EXPERIMENT NO – 3

Name :Prof. Dhanashree Bhanushali

Aim – Descriptive Statistics, Function Application, Reindexing, Iteration

Theory –

Descriptive Statistics

A large number of methods collectively compute descriptive statistics and other related operations on DataFrame. Most of these are aggregations like sum(), mean(), but some of them, like sumsum(), produce an object of the same size.

- sum() – Returns the sum of the values for the requested axis. By default, axis is index (axis=0).
- mean() – Returns the average value
- std() – Returns the Bressel standard deviation of the numerical columns.

Functions and Description

Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value

abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

describe() – The describe() function computes a summary of statistics pertaining to the DataFrame columns. This function gives the mean, std and IQR values. And, function excludes the character columns and given summary about numeric columns. 'include' is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. Takes the list of values; by default, 'number'.

- object − Summarizes String columns
- number − Summarizes Numeric columns
- all − Summarizes all columns together (Should not pass it as a list value)

Function Application

To apply your own or another library's functions to Pandas objects, you should be aware of the three important methods. The appropriate method to use depends on whether your function expects to operate on an entire DataFrame, row- or column-wise, or element wise.

- Table wise Function Application: pipe()
- Row or Column Wise Function Application: apply()
- Element wise Function Application: applymap()

adder() – The adder function adds two numeric values as parameters and returns the sum.

Row or Columnwise Function Application

Arbitrary functions can be applied along the axes of a DataFrame or Panel using the apply() method, which, like the descriptive statistics methods, takes an optional axis argument. By default, the operation performs column wise, taking each column as an array-like.

Elementwise Function Application

Not all functions can be vectorized (neither the NumPy arrays which return another array nor any value), the methods applymap() on DataFrame and analogously map() on Series accept any Python function taking a single value and returning a single value.

Reindexing

Reindexing changes the row labels and column labels of a DataFrame. To reindex means to conform the data to match a given set of labels along a particular axis.

Multiple operations can be accomplished through indexing like −

- Reorder the existing data to match a new set of labels.
- Insert missing value (NA) markers in label locations where no data for the label existed.

reindex() takes an optional parameter method which is a filling method with values as follows −

- pad/ffill − Fill values forward
- bfill/backfill − Fill values backward
- nearest − Fill from the nearest index values

The rename() method allows you to relabel an axis based on some mapping (a dict or Series) or an arbitrary function. The rename() method provides an inplace named parameter, which by default is False and copies the underlying data. Pass inplace=True to rename the data in place.

Iteration

Iterating a DataFrame gives column names. The behavior of basic iteration over Pandas objects depends on the type. When iterating over a Series, it is regarded as array-like, and basic iteration produces the values. Other data structures, like DataFrame and Panel, follow the dictionary-like convention of iterating over the keys of the objects.

Basic iteration (for i in object) produces −

- Series − values
- DataFrame − column labels
- Panel − item labels

To iterate over the rows of the DataFrame, we can use the following functions −

- iteritems() − to iterate over the (key,value) pairs
- iterrows() − iterate over the rows as (index,series) pairs
- itertuples() − iterate over the rows as namedtuples

iteritems() – Iterates over each column as key, value pair with label as key and column value as a Series object.

iterrows() – iterrows() returns the iterator yielding each index value along with a series containing the data in each row.

itertuples() – itertuples() method will return an iterator yielding a named tuple for each row in the DataFrame. The first element of the tuple will be the row's corresponding index value, while the remaining values are the row values.

Code –

Descriptive Statistics

```
import pandas as pd

import numpy as np

d = {'Name' : pd.Series (['Naman', 'Shrey', 'Michaela', 'Candice', 'Sam', 'Vinay', 'Ash', 'Tom',
'James', 'Ricky']), 'Age' : pd.Series ([22, 21, 23, 25, 30, 20, 35, 18, 19, 25]), 'Salary' : pd.Series
([100000, 200000, 170000, 150000, 250000, 80000, 230000, 50000, 60000, 100000])}
#Creating a dictionary of Series

df = pd.DataFrame(d) #Creating a DataFrame

print (df)


print (df.sum()) #Returns the sum of the values for the requested axis. By default, axis is
index (axis=0).

print (df.sum(1)) #Axis=1

print (df.mean()) #Returns the average value

print (df.std()) #Returns Bressel Standard Deviation of the numerical column
```

Functions and Description

```
print (df.describe()) #Computes a summary of statistics pertaining to the DataFrame
columns (By default considers only the numerical value)

print (df.describe(include = ['object'])) #Summarises the string columns

print (df.describe(include = 'all')) #Summarises both String and Numeric columns
```

Function Application (Row/Columnwise)

```
import pandas as pd

import numpy as np

def adder (ele1, ele2) : return ele1+ele2 #Adder function
```

```
df = pd.DataFrame (np.random.randn(6,3), columns = ['col1', 'col2', 'col3'])

df.pipe (adder,2) #pipe is a table wise function application

print (df.apply(np.mean))


import pandas as pd

import numpy as np

df = pd.DataFrame (np.random.randn(6,3), columns = ['col1','col2','col3'])

df.apply(np.mean) #By default function is performed columnwise

print (df.apply(np.mean))

df = pd.DataFrame (np.random.randn(6,3), columns = ['col1','col2','col3'])

df.apply(np.mean,axis=1) #By passing axis parameter function is performed rowwise

print (df.apply(np.mean))
```

Function Application (Elementwise)

```
df = pd.DataFrame(np.random.randn(6,3), columns = ['col1','col2','col3'])

df['col1'].map(lambda x:x/500) #Custom Function

print (df.apply(np.mean))
```

