CO2160714.1 Assignment:

1. **Analyse 5 data sets from the UCI repository. Print the following details about each data set**
   - **(a) number of records/instances**
   - **(b) number of incomplete records**
   - **(c) number of attributes**

Input:

```
import fnmatch
import os
import pandas as pd

f_n=[]
for f in os.listdir('.'):
   if fnmatch.fnmatch(f, '*.csv'):
      f_n.append(f)
print ("Total files are:",f_n)
for i in f_n:
   print("\nFile name is ",i)
   i=pd.read_csv(i,delimiter=',')
   k=i.keys()
   #print(len(k))
   print("Total number of records/instances:",len(i))
   print("Total number of null records:",i.isnull().values.sum())

   c1=0
   c=i['lat'].count().sum()
   for x in range(0,c):
      if((i['lat'][x]< -90 or i['lat'][x]>90) and (i['long'][x]< -180 or i['long'][x]>180) ):
          c1=c1+1
   print("Total number of Inconsistant Data:",c1)
   print("Total number of attributes:",len(k))
```

Output:

```
Total files are: ['airport-codes1.csv', 'NYC_Airbnb.csv', 'urbanGB.csv
', 'worldcities.csv', 'world_country_and_usa_states_latitude_and_longi
tude_values.csv']

File name is  airport-codes1.csv
Total number of records/instances: 99
Total number of null records: 215
Total number of Inconsistant Data: 2
Total number of attributes: 13

File name is  NYC_Airbnb.csv
Total number of records/instances: 5
Total number of null records: 2
Total number of Inconsistant Data: 0
Total number of attributes: 5
```

```
File name is  urbanGB.csv
Total number of records/instances: 52
Total number of null records: 32
Total number of Inconsistant Data: 2
Total number of attributes: 3

File name is  worldcities.csv
Total number of records/instances: 26569
Total number of null records: 20036
Total number of Inconsistant Data: 0
Total number of attributes: 11

File name is  world_country_and_usa_states_latitude_and_longitude_valu
es.csv
Total number of records/instances: 245
Total number of null records: 775
Total number of Inconsistant Data: 2
Total number of attributes: 8
```

2.	**Write a program to implement data cleaning(*incomplete, noisy, inconsistent, redundant*) on your data set. Implement each technique.** **(a). Binning with means and/or mode**

Input:

```
import pandas as pd
import numpy as np
df1=pd.read_csv("airport-codes1.csv")
df=[]
for i in range(0,50):
   df.append(df1['elevation_ft'][i])
print("Original Data:",df)
arr_1D= np.sort(df)
print("Sorted Data : ",arr_1D)
nb=input("Enter how many bins you want to create:")
size=len(df)//int(nb)
means=[]
meadians=[]
bins=[arr_1D[i:i+size] for i in range(0, len(arr_1D), size)]
print("Bins : ")
for x in bins:
   print(x)
   m=np.mean(x)
   m1=np.median(x)
   for y in x:
      means.append(m)
      meadians.append(m1)
mean_bin=[means[i:i+size] for i in range(0, len(means), size)]
print("After Smoothing Using Mean:")
for x in mean_bin:
   print(x)
median_bin=[meadians[i:i+size] for i in range(0, len(meadians), size)]
print("After Smoothing Using Median:")
for x in median_bin:
   print(x)
d_b=[]
for x in bins:
   lb=x[0]
   rb=x[-1]
   d_b.append(lb)
   for y in range(1,len(x)-1):
      if((x[y]-lb)<(rb-x[y])):
         d_b.append(lb)
      else:
         d_b.append(rb)
   d_b.append(rb)
bbin=[d_b[i:i+size] for i in range(0, len(d_b), size)]
print("Smoothing using boundaries : ")
for i in bbin:
```

	print(i)
	Output:

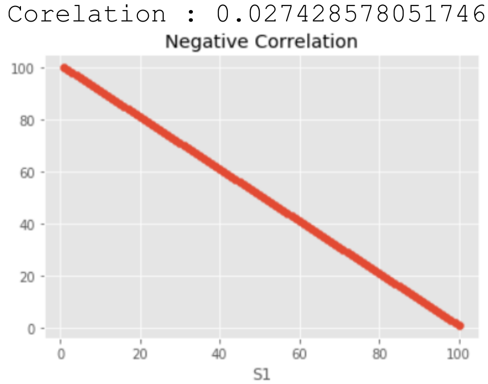
```
Original Data: [11.0, 3435.0, 450.0, 820.0, 237.0, 1100.0, 3810.0, 303
8.0, 87.0, 3350.0, 4830.0, 53.0, 25.0, 35.0, 700.0, 957.0, 43.0, 2064.
0, 3359.0, 600.0, 840.0, 634.0, 820.0, 1100.0, 1265.0, 15.0, 600.0, 12
.0, 45.0, 588.0, 1365.0, 970.0, 2600.0, 105.0, 348.0, 78.0, 96.0, 1000
.0, 785.0, 905.0, 960.0, 195.0, 402.0, 1301.0, 815.0, 1620.0, 150.0, 1
590.0, 598.0, 600.0]
Sorted Data :  [  11.   12.   15.   25.   35.   43.   45.   53.   78.
87.   96.  105.
  150.  195.  237.  348.  402.  450.  588.  598.  600.  600.  600.  63
4.
  700.  785.  815.  820.  820.  840.  905.  957.  960.  970. 1000. 110
0.
 1100. 1265. 1301. 1365. 1590. 1620. 2064. 2600. 3038. 3350. 3359. 343
5.
 3810. 4830.]
Enter how many bins you want to create:5
Bins :
[11. 12. 15. 25. 35. 43. 45. 53. 78. 87.]
[ 96. 105. 150. 195. 237. 348. 402. 450. 588. 598.]
[600. 600. 600. 634. 700. 785. 815. 820. 820. 840.]
[ 905.  957.  960.  970. 1000. 1100. 1100. 1265. 1301. 1365.]
[1590. 1620. 2064. 2600. 3038. 3350. 3359. 3435. 3810. 4830.]
After Smoothing Using Mean:
[40.4, 40.4, 40.4, 40.4, 40.4, 40.4, 40.4, 40.4, 40.4, 40.4]
[316.9, 316.9, 316.9, 316.9, 316.9, 316.9, 316.9, 316.9, 316.9, 316.9]
[721.4, 721.4, 721.4, 721.4, 721.4, 721.4, 721.4, 721.4, 721.4, 721.4]
[1092.3, 1092.3, 1092.3, 1092.3, 1092.3, 1092.3, 1092.3, 1092.3, 1092.
3, 1092.3]
[2969.6, 2969.6, 2969.6, 2969.6, 2969.6, 2969.6, 2969.6, 2969.6, 2969.
6, 2969.6]
After Smoothing Using Median:
[39.0, 39.0, 39.0, 39.0, 39.0, 39.0, 39.0, 39.0, 39.0, 39.0]
[292.5, 292.5, 292.5, 292.5, 292.5, 292.5, 292.5, 292.5, 292.5, 292.5]
[742.5, 742.5, 742.5, 742.5, 742.5, 742.5, 742.5, 742.5, 742.5, 742.5]
[1050.0, 1050.0, 1050.0, 1050.0, 1050.0, 1050.0, 1050.0, 1050.0, 1050.
0, 1050.0]
[3194.0, 3194.0, 3194.0, 3194.0, 3194.0, 3194.0, 3194.0, 3194.0, 3194.
0, 3194.0]
Smoothing using boundaries :
[11.0, 11.0, 11.0, 11.0, 11.0, 11.0, 11.0, 87.0, 87.0, 87.0]
[96.0, 96.0, 96.0, 96.0, 96.0, 598.0, 598.0, 598.0, 598.0, 598.0]
[600.0, 600.0, 600.0, 600.0, 600.0, 840.0, 840.0, 840.0, 840.0, 840.0]
[905.0, 905.0, 905.0, 905.0, 905.0, 905.0, 905.0, 1365.0, 1365.0, 1365
.0]
[1590.0, 1590.0, 1590.0, 1590.0, 1590.0, 4830.0, 4830.0, 4830.0, 4830.
0, 4830.0]
```

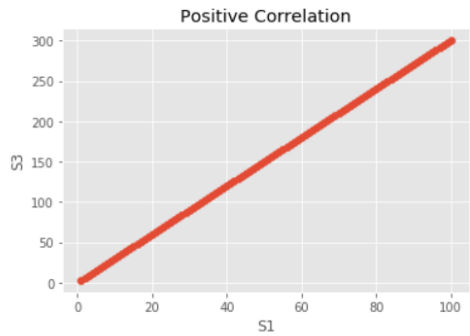
2.	**Write a program to implement data cleaning(*incomplete, noisy, inconsistent, redundant*) on your data set. Implement each technique.** **(b.) Find covariance(cov) and correlation(r), Sx and Sy are standard deviation,** $\bar{x}$ **and** $\bar{y}$ **are means.** $$Cov(x,y) = \sum_{i=1}^{n} (X_i - \underline{X})(Y_i - \underline{Y})/(n-1)$$ $$r_{(x,y)} = Cov(x,y)/s_x s_y$$ **Plot the correlation, to show whether two variables are positively correlated, negatively correlated or no relation between them.**

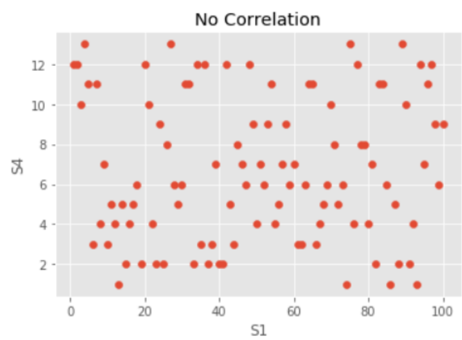
Input:

```
from numpy import array
import pandas as pd
import numpy as np
def covari(x,y):
   xmean  = np.mean(x)
   ymean = np.mean(y)
   n = len(data)
   ans = 0
   for i in range(0,n):
      ans = ans + (x[i]-xmean) * (y[i]-ymean)
      result = ans / (n-1)
   return round(result,2)
def Corelation(x,y,cov):
   sx = np.std(x)
   sy = np.std(y)
   rxy = cov / (sx * sy)
   return rxy
usecols=['S1', 'S2','S3','S4']
data = pd.read_csv("soccer_goals.csv",delimiter=',', names=usecols)
S1=array(data['S1'])
S2=array(data['S2'])
S3=array(data['S3'])
S4=array(data['S4'])
co=covari(S1,S2)
rel=Corelation(S1,S2,co)
print("Covarience is:",co)
print("Corelation :",rel)
co=covari(S1,S3)
rel=Corelation(S1,S3,co)
print("Covarience is:",co)
print("Corelation :",rel)
co=covari(S1,S4)
rel=Corelation(S1,S4,co)
print("Covarience is:",co)
print("Corelation :",rel)

import matplotlib
```

	import matplotlib.pyplot as plt matplotlib.style.use('ggplot') plt.scatter(S1, S2) plt.xlabel("S1") plt.ylabel("S2") plt.title("Negative Correlation") plt.show() plt.scatter(S1, S3) plt.xlabel("S1") plt.ylabel("S3") plt.title("Positive Correlation") plt.show() plt.scatter(S1, S4) plt.xlabel("S1") plt.ylabel("S4") plt.title("No Correlation") plt.show()
	Output: Covarience is: -841.67 Corelation : -1.01010501050105 Covarience is: 2525.0 Corelation : 1.0101010101010102 Covarience is: 2.85 Corelation : 0.027428578051746585 


Positive Correlation
S3
S1
0
50
100
150
200
250
300
0
20
40
60
80
100


No Correlation
S4
S1
2
4
6
8
10
12
0
20
40
60
80
100

3. **Implement chi-square test to detect whether two variables are dependent or independent variables for your dataset.**

Input:

```
import pandas as pd
import numpy as np
import scipy.stats as stats
from scipy.stats import chi2_contingency
class ChiSquare:
    def __init__(self, dataframe):
        self.df = dataframe
        self.p = None #P-Value
        self.chi2 = None
        self.dof = None
        self.dfObserved = None
        self.dfExpected = None
    def _print_chisquare_result(self, colX, alpha):
        result = ""
        if self.p<alpha:
            result="{0} is IMPORTANT for Prediction".format(colX)
        else:
            result="{0} is NOT an important predictor. (Discard {0} from model)".format(colX)
        print(result)
    def TestIndependence(self,colX,colY, alpha=0.09):
        X = self.df[colX].astype(str)
        Y = self.df[colY].astype(str)
        self.dfObserved = pd.crosstab(Y,X)
        chi2, p, dof, expected = stats.chi2_contingency(self.dfObserved.values)
        self.p = p
        self.chi2 = chi2
        self.dof = dof
        self.dfExpected = pd.DataFrame(expected, columns=self.dfObserved.columns, index =
self.dfObserved.index)
        self._print_chisquare_result(colX, alpha)
n=['ident','type','name','elevation_ft','continent','iso_country','iso_region','municipality','gps_code','i
ata_code','local_code','lat','long']
df = pd.pandas.read_csv("airport-codes1.csv",delimiter=",",names=n)
#Initialize ChiSquare Class
cT = ChiSquare(df)
#Feature Selection
testColumns = ['type','local_code','lat','long']
for var in testColumns:
    cT.TestIndependence(colX=var,colY="ident" )
```

