Data Analysis And Practical - 4

Shad Jamil

CSC/21/45

In [4]:

```
import numpy as np
import pandas as pd
#Creating a dataframe with 3 columns and 50 rows
dat = pd.DataFrame(np.random.rand(50,3), columns=['Col1','Col2','Col3'])
#Replacing 10% values with null values
x = int(0.1*dat.size)
indices_to_replace = np.unravel_index(np.random.choice(dat.size, x, replac
dat.iloc[indices_to_replace] = np.nan
#Printing Results
print('Acquired DataFrame:\n')
print(dat)
```

```
Acquired DataFrame:

        Col1      Col2      Col3
0   0.682881  0.206784  0.013455
1   0.783925  0.393434  0.049236
2   0.772594  0.206481  0.288398
3   0.492280  0.484211  0.194326
4   0.197248  0.068652  0.268006
5   0.892160  0.080545  0.433682
6   0.305926  0.422013  0.604318
7   0.313714  0.043631  0.774311
8   0.339754  0.965632  0.034533
9   0.148361  0.725745  0.065957
10  0.454817  0.639184  0.845437
11  0.846323  0.828858  0.199262
12  0.021332  0.195676  0.139573
13  0.922980  0.738134  0.545251
14  0.967218  0.636276  0.668695
15       NaN       NaN       NaN
16       NaN       NaN       NaN
17  0.669454  0.831103  0.313997
18  0.185592  0.747234  0.676615
19  0.541976  0.502911  0.676689
20  0.280902  0.775334  0.883078
21       NaN       NaN       NaN
22  0.419839  0.531416  0.274781
23  0.683998  0.927282  0.632482
24       NaN       NaN       NaN
25       NaN       NaN       NaN
26  0.817794  0.678031  0.030699
27  0.356871  0.192558  0.062831
28  0.487123  0.912645  0.873654
29  0.048407  0.658765  0.620407
30  0.494716  0.308894  0.938305
31  0.820723  0.104469  0.264013
32  0.918657  0.797679  0.222038
33       NaN       NaN       NaN
34       NaN       NaN       NaN
35  0.900773  0.481834  0.259489
36  0.529321  0.710574  0.490034
37       NaN       NaN       NaN
38       NaN       NaN       NaN
39  0.458132  0.604310  0.754716
40  0.083181  0.165589  0.275054
41  0.396066  0.545951  0.461533
42       NaN       NaN       NaN
43  0.663116  0.556754  0.263829
44  0.030904  0.587509  0.907292
45  0.381704  0.326459  0.607293
46       NaN       NaN       NaN
47  0.010397  0.444512  0.551197
48  0.214805  0.222936  0.274977
49       NaN       NaN       NaN
```

In [5]:

```
#(a) - Identify and Count missing values
#Identifying missing values
missing_values_identified_df = dat.isnull()
#Counting missing values
total_missing_values = missing_values_identified_df.sum().sum()
#Printing Results
print('DataFrame With Missing Values:\n')
print(missing_values_identified_df)
print('Total Missing Values:', total_missing_values)
```

```
DataFrame With Missing Values:

     Col1   Col2   Col3
0   False  False  False
1   False  False  False
2   False  False  False
3   False  False  False
4   False  False  False
5   False  False  False
6   False  False  False
7   False  False  False
8   False  False  False
9   False  False  False
10  False  False  False
11  False  False  False
12  False  False  False
13  False  False  False
14  False  False  False
15   True   True   True
16   True   True   True
17  False  False  False
18  False  False  False
19  False  False  False
20  False  False  False
21   True   True   True
22  False  False  False
23  False  False  False
24   True   True   True
25   True   True   True
26  False  False  False
27  False  False  False
28  False  False  False
29  False  False  False
30  False  False  False
31  False  False  False
32  False  False  False
33   True   True   True
34   True   True   True
35  False  False  False
36  False  False  False
37   True   True   True
38   True   True   True
39  False  False  False
40  False  False  False
41  False  False  False
42   True   True   True
43  False  False  False
44  False  False  False
45  False  False  False
46   True   True   True
47  False  False  False
48  False  False  False
49   True   True   True
Total Missing Values: 36
```

In [6]:

```python
#(b) - Dropping column with more than 5 null values
#Checking for columns with more than 5 null values
columns_to_drop = dat.columns[dat.isnull().sum()>5]
#Dropping appropriate columns
columns_dropped_df = dat.drop(columns=columns_to_drop)
#Printing Results
print('DataFrame With Columns Dropped:\n')
print(columns_dropped_df)

