Data Manipulation in Python using Pandas

Creating DataFrame

```
# Importing the pandas library
import pandas as pd
# creating a dataframe object
student_register = pd.DataFrame()
# assigning values to the
# rows and columns of the dataframe
student_register['Name'] = ['Abhijit','Smriti',
                           'Akash', 'Roshni']
student_register['Age'] = [20, 19, 20, 14]
student_register['Student'] = [False, True,
                              True, False]
print(student_register)
```

Output:

```
     Name  Age  Student
0  Abhijit   20    False
1   Smriti   19     True
2    Akash   20     True
3   Roshni   14    False
```

Adding data in DataFrame using Append Function

Next, for some reason we want to add a new student in the datagram, i.e you want to add a new row to your existing data frame, that can be achieved by the following code snippet. One important concept is that the “dataframe” object of Python, consists of rows which are “series” objects instead, stack together to form a table. Hence adding a new row means creating a new series object and appending it to the dataframe.

Code:

```
# creating a new pandas
# series object
new_person = pd.Series(['Mansi', 19, True],
                      index = ['Name', 'Age',
                              'Student'])
# using the .append() function
# to add that row to the dataframe
student_register.append(new_person,
ignore_index = True)
print(student_register)
```

Output:

```
      Name  Age  Student
0  Abhijit   20    False
1   Smriti   19     True
2    Akash   20     True
3   Roshni   14    False
```

Getting Shape and information of the data

Let’s exact information of each column, i.e. what type of value it stores and how many of them are unique. There are three support functions, .shape, .info() and .corr() which output the shape of the table, information on rows and columns, and correlation between numerical columns.

Code:

```
# dimension of the dataframe
print('Shape: ')
print(student_register.shape)
print('--------------------------------------')
# showing info about the data
print('Info: ')
print(student_register.info())
print('--------------------------------------')
# correlation between columns
print('Correlation: ')
print(student_register.corr())
```

Output:

```
Shape:
(4, 3)
--------------------------------------
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Name     4 non-null      object
 1   Age      4 non-null      int64
 2   Student  4 non-null      bool
dtypes: bool(1), int64(1), object(1)
memory usage: 196.0+ bytes
None
--------------------------------------
Correlation:
              Age   Student
Age      1.000000  0.502519
Student  0.502519  1.000000
```

In the above example, the .shape function gives an output (4, 3) as that is the size of the created dataframe.

The description of the output given by .info() method is as follows:

RangeIndex describes about the index column, i.e. [0, 1, 2, 3] in our datagram. Which is the number of rows in our dataframe.

As the name suggests Data columns give the total number of columns as output.

Name, Age, Student are the name of the columns in our data, non-null tells us that in the corresponding column, there is no NA/

Nan/ None value exists. object, int64 and bool are the datatypes each column have.

dtype gives you an overview of how many data types present in the datagram, which in term simplifies the data cleaning process.

Also, in high-end machine learning models, memory usage is an important term, we can’t neglect that.

Getting Statistical Analysis of Data

Before processing and wrangling any data you need to get the total overview of it, which includes statistical conclusions like standard deviation(std), mean and it’s quartile distributions.

```
# for showing the statistical
# info of the dataframe
print('Describe')
print(student_register.describe())
```

Output:

```
Describe
            Age
count   4.000000
mean   18.250000
std     2.872281
min    14.000000
25%    17.750000
50%    19.500000
75%    20.000000
max    20.000000
```

The description of the output given by .describe() method is as follows:

count is the number of rows in the dataframe.

mean is the mean value of all the entries in the “Age” column.

std is the standard deviation of the corresponding column.

min and max are the minimum and maximum entry in the column respectively.

25%, 50% and 75% are the First Quartiles, Second Quartile(Median) and Third Quartile respectively, which gives us important info on the distribution of the dataset and makes it simpler to apply an ML model.

Dropping Columns from Data

Let’s drop a column from the data. We will use the drop function from the pandas. We will keep axis = 1 for columns.

```
students = student_register.drop('Age', axis=1)
print(students.head())
```

Output:

```
     Name  Student
0  Abhijit    False
1   Smriti     True
2   Akash      True
3   Roshni    False
```

From the output, we can see that the ‘Age’ column is dropped.

Dropping Rows from Data

Let’s try dropping a row from the dataset, for this, we will use drop function. We will keep axis=0.

```
students = students.drop(2, axis=0)
print(students.head())
```

Output:

```
     Name  Student
0  Abhijit    False
1   Smriti     True
3   Roshni    False
```

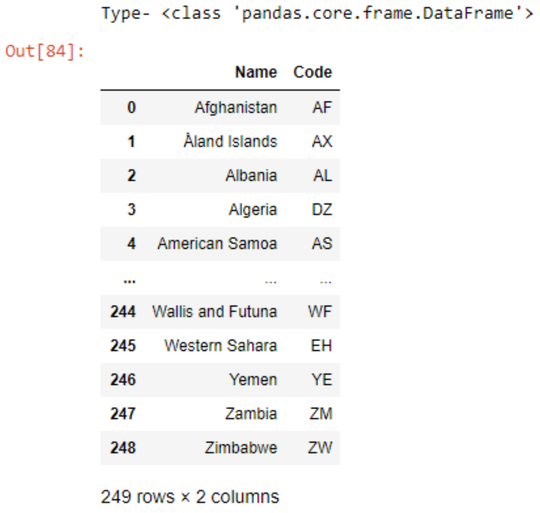
In the output we can see that the 2 row is dropped.

Below are various operations used to manipulate the dataframe:

- First, import the library which is used in data manipulation i.e. pandas then assign and read the dataframe:

```
# import module
import pandas as pd
# assign dataset
df = pd.read_csv("country_code.csv")
# display
print("Type-", type(df))
df
```

Output:



```
Type- <class 'pandas.core.frame.DataFrame'>
Out[84]:
```

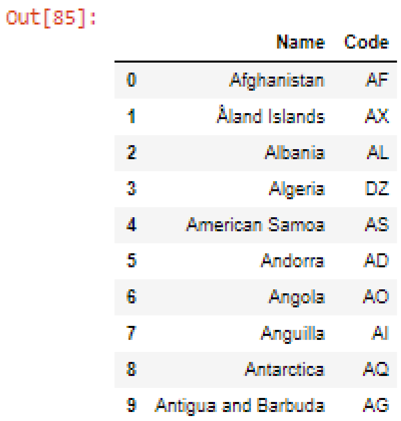
	Name	Code
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
...	...	...
244	Wallis and Futuna	WF
245	Western Sahara	EH
246	Yemen	YE
247	Zambia	ZM
248	Zimbabwe	ZW

249 rows × 2 columns

- We can read the dataframe by using **head()** function also which is having an argument (n) i.e. number of rows to be displayed.

```
df.head(10)
```

Output:



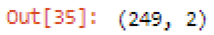
Out[85]:

	Name	Code
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
5	Andorra	AD
6	Angola	AO
7	Anguilla	AI
8	Antarctica	AQ
9	Antigua and Barbuda	AG

- Counting the rows and columns in DataFrame using **shape().** It returns the no. of rows and columns enclosed in a tuple.

df.shape

Output:

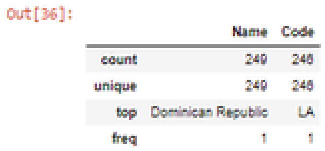


```
Out[35]: (249, 2)
```

- Summary of Statistics of DataFrame using **describe()** method.

df.describe()

Output:



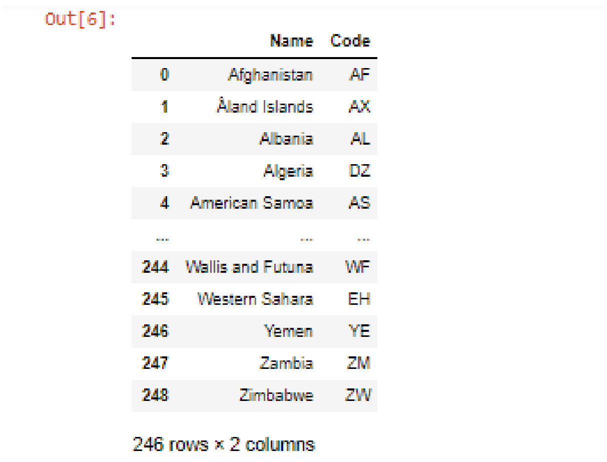
Out[36]:

	Name	Code
count	249	246
unique	249	246
top	Dominican Republic	LA
freq	1	1

- Dropping the missing values in DataFrame, it can be done using the **dropna()** method, it removes all the NaN values in the dataframe.

df.dropna()

Output:



Out[6]:

	Name	Code
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
...	...	...
244	Wallis and Futuna	WF
245	Western Sahara	EH
246	Yemen	YE
247	Zambia	ZM
248	Zimbabwe	ZW

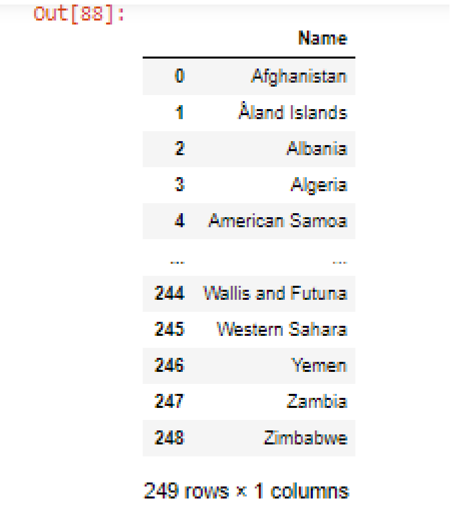
246 rows × 2 columns

Another example is:

df.dropna(axis=1)

This will drop all the columns with any missing values.

Output:



Out[88]:

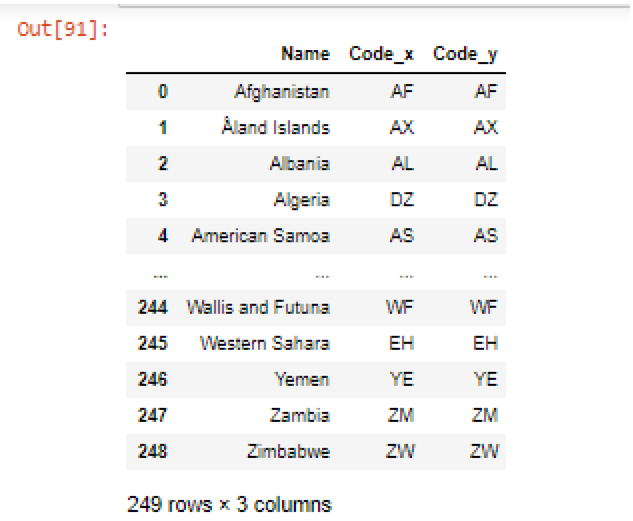
	Name
0	Afghanistan
1	Åland Islands
2	Albania
3	Algeria
4	American Samoa
...	...
244	Wallis and Futuna
245	Western Sahara
246	Yemen
247	Zambia
248	Zimbabwe

249 rows × 1 columns

- Merging DataFrames using **merge()**, arguments passed are the dataframes to be merged along with the column name.

df1 = pd.read_csv("country_code.csv")
merged_col = pd.merge(df, df1, on='Name')
merged_col

Output:



Out[91]:

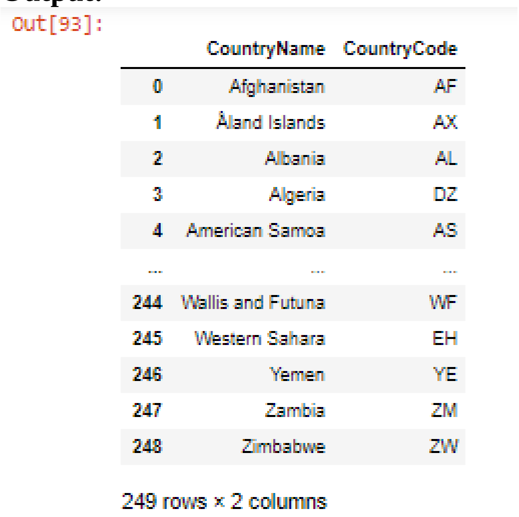
	Name	Code_x	Code_y
0	Afghanistan	AF	AF
1	Åland Islands	AX	AX
2	Albania	AL	AL
3	Algeria	DZ	DZ
4	American Samoa	AS	AS
...	...	...	...
244	Wallis and Futuna	WF	WF
245	Western Sahara	EH	EH
246	Yemen	YE	YE
247	Zambia	ZM	ZM
248	Zimbabwe	ZW	ZW

249 rows × 3 columns

- An additional argument 'on' is the name of the common column, here 'Name' is the common column given to the merge() function. df is the first dataframe & df1 is the second dataframe that is to be merged.
- Renaming the columns of dataframe using **rename()**, arguments passed are the columns to be renamed & inplace.

country_code = df.rename(columns={'Name': 'CountryName','Code': 'CountryCode'},inplace=False)
country_code

Output:



Out[93]:

	CountryName	CountryCode
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
...	...	...
244	Wallis and Futuna	WF
245	Western Sahara	EH
246	Yemen	YE
247	Zambia	ZM
248	Zimbabwe	ZW

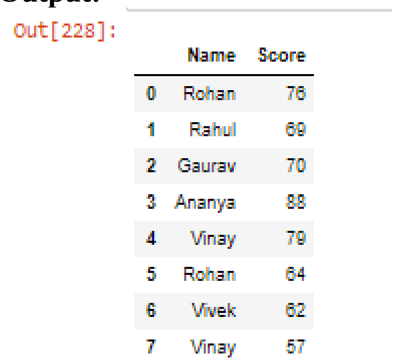
249 rows × 2 columns

The code 'inplace = False' means the result would be stored in a new DataFrame instead of the original one.

- Creating a dataframe manually:

student = pd.DataFrame({'Name': ['Rohan', 'Rahul', 'Gaurav', 'Ananya', 'Vinay', 'Rohan', 'Vivek', 'Vinay'], 'Score': [76, 69, 70, 88, 79, 64, 62, 57]})
# Reading Dataframe
student

Output:



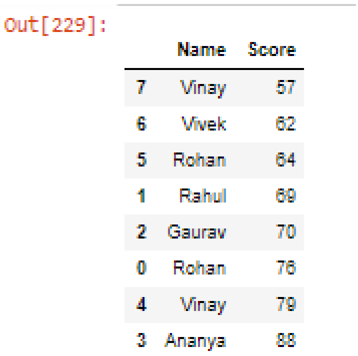
Out[228]:

	Name	Score
0	Rohan	76
1	Rahul	69
2	Gaurav	70
3	Ananya	88
4	Vinay	79
5	Rohan	64
6	Vivek	62
7	Vinay	57

- Sorting the DataFrame using **sort_values()** method.

student.sort_values(by=['Score'], ascending=True)

Output:



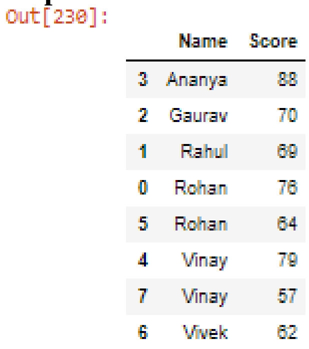
Out[229]:

	Name	Score
7	Vinay	57
6	Vivek	62
5	Rohan	64
1	Rahul	69
2	Gaurav	70
0	Rohan	76
4	Vinay	79
3	Ananya	88

- Sorting the DataFrame using multiple columns:

student.sort_values(by=['Name', 'Score'],ascending=[True, False])

Output:



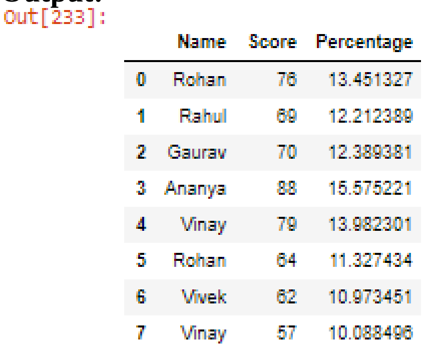
Out[230]:

	Name	Score
3	Ananya	88
2	Gaurav	70
1	Rahul	69
0	Rohan	76
5	Rohan	64
4	Vinay	79
7	Vinay	57
6	Vivek	62

- Creating another column in DataFrame, Here we will create column name percentage which will calculate the percentage of student score by using aggregate function sum().

student['Percentage'] = (student['Score'] / student['Score'].sum()) * 100
student

Output:



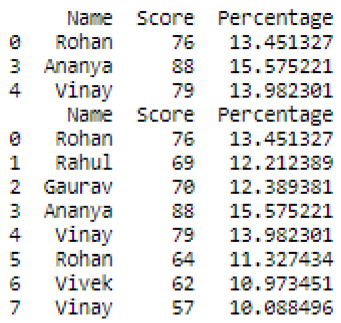
Out[233]:

	Name	Score	Percentage
0	Rohan	76	13.451327
1	Rahul	69	12.212389
2	Gaurav	70	12.389381
3	Ananya	88	15.575221
4	Vinay	79	13.982301
5	Rohan	64	11.327434
6	Vivek	62	10.973451
7	Vinay	57	10.088496

- Selecting DataFrame rows using logical operators:

# Selecting rows where score is
# greater than 70
print(student[student.Score>70])
# Selecting rows where score is greater than 60
# OR less than 70
print(student[(student.Score>60) | (student.Score<70)])

Output:



```
     Name  Score  Percentage
0   Rohan     76   13.451327
3  Ananya     88   15.575221
4   Vinay     79   13.982301
     Name  Score  Percentage
0   Rohan     76   13.451327
1   Rahul     69   12.212389
2  Gaurav     70   12.389381
3  Ananya     88   15.575221
4   Vinay     79   13.982301
5   Rohan     64   11.327434
6   Vivek     62   10.973451
7   Vinay     57   10.088496
```

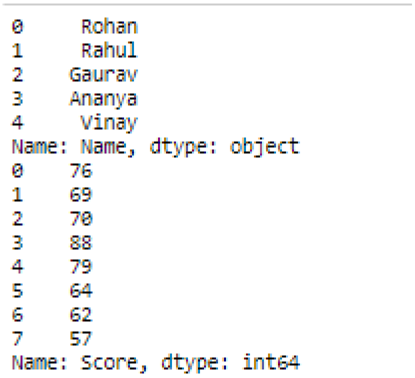
- Indexing & Slicing :

Here **.loc** is label base & **.iloc** is integer position based methods used for slicing & indexing of data.

# Printing five rows with name column only
# i.e. printing first 5 student names.
print(student.loc[0:4, 'Name'])
# Printing all the rows with score column
# only i.e. printing score of all the
# students
print(student.loc[:, 'Score'])
# Printing only first rows having name,
# score columns i.e. print first student
# name & their score.
print(student.iloc[0, 0:2])
# Printing first 3 rows having name,score &
# percentage columns i.e. printing first three
# student name,score & percentage.
print(student.iloc[0:3, 0:3])
# Printing all rows having name & score
# columns i.e. printing all student
# name & their score.
print(student.iloc[:, 0:2])

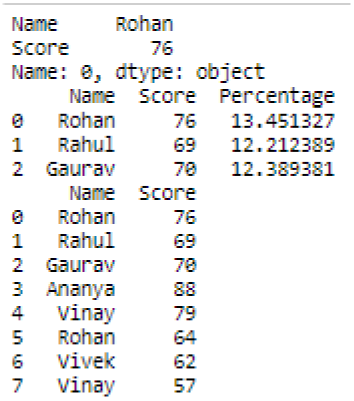
Output:

.loc:



```
0     Rohan
1     Rahul
2    Gaurav
3    Ananya
4     Vinay
Name: Name, dtype: object
0    76
1    69
2    70
3    88
4    79
5    64
6    62
7    57
Name: Score, dtype: int64
```

.iloc:

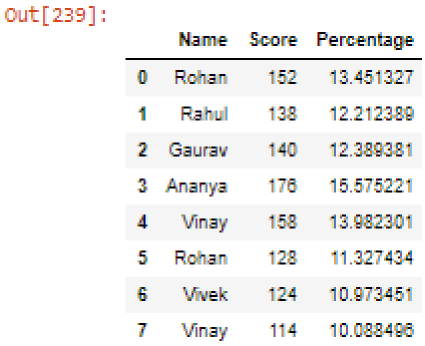


```
Name     Rohan
Score       76
Name: 0, dtype: object
     Name  Score  Percentage
0   Rohan     76   13.451327
1   Rahul     69   12.212389
2  Gaurav     70   12.389381
     Name  Score
0   Rohan     76
1   Rahul     69
2  Gaurav     70
3  Ananya     88
4   Vinay     79
5   Rohan     64
6   Vivek     62
7   Vinay     57
```

- Apply Functions, this function is used to apply a function along an axis of dataframe whether it can be row (axis=0) or column (axis=1).

# explicit function
def double(a):
    return 2*a
student['Score'] =
student['Score'].apply(double)
# Reading Dataframe
student

Output:



Out[239]:

	Name	Score	Percentage
0	Rohan	152	13.451327
1	Rahul	138	12.212389
2	Gaurav	140	12.389381
3	Ananya	176	15.575221
4	Vinay	158	13.982301
5	Rohan	128	11.327434
6	Vivek	124	10.973451
7	Vinay	114	10.088496