Reindexing

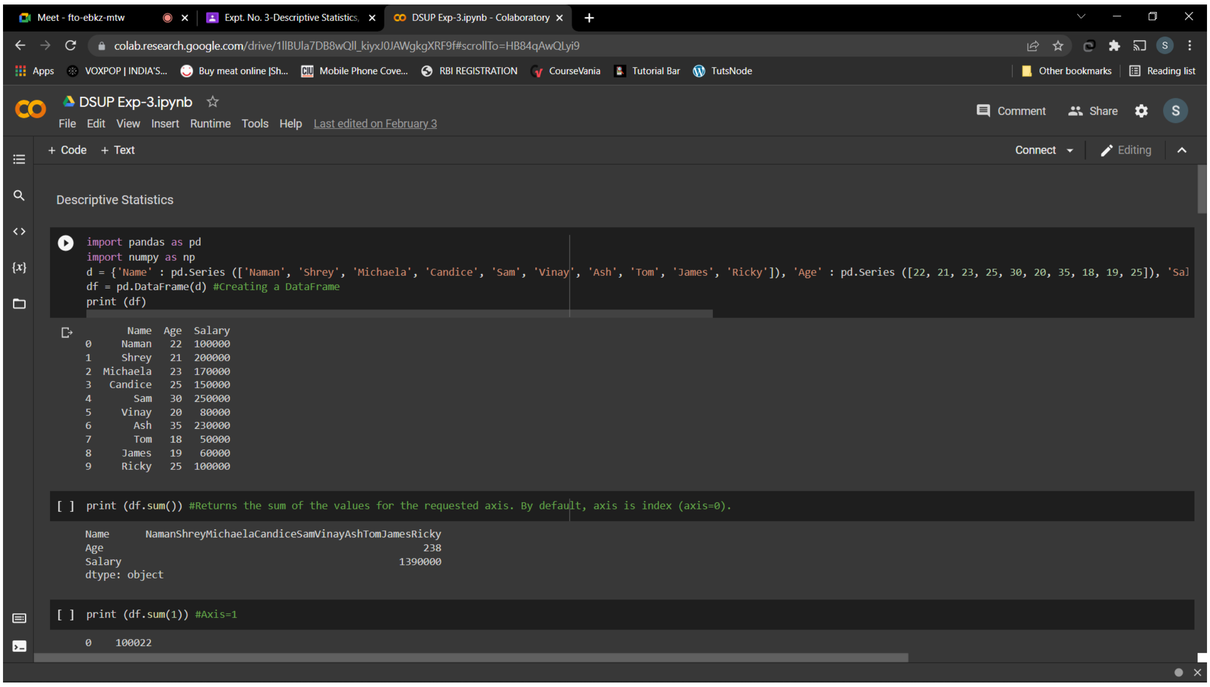
```
N=40

df = pd.DataFrame({'A': pd.date_range(start='2016-01-01',periods=N,freq='D'), 'x': np.linspace(0,stop=N-1,num=N), 'y': np.random.rand(N), 'C': np.random.choice(['Low','Medium','High'],N).tolist(), 'D': np.random.normal(100, 10, size=(N)).tolist()})

df_reind = df.reindex(index=[0,1,2], columns=['A', 'B', 'C']) #Reindexing the DataFrame

print (df_reind)

df1 = pd.DataFrame(np.random.randn(10,4),columns=['col1','col2','col3', 'col4'])

df2 = pd.DataFrame(np.random.randn(5,4),columns=['col1','col2','col3', 'col4'])

df1 = df1.reindex_like(df2)

print (df1)
```

```
df1 = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])

df2 = pd.DataFrame(np.random.randn(3,3),columns=['col1','col2','col3'])

print (df2.reindex_like(df1)) #Padding NAN's

# Now Fill the NAN's with preceding Values

print ("Data Frame with Forward Fill :")

print (df2.reindex_like(df1,method = 'ffill'))

print ("Data Frame with Backward Fill :")

print (df2.reindex_like(df1,method = 'bfill'))


df1 = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])

print (df1)

print ("After renaming the rows and columns :")

print (df1.rename(columns={'col1' : 'c1', 'col2' : 'c2'}, index = {0 : 'X', 1 : 'Y', 2 : 'Z'})) #Renaming col1 and col2 and renaming rows 1,2,3
```

Iteration

```
import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])

for key,value in df.iteritems() : print (key,value) #Iterating over key,value pairs

for row_index,row in df.iterrows() : print (row_index,row) #Iterating over rows as index,series pairs

for row in df.itertuples() : print (row) #Iterating over rows as named tuples
```

Output –



Descriptive Statistics

```
import pandas as pd
import numpy as np
d = {'Name' : pd.Series (['Naman', 'Shrey', 'Michaela', 'Candice', 'Sam', 'Vinay', 'Ash', 'Tom', 'James', 'Ricky']), 'Age' : pd.Series ([22, 21, 23, 25, 30, 20, 35, 18, 19, 25]), 'Sal
df = pd.DataFrame(d) #Creating a DataFrame
print (df)
```

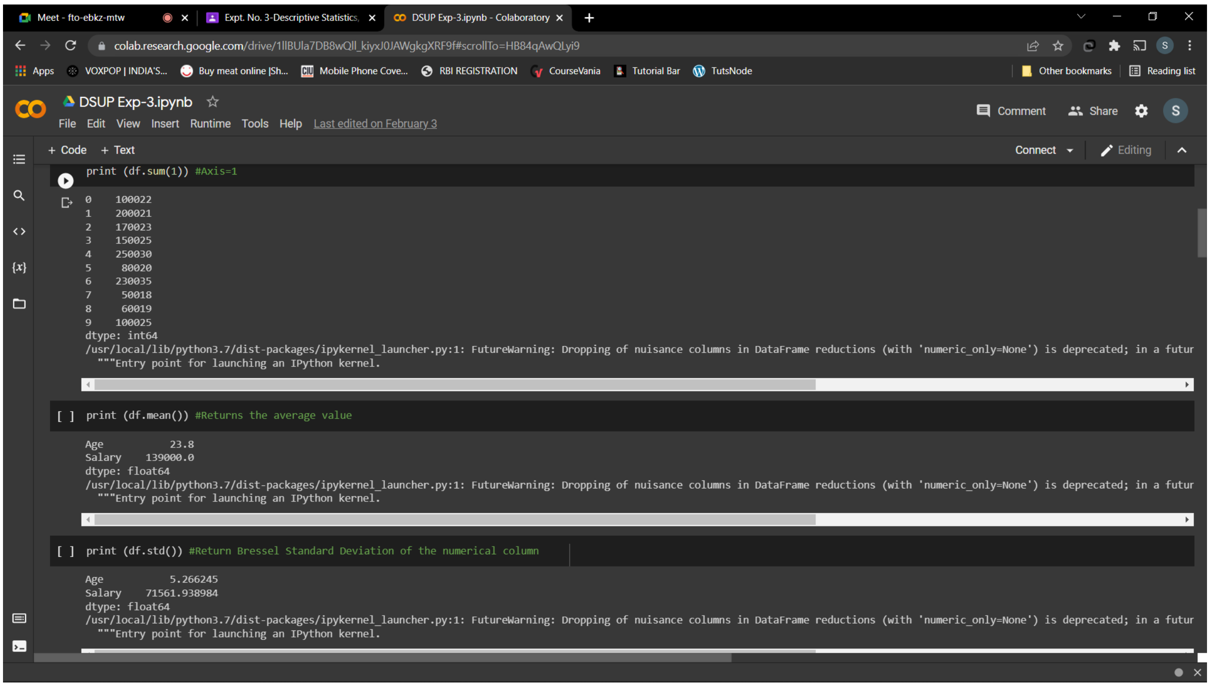
```
       Name  Age  Salary
0     Naman   22  100000
1     Shrey   21  200000
2  Michaela   23  170000
3   Candice   25  150000
4       Sam   30  250000
5     Vinay   20   80000
6       Ash   35  230000
7       Tom   18   50000
8     James   19   60000
9     Ricky   25  100000
```

```
[ ] print (df.sum()) #Returns the sum of the values for the requested axis. By default, axis is index (axis=0).
```

```
Name      NamanShreyMichaelaCandiceSamVinayAshTomJamesRicky
Age                                                     238
Salary                                              1390000
dtype: object
```

```
[ ] print (df.sum(1)) #Axis=1
```

```
0    100022
```



```
print (df.sum(1)) #Axis=1
```

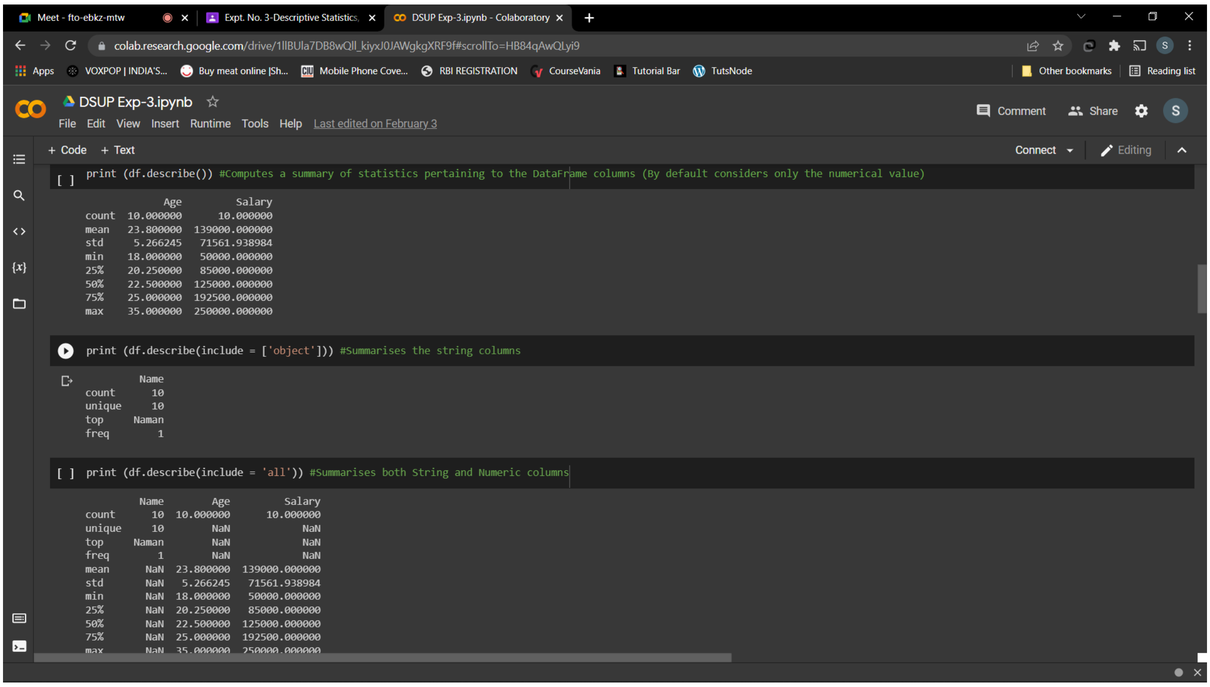
```
0    100022
1    200021
2    170023
3    150025
4    250030
5     80020
6    230035
7     50018
8     60019
9    100025
dtype: int64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a futur
  """Entry point for launching an IPython kernel.
```

```
[ ] print (df.mean()) #Returns the average value
```

```
Age           23.8
Salary    139000.0
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a futur
  """Entry point for launching an IPython kernel.
```

```
[ ] print (df.std()) #Return Bressel Standard Deviation of the numerical column
```

```
Age           5.266245
Salary    71561.938984
dtype: float64
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a futur
  """Entry point for launching an IPython kernel.
```



```
print (df.describe()) #Computes a summary of statistics pertaining to the DataFrame columns (By default considers only the numerical value)
```