Output:

```
type is NOT an important predictor. (Discard type from model)
local_code is NOT an important predictor. (Discard local_code from mode
l)
lat is NOT an important predictor. (Discard lat from model)
long is NOT an important predictor. (Discard long from model)
```

4.	**Write a program to implement normalization techniques (a)min max (b) z-score (c) decimal scaling on your data set.**
	Input: import statistics as st import pandas as pd usecols=['ident','type','name','elevation_ft','continent','iso_country','iso_region','municipality','gps_code','iata_code','local_code','lat','long'] data1 = pd.read_csv("airport-codes1.csv",delimiter=',',names=usecols) data=data1['lat'] newmin = 0 newmax = 1 v = 10 minmax = newmin+((v-min(data))*(newmax-newmin)/(max(data)-min(data))) print("minmax normalization : ",round(minmax,2)) zscore = (v-st.mean(data))/st.stdev(data) print("z-score : ", zscore) decimalscale = v/100 print("decimal scaling : ",decimalscale)
	Output: minmax normalization : 2.11 z-score : 5.31864647094237 decimal scaling : 0.1

5.	**Write a program to implement data reduction techniques for your data.**

Input:

```
import pandas as pd

usecols=['ident','type','name','elevation_ft','continent','iso_country','iso_region','municipality','gps_code','iata_code','local_code','lat','long']
   df = pd.read_csv('airport-codes1.csv',names= usecols)
   print("BEFORE REDUCTION")
   print("Rows:",df.shape[0]," Columns: ",df.shape[1])
   print("SIZE:",df.size)
   columns = ['ident','type','name','iso_country','iso_region']
   df.drop(columns,axis=1, inplace = True)
   print("\n AFTER REDUCTION")
   print("Rows:",df.shape[0]," Columns: ",df.shape[1])
   print("SIZE:",df.size)
```

Output:

```
STRING/DATA REDUCTION

BEFORE REDUCTION
Rows: 14  Columns:  13
SIZE: 182

 AFTER REDUCTION
Rows: 14  Columns:  8
SIZE: 112
```

6.	**Write a program to implement any method of data discretization.**

Input:

```
#using label
from numpy import array
import pandas as pd
usecols=['ident','type','name','elevation_ft','continent','iso_country','iso_region','municipality','gps_code','iata_code','local_code','lat','long']
data = pd.read_csv("airport-codes1.csv",delimiter=',', names=usecols)
c=[]
card1=data['type']
for x in range(0,13):
      c.append(card1[x])
print("Your Data")
print(c)
heliport=[]
closed=[]
small_airport=[]
for x in c:
   if(x=='heliport'):
      heliport.append(x)
   elif(x=='closed'):
      closed.append(x)
   else:
      small_airport.append(x)

print("\nheliport : ",heliport)
print("\nsmall_airport : ",small_airport)
print("\nclosed : ",closed)
```

Output:

```
Your Data
['heliport', 'small_airport', 'small_airport', 'small_airport', 'closed'
, 'small_airport', 'small_airport', 'small_airport', 'small_airport', 'h
eliport', 'closed', 'small_airport', 'heliport']

heliport :  ['heliport', 'heliport', 'heliport']

small_airport :  ['small_airport', 'small_airport', 'small_airport', 'sm
all_airport', 'small_airport', 'small_airport', 'small_airport', 'small_
airport']

closed :  ['closed', 'closed']
```

	CO2160714.2 Assignment:
7.	**Implement apriori algorithm and show the output as candidate sets in each iteration, as well as show association rules generated.**

Input:

```
# apriory
import itertools
support = int(input("Please enter support value in %: "))
confidence = int(input("Please enter confidence value in %: "))
C1 = {}

transactions = 0
D = []
T = []
with open("foods_type.csv", "r") as f:
    for line in f:
        T = