = max\_air\_df.rename(columns={'air': 'max\_air', 'sr\_no': 'max\_sr\_no', 'neet\_roll\_no': 'max\_neet\_roll\_no', 'cet\_form\_no': 'max\_cet\_form\_no', 'Quota': 'max\_quota'})

# Select relevant columns for the final output

columns\_to\_keep = [

    'code\_college', 'category', 'gender',

    'max\_air', 'max\_quota',

]

# Display the result

final\_df = max\_air\_df[columns\_to\_keep]

# Convert to CSV

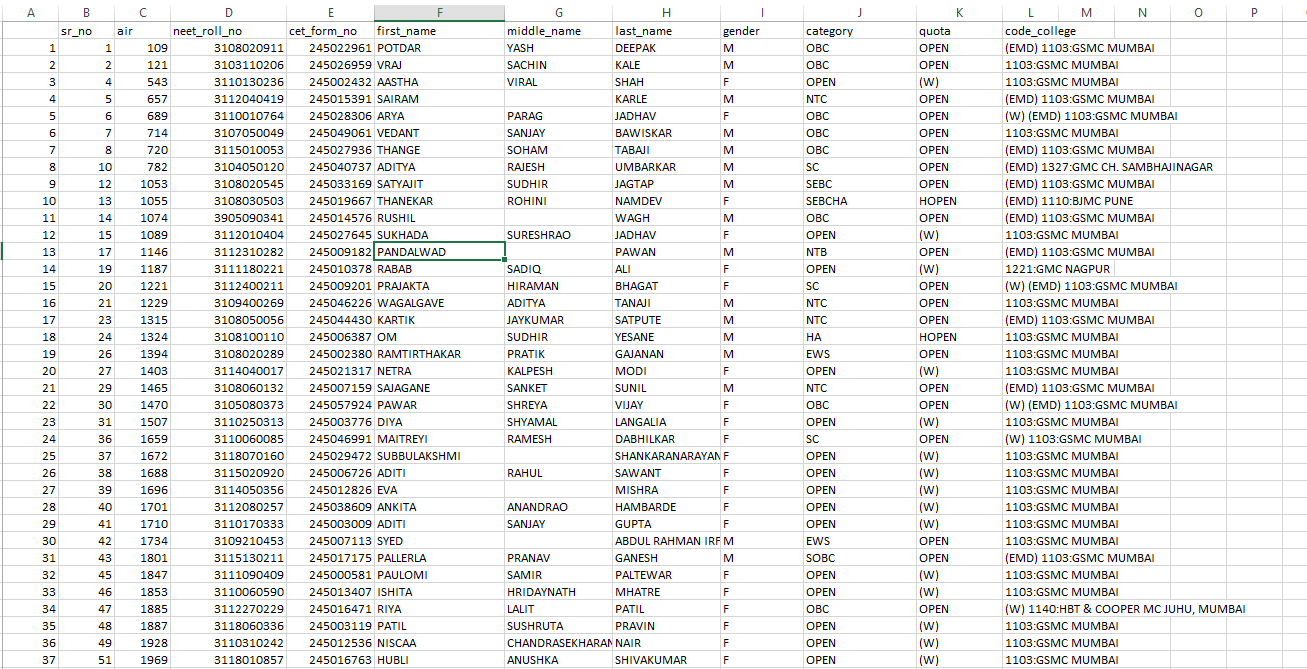
final\_df.to\_csv('output\_Max.csv', index=False)

print(final\_df)

Here's a simple explanation of what each part of the script does:

1. **Group by College, Category, and Gender to Find Maximum AIR:**, and quota. It then finds the row with the highest All India Rank (AIR) within each group and creates a new DataFrame (max\_air\_df) with these maximum AIR values.
2. **Rename Columns to Indicate Maximum Values:**This renames the columns in max\_air\_df to clearly indicate that they contain the maximum values (e.g., 'max\_air' instead of 'air').
3. **Merge Minimum and Maximum DataFrames:**This merges min\_air\_df (which presumably contains minimum AIR values) with max\_air\_df on the columns code\_college, category, gender, and Quota. The result is a DataFrame (result\_df) that contains both minimum and maximum AIR values for each group.
4. **Select Relevant Columns for Final Output:**This selects only the columns of interest from result\_df (e.g., college code, category, gender, maximum AIR, and maximum quota) and creates a new DataFrame (final\_df) with these selected columns.
5. **Convert Final DataFrame to CSV:**This saves the final\_df DataFrame to a CSV file named output\_Max.csv without including row indices.

In summary, the script processes data to find the maximum AIR for each college, category, gender, and quota combination, merges this information with minimum AIR data, selects relevant columns, and then saves the final results to a CSV file.

**Input SS:**

**Output SS:**