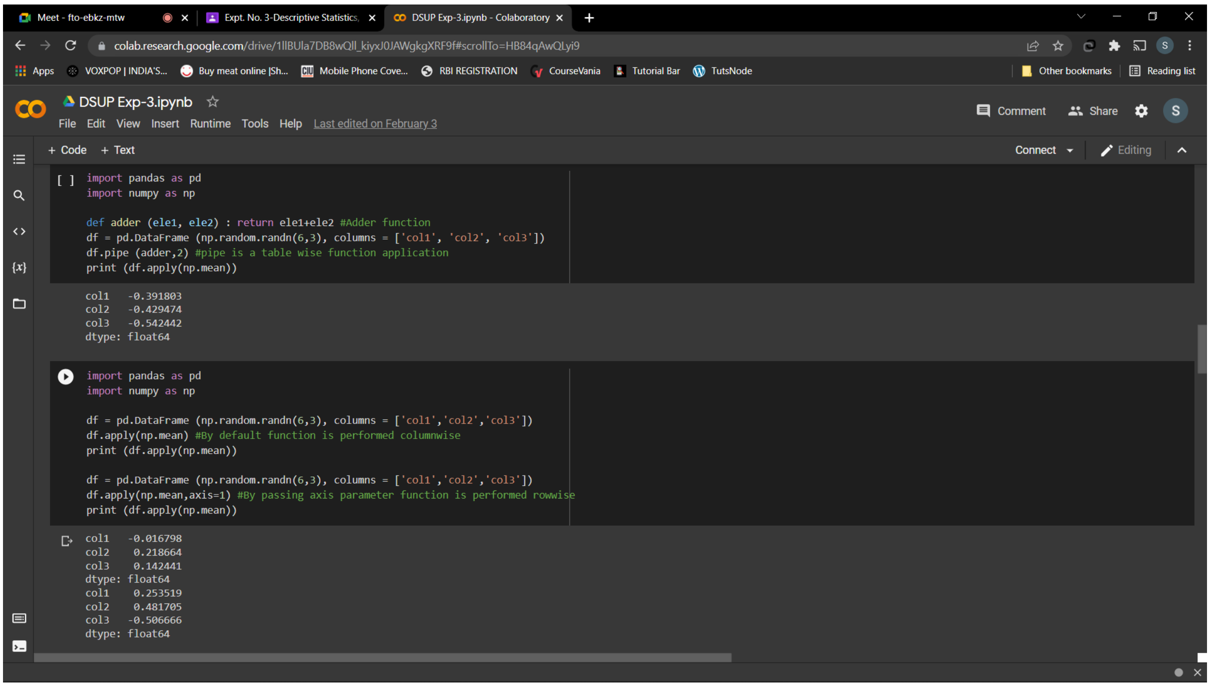
```
             Age         Salary
count  10.000000      10.000000
mean   23.800000  139000.000000
std     5.266245   71561.938984
min    18.000000   50000.000000
25%    20.250000   85000.000000
50%    22.500000  125000.000000
75%    25.000000  192500.000000
max    35.000000  250000.000000
```

```
print (df.describe(include = ['object'])) #Summarises the string columns
```

```
         Name
count      10
unique     10
top     Naman
freq        1
```

```
print (df.describe(include = 'all')) #Summarises both String and Numeric columns
```

```
         Name        Age         Salary
count      10  10.000000      10.000000
unique     10        NaN            NaN
top     Naman        NaN            NaN
freq        1        NaN            NaN
mean      NaN  23.800000  139000.000000
std       NaN   5.266245   71561.938984
min       NaN  18.000000   50000.000000
25%       NaN  20.250000   85000.000000
50%       NaN  22.500000  125000.000000
75%       NaN  25.000000  192500.000000
max       NaN  35.000000  250000.000000
```

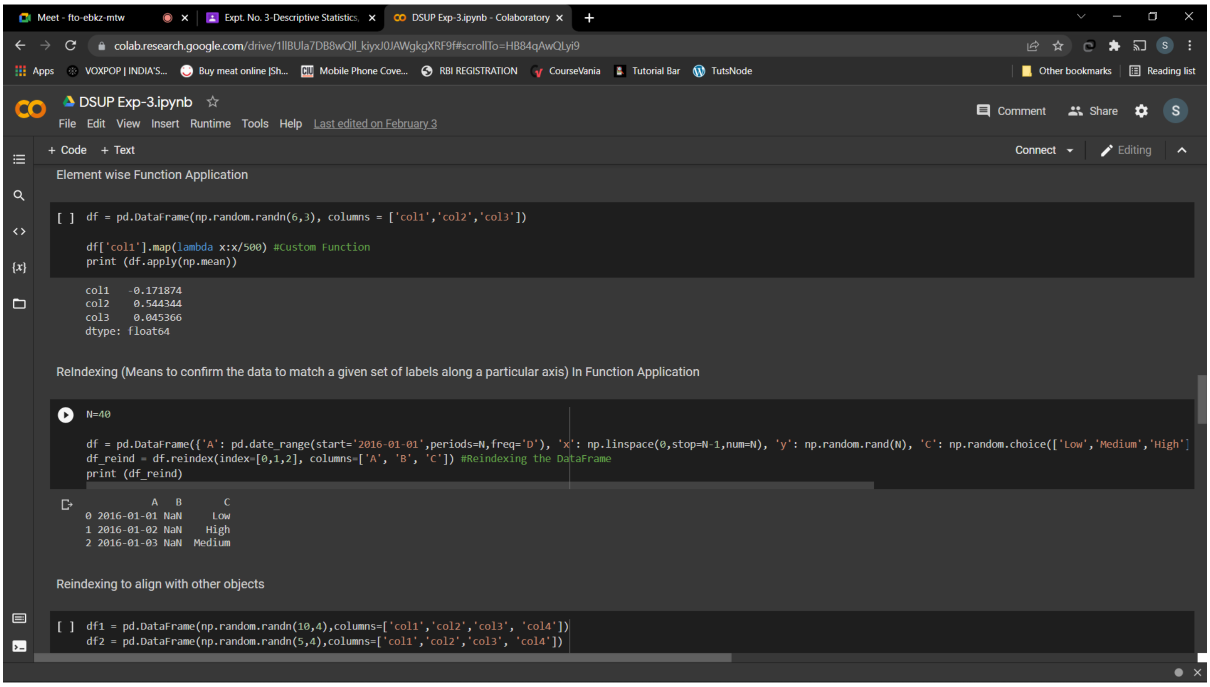


```
import pandas as pd
import numpy as np

def adder (ele1, ele2) : return ele1+ele2 #Adder function
df = pd.DataFrame (np.random.randn(6,3), columns = ['col1', 'col2', 'col3'])
df.pipe (adder,2) #pipe is a table wise function application
print (df.apply(np.mean))
```

```
col1   -0.391803
col2   -0.429474
col3   -0.542442
dtype: float64
```

```
import pandas as pd
import numpy as np

df = pd.DataFrame (np.random.randn(6,3), columns = ['col1','col2','col3'])
df.apply(np.mean) #By default function is performed columnwise
print (df.apply(np.mean))

df = pd.DataFrame (np.random.randn(6,3), columns = ['col1','col2','col3'])
df.apply(np.mean,axis=1) #By passing axis parameter function is performed rowwise
print (df.apply(np.mean))
```

```
col1   -0.016798
col2    0.218664
col3    0.142441
dtype: float64
col1    0.253519
col2    0.481705
col3   -0.506666
dtype: float64
```



Element wise Function Application

```
[ ] df = pd.DataFrame(np.random.randn(6,3), columns = ['col1','col2','col3'])

    df['col1'].map(lambda x:x/500) #Custom Function
    print (df.apply(np.mean))
```

```
col1   -0.171874
col2    0.544344
col3    0.045366
dtype: float64
```

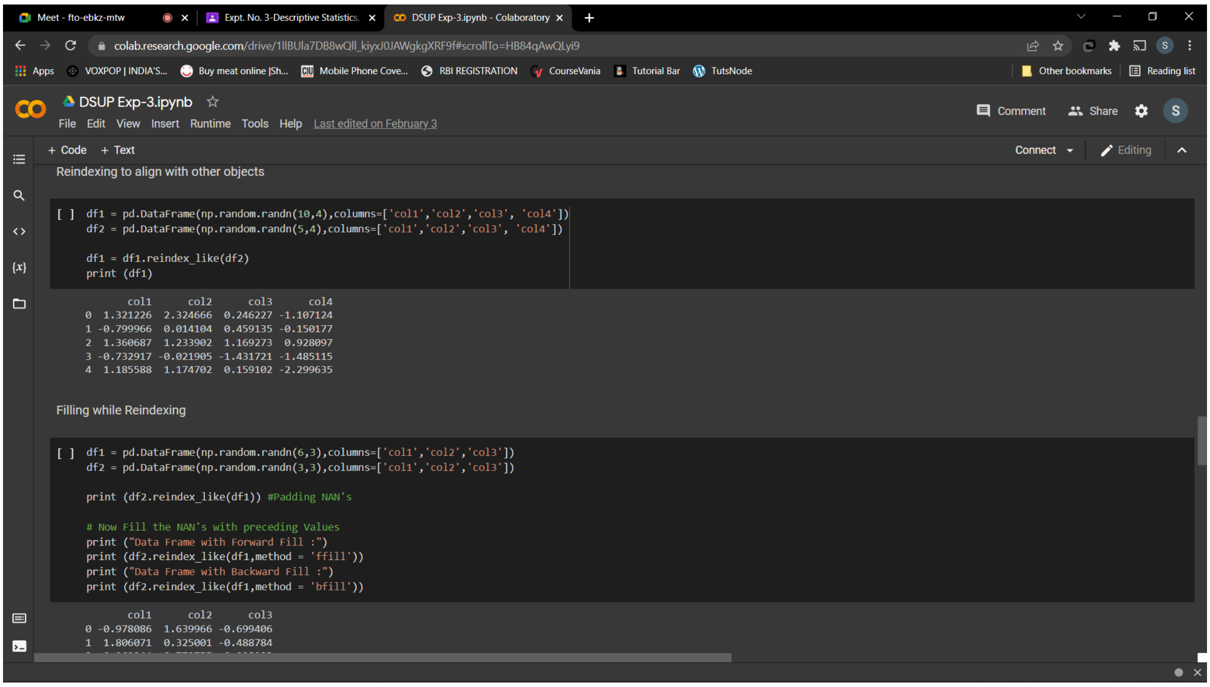
ReIndexing (Means to confirm the data to match a given set of labels along a particular axis) In Function Application

```
N=40

df = pd.DataFrame({'A': pd.date_range(start='2016-01-01',periods=N,freq='D'), 'x': np.linspace(0,stop=N-1,num=N), 'y': np.random.rand(N), 'C': np.random.choice(['Low','Medium','High']
df_reind = df.reindex(index=[0,1,2], columns=['A', 'B', 'C']) #Reindexing the DataFrame
print (df_reind)
```

```
           A   B       C
0 2016-01-01 NaN     Low
1 2016-01-02 NaN    High
2 2016-01-03 NaN  Medium
```

Reindexing to align with other objects

```
[ ] df1 = pd.DataFrame(np.random.randn(10,4),columns=['col1','col2','col3', 'col4'])
    df2 = pd.DataFrame(np.random.randn(5,4),columns=['col1','col2','col3', 'col4'])
```



Reindexing to align with other objects

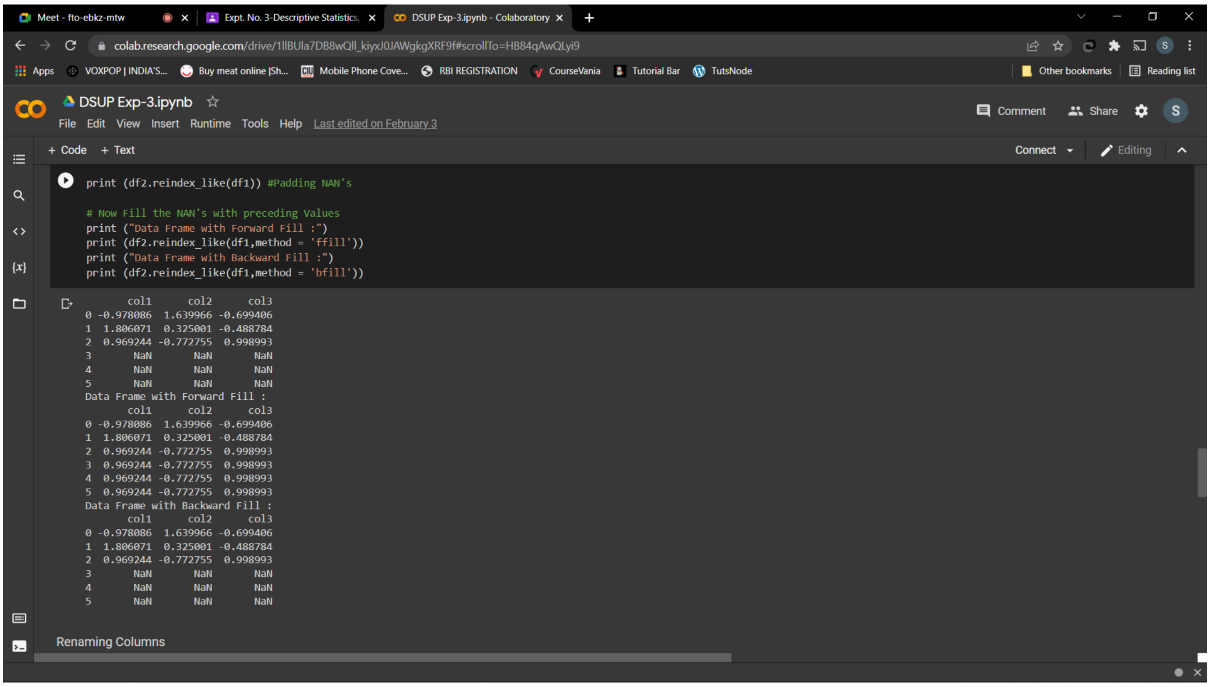
```
[ ] df1 = pd.DataFrame(np.random.randn(10,4),columns=['col1','col2','col3', 'col4'])
    df2 = pd.DataFrame(np.random.randn(5,4),columns=['col1','col2','col3', 'col4'])

    df1 = df1.reindex_like(df2)
    print (df1)
```

```
       col1      col2      col3      col4
0  1.321226  2.324666  0.246227 -1.107124
1 -0.799966  0.014104  0.459135 -0.150177
2  1.360687  1.233902  1.169273  0.928097
3 -0.732917 -0.021905 -1.431721 -1.485115
4  1.185588  1.174702  0.159102 -2.299635
```

Filling while Reindexing

```
[ ] df1 = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])
    df2 = pd.DataFrame(np.random.randn(3,3),columns=['col1','col2','col3'])

    print (df2.reindex_like(df1)) #Padding NAN's

    # Now Fill the NAN's with preceding Values
    print ("Data Frame with Forward Fill :")
    print (df2.reindex_like(df1,method = 'ffill'))
    print ("Data Frame with Backward Fill :")
    print (df2.reindex_like(df1,method = 'bfill'))
```

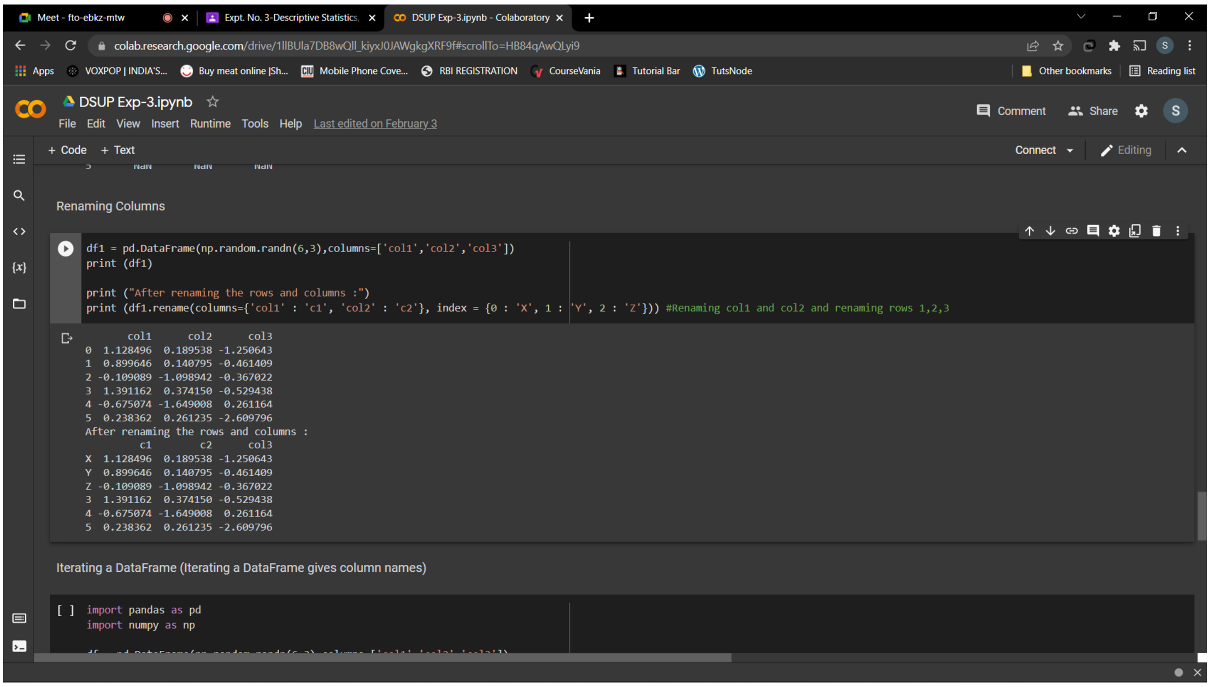
```
       col1      col2      col3
0 -0.978086  1.639966 -0.699406
1  1.806071  0.325001 -0.488784
```