```

```
DataFrame With Columns Dropped:

Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 3
9, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
```

In [7]:

```
#(c) - Identify the row label having maximum of the sum of all values in c
#Identifying the row label having max sum
max_sum_row = dat.sum(axis=1).idxmax()
#Dropping appropriate column
max_sum_row_dropped_df = dat.drop(max_sum_row)
#Printing Results
print('Row Index with Max Sum:', max_sum_row, '\n')
print('DataFrame with Row With Max Sum Dropped:\n')
print(max_sum_row_dropped_df)
```

```
Row Index with Max Sum: 28

DataFrame with Row With Max Sum Dropped:

        Col1      Col2      Col3
0   0.682881  0.206784  0.013455
1   0.783925  0.393434  0.049236
2   0.772594  0.206481  0.288398
3   0.492280  0.484211  0.194326
4   0.197248  0.068652  0.268006
5   0.892160  0.080545  0.433682
6   0.305926  0.422013  0.604318
7   0.313714  0.043631  0.774311
8   0.339754  0.965632  0.034533
9   0.148361  0.725745  0.065957
10  0.454817  0.639184  0.845437
11  0.846323  0.828858  0.199262
12  0.021332  0.195676  0.139573
13  0.922980  0.738134  0.545251
14  0.967218  0.636276  0.668695
15       NaN       NaN       NaN
16       NaN       NaN       NaN
17  0.669454  0.831103  0.313997
18  0.185592  0.747234  0.676615
19  0.541976  0.502911  0.676689
20  0.280902  0.775334  0.883078
21       NaN       NaN       NaN
22  0.419839  0.531416  0.274781
23  0.683998  0.927282  0.632482
24       NaN       NaN       NaN
25       NaN       NaN       NaN
26  0.817794  0.678031  0.030699
27  0.356871  0.192558  0.062831
29  0.048407  0.658765  0.620407
30  0.494716  0.308894  0.938305
31  0.820723  0.104469  0.264013
32  0.918657  0.797679  0.222038
33       NaN       NaN       NaN
34       NaN       NaN       NaN
35  0.900773  0.481834  0.259489
36  0.529321  0.710574  0.490034
37       NaN       NaN       NaN
38       NaN       NaN       NaN
39  0.458132  0.604310  0.754716
40  0.083181  0.165589  0.275054
41  0.396066  0.545951  0.461533
42       NaN       NaN       NaN
43  0.663116  0.556754  0.263829
44  0.030904  0.587509  0.907292
45  0.381704  0.326459  0.607293
46       NaN       NaN       NaN
47  0.010397  0.444512  0.551197
48  0.214805  0.222936  0.274977
49       NaN       NaN       NaN
```

In [8]:

```python
#(d) - Sort the dataframe on the basis of the first column
sorted_df = dat.sort_values(by='Col1')
#Printing Results
print('DataFrame Sorted by Col1:\n')
print(sorted_df)

```

```
DataFrame Sorted by Col1:

        Col1      Col2      Col3
47  0.010397  0.444512  0.551197
12  0.021332  0.195676  0.139573
44  0.030904  0.587509  0.907292
29  0.048407  0.658765  0.620407
40  0.083181  0.165589  0.275054
9   0.148361  0.725745  0.065957
18  0.185592  0.747234  0.676615
4   0.197248  0.068652  0.268006
48  0.214805  0.222936  0.274977
20  0.280902  0.775334  0.883078
6   0.305926  0.422013  0.604318
7   0.313714  0.043631  0.774311
8   0.339754  0.965632  0.034533
27  0.356871  0.192558  0.062831
45  0.381704  0.326459  0.607293
41  0.396066  0.545951  0.461533
22  0.419839  0.531416  0.274781
10  0.454817  0.639184  0.845437
39  0.458132  0.604310  0.754716
28  0.487123  0.912645  0.873654
3   0.492280  0.484211  0.194326
30  0.494716  0.308894  0.938305
36  0.529321  0.710574  0.490034
19  0.541976  0.502911  0.676689
43  0.663116  0.556754  0.263829
17  0.669454  0.831103  0.313997
0   0.682881  0.206784  0.013455
23  0.683998  0.927282  0.632482
2   0.772594  0.206481  0.288398
1   0.783925  0.393434  0.049236
26  0.817794  0.678031  0.030699
31  0.820723  0.104469  0.264013
11  0.846323  0.828858  0.199262
5   0.892160  0.080545  0.433682
35  0.900773  0.481834  0.259489
32  0.918657  0.797679  0.222038
13  0.922980  0.738134  0.545251
14  0.967218  0.636276  0.668695
15       NaN       NaN       NaN
16       NaN       NaN       NaN
21       NaN       NaN       NaN
24       NaN       NaN       NaN
25       NaN       NaN       NaN
33       NaN       NaN       NaN
34       NaN       NaN       NaN
37       NaN       NaN       NaN
38       NaN       NaN       NaN
42       NaN       NaN       NaN
46       NaN       NaN       NaN
49       NaN       NaN       NaN
```

In [9]:

```python
#(e) - Remove all duplicates from the first column
duplicates_dropped_df = dat.drop_duplicates(subset='Col1')
#Printing Results
print('DataFrame with Duplicates Removed from Col1:\n')
print(duplicates_dropped_df)
```

```
DataFrame with Duplicates Removed from Col1:

        Col1      Col2      Col3
0   0.682881  0.206784  0.013455
1   0.783925  0.393434  0.049236
2   0.772594  0.206481  0.288398
3   0.492280  0.484211  0.194326
4   0.197248  0.068652  0.268006
5   0.892160  0.080545  0.433682
6   0.305926  0.422013  0.604318
7   0.313714  0.043631  0.774311
8   0.339754  0.965632  0.034533
9   0.148361  0.725745  0.065957
10  0.454817  0.639184  0.845437
11  0.846323  0.828858  0.199262
12  0.021332  0.195676  0.139573
13  0.922980  0.738134  0.545251
14  0.967218  0.636276  0.668695
15       NaN       NaN       NaN
17  0.669454  0.831103  0.313997
18  0.185592  0.747234  0.676615
19  0.541976  0.502911  0.676689
20  0.280902  0.775334  0.883078
22  0.419839  0.531416  0.274781
23  0.683998  0.927282  0.632482
26  0.817794  0.678031  0.030699
27  0.356871  0.192558  0.062831
28  0.487123  0.912645  0.873654
29  0.048407  0.658765  0.620407
30  0.494716  0.308894  0.938305
31  0.820723  0.104469  0.264013
32  0.918657  0.797679  0.222038
35  0.900773  0.481834  0.259489
36  0.529321  0.710574  0.490034
39  0.458132  0.604310  0.754716
40  0.083181  0.165589  0.275054
41  0.396066  0.545951  0.461533
43  0.663116  0.556754  0.263829
44  0.030904  0.587509  0.907292
45  0.381704  0.326459  0.607293
47  0.010397  0.444512  0.551197
48  0.214805  0.222936  0.274977
```

In [11]:

```python
#(f) - Find the correlation between first and second column and covariance
#Correlation (1st - 2nd col)
corr_1_2 = dat['Col1'].corr(dat['Col2'])
#Covariance (2nd - 3rd col)
cov_2_3 = dat['Col2'].cov(dat['Col3'])
#Printing Results
print('Correlation b/w 1st and 2nd Column:', corr_1_2)
print('Covariance b/w 2nd and 3rd Column:',cov_2_3)
```

```
Correlation b/w 1st and 2nd Column: 0.1420138739574065
Covariance b/w 2nd and 3rd Column: 0.013539665909745479
```

In [12]:

```
#(g) - Detect the outliers and remove the rows having outliers
#Function to Detect and Remove Outliers
def remove_outliers(dat, zscore_thresh=3):
 z_scores = np.abs((dat - dat.mean()) / dat.std())
 outlier_mask = (z_scores < zscore_thresh).all(axis=1)
 new_df = dat[outlier_mask]
 return new_df
#Removing Outliers From Data
outliers_removed_df = remove_outliers(dat)
#Printing Results
print('DataFrame with Outliers Removed:\n')
print(outliers_removed_df)

```

```
DataFrame with Outliers Removed:

        Col1      Col2      Col3
0   0.682881  0.206784  0.013455
1   0.783925  0.393434  0.049236
2   0.772594  0.206481  0.288398
3   0.492280  0.484211  0.194326
4   0.197248  0.068652  0.268006
5   0.892160  0.080545  0.433682
6   0.305926  0.422013  0.604318
7   0.313714  0.043631  0.774311
8   0.339754  0.965632  0.034533
9   0.148361  0.725745  0.065957
10  0.454817  0.639184  0.845437
11  0.846323  0.828858  0.199262
12  0.021332  0.195676  0.139573
13  0.922980  0.738134  0.545251
14  0.967218  0.636276  0.668695
17  0.669454  0.831103  0.313997
18  0.185592  0.747234  0.676615
19  0.541976  0.502911  0.676689
20  0.280902  0.775334  0.883078
22  0.419839  0.531416  0.274781
23  0.683998  0.927282  0.632482
26  0.817794  0.678031  0.030699
27  0.356871  0.192558  0.062831
28  0.487123  0.912645  0.873654
29  0.048407  0.658765  0.620407
30  0.494716  0.308894  0.938305
31  0.820723  0.104469  0.264013
32  0.918657  0.797679  0.222038
35  0.900773  0.481834  0.259489
36  0.529321  0.710574  0.490034
39  0.458132  0.604310  0.754716
40  0.083181  0.165589  0.275054
41  0.396066  0.545951  0.461533
43  0.663116  0.556754  0.263829
44  0.030904  0.587509  0.907292
45  0.381704  0.326459  0.607293
47  0.010397  0.444512  0.551197
48  0.214805  0.222936  0.274977
```

In [13]:

```
#(h) - Discretize second column and create 5 bins
#Filling Missing (NaN) Values in Col2 because pandas.cut cannot appropriat
dat['Col2'] = dat['Col2'].fillna(dat['Col2'].mean())
#Envoking pandas.cut funtion
dat['Col2_Discretized'] = pd.cut(dat.iloc[:,1], bins=5, labels=False)
#Printing Results
print('DataFrame with Col2 Discretized into 5 bins:\n')
print(dat)

```

```
DataFrame with Col2 Discretized into 5 bins:

        Col1      Col2      Col3  Col2_Discretized
0   0.682881  0.206784  0.013455                 0
1   0.783925  0.393434  0.049236                 1
2   0.772594  0.206481  0.288398                 0
3   0.492280  0.484211  0.194326                 2
4   0.197248  0.068652  0.268006                 0
5   0.892160  0.080545  0.433682                 0
6   0.305926  0.422013  0.604318                 2
7   0.313714  0.043631  0.774311                 0
8   0.339754  0.965632  0.034533                 4
9   0.148361  0.725745  0.065957                 3
10  0.454817  0.639184  0.845437                 3
11  0.846323  0.828858  0.199262                 4
12  0.021332  0.195676  0.139573                 0
13  0.922980  0.738134  0.545251                 3
14  0.967218  0.636276  0.668695                 3
15       NaN  0.506579       NaN                 2
16       NaN  0.506579       NaN                 2
17  0.669454  0.831103  0.313997                 4
18  0.185592  0.747234  0.676615                 3
19  0.541976  0.502911  0.676689                 2
20  0.280902  0.775334  0.883078                 3
21       NaN  0.506579       NaN                 2
22  0.419839  0.531416  0.274781                 2
23  0.683998  0.927282  0.632482                 4
24       NaN  0.506579       NaN                 2
25       NaN  0.506579       NaN                 2
26  0.817794  0.678031  0.030699                 3
27  0.356871  0.192558  0.062831                 0
28  0.487123  0.912645  0.873654                 4
29  0.048407  0.658765  0.620407                 3
30  0.494716  0.308894  0.938305                 1
31  0.820723  0.104469  0.264013                 0
32  0.918657  0.797679  0.222038                 4
33       NaN  0.506579       NaN                 2
34       NaN  0.506579       NaN                 2
35  0.900773  0.481834  0.259489                 2
36  0.529321  0.710574  0.490034                 3
37       NaN  0.506579       NaN                 2
38       NaN  0.506579       NaN                 2
39  0.458132  0.604310  0.754716                 3
40  0.083181  0.165589  0.275054                 0
41  0.396066  0.545951  0.461533                 2
42       NaN  0.506579       NaN                 2
43  0.663116  0.556754  0.263829                 2
44  0.030904  0.587509  0.907292                 2
45  0.381704  0.326459  0.607293                 1
46       NaN  0.506579       NaN                 2
47  0.010397  0.444512  0.551197                 2
48  0.214805  0.222936  0.274977                 0
49       NaN  0.506579       NaN                 2
```