Meet - fto-ebkz-mtw
Expt. No. 3-Descriptive Statistics.
DSUP Exp-3.ipynb - Colaboratory
colab.research.google.com/drive/1llBUla7DB8wQll_kiyxJ0JAWgkgXRF9f#scrollTo=HB84qAwQLyi9
DSUP Exp-3.ipynb
File Edit View Insert Runtime Tools Help Last edited on February 3
Comment Share
+ Code + Text
Connect Editing

```
print (df2.reindex_like(df1)) #Padding NAN's

# Now Fill the NAN's with preceding Values
print ("Data Frame with Forward Fill :")
print (df2.reindex_like(df1,method = 'ffill'))
print ("Data Frame with Backward Fill :")
print (df2.reindex_like(df1,method = 'bfill'))
```

```
       col1      col2      col3
0 -0.978086  1.639966 -0.699406
1  1.806071  0.325001 -0.488784
2  0.969244 -0.772755  0.998993
3       NaN       NaN       NaN
4       NaN       NaN       NaN
5       NaN       NaN       NaN
Data Frame with Forward Fill :
       col1      col2      col3
0 -0.978086  1.639966 -0.699406
1  1.806071  0.325001 -0.488784
2  0.969244 -0.772755  0.998993
3  0.969244 -0.772755  0.998993
4  0.969244 -0.772755  0.998993
5  0.969244 -0.772755  0.998993
Data Frame with Backward Fill :
       col1      col2      col3
0 -0.978086  1.639966 -0.699406
1  1.806071  0.325001 -0.488784
2  0.969244 -0.772755  0.998993
3       NaN       NaN       NaN
4       NaN       NaN       NaN
5       NaN       NaN       NaN
```

Renaming Columns


Meet - fto-ebkz-mtw
Expt. No. 3-Descriptive Statistics.
DSUP Exp-3.ipynb - Colaboratory
colab.research.google.com/drive/1llBUla7DB8wQll_kiyxJ0JAWgkgXRF9f#scrollTo=HB84qAwQLyi9
DSUP Exp-3.ipynb
File Edit View Insert Runtime Tools Help Last edited on February 3
Comment Share
+ Code + Text
Connect Editing

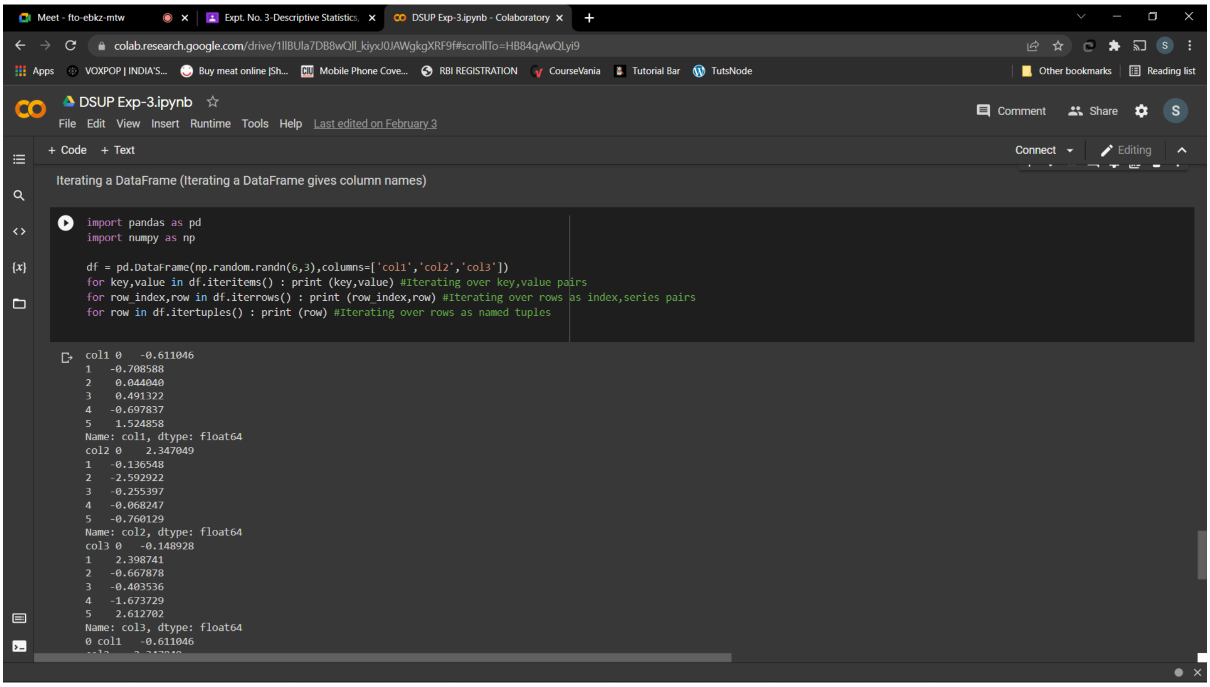
Renaming Columns

```
df1 = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])
print (df1)

print ("After renaming the rows and columns :")
print (df1.rename(columns={'col1' : 'c1', 'col2' : 'c2'}, index = {0 : 'X', 1 : 'Y', 2 : 'Z'})) #Renaming col1 and col2 and renaming rows 1,2,3
```

```
       col1      col2      col3
0  1.128496  0.189538 -1.250643
1  0.899646  0.140795 -0.461409
2 -0.109089 -1.098942 -0.367022
3  1.391162  0.374150 -0.529438
4 -0.675074 -1.649008  0.261164
5  0.238362  0.261235 -2.609796
After renaming the rows and columns :
         c1        c2      col3
X  1.128496  0.189538 -1.250643
Y  0.899646  0.140795 -0.461409
Z -0.109089 -1.098942 -0.367022
3  1.391162  0.374150 -0.529438
4 -0.675074 -1.649008  0.261164
5  0.238362  0.261235 -2.609796
```

Iterating a DataFrame (Iterating a DataFrame gives column names)

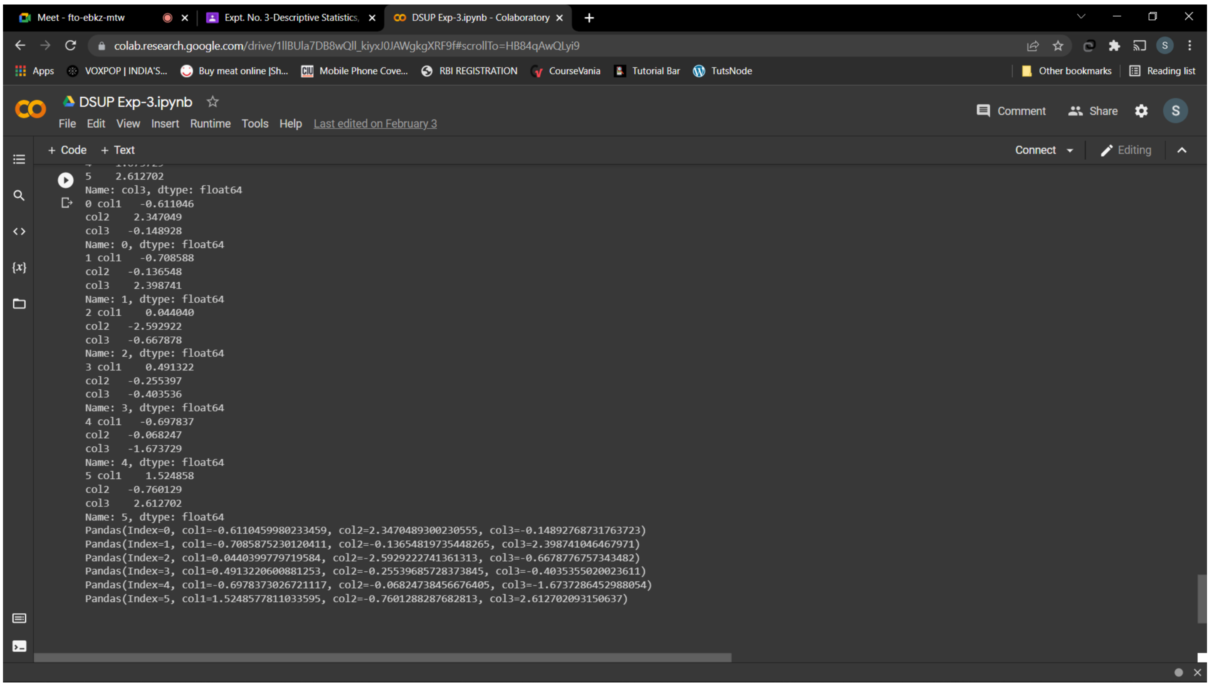
```
import pandas as pd
import numpy as np
```


DSUP Exp-3.ipynb
File Edit View Insert Runtime Tools Help Last edited on February 3
+ Code + Text
Comment Share
Connect Editing

Iterating a DataFrame (Iterating a DataFrame gives column names)

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6,3),columns=['col1','col2','col3'])
for key,value in df.iteritems() : print (key,value) #Iterating over key,value pairs
for row_index,row in df.iterrows() : print (row_index,row) #Iterating over rows as index,series pairs
for row in df.itertuples() : print (row) #Iterating over rows as named tuples
```

```
col1 0   -0.611046
1   -0.708588
2    0.044040
3    0.491322
4   -0.697837
5    1.524858
Name: col1, dtype: float64
col2 0    2.347049
1   -0.136548
2   -2.592922
3   -0.255397
4   -0.068247
5   -0.760129
Name: col2, dtype: float64
col3 0   -0.148928
1    2.398741
2   -0.667878
3   -0.403536
4   -1.673729
5    2.612702
Name: col3, dtype: float64
0 col1   -0.611046
```


DSUP Exp-3.ipynb
File Edit View Insert Runtime Tools Help Last edited on February 3
+ Code + Text
Comment Share
Connect Editing

```
5    2.612702
Name: col3, dtype: float64
0 col1   -0.611046
col2    2.347049
col3   -0.148928
Name: 0, dtype: float64
1 col1   -0.708588
col2   -0.136548
col3    2.398741
Name: 1, dtype: float64
2 col1    0.044040
col2   -2.592922
col3   -0.667878
Name: 2, dtype: float64
3 col1    0.491322
col2   -0.255397
col3   -0.403536
Name: 3, dtype: float64
4 col1   -0.697837
col2   -0.068247
col3   -1.673729
Name: 4, dtype: float64
5 col1    1.524858
col2   -0.760129
col3    2.612702
Name: 5, dtype: float64
Pandas(Index=0, col1=-0.6110459980233459, col2=2.3470489300230555, col3=-0.14892768731763723)
Pandas(Index=1, col1=-0.7085875230120411, col2=-0.13654819735448265, col3=2.398741046467971)
Pandas(Index=2, col1=0.0440399779719584, col2=-2.5929222741361313, col3=-0.6678776757343482)
Pandas(Index=3, col1=0.4913220600881253, col2=-0.25539685728373845, col3=-0.4035355020023611)
Pandas(Index=4, col1=-0.6978373026721117, col2=-0.06824738456676405, col3=-1.6737286452988054)
Pandas(Index=5, col1=1.5248577811033595, col2=-0.7601288287682813, col3=2.612702093150637)
```

Conclusion –

Thus we have successfully performed Descriptive Statistics, Function Application, Reindexing and Iteration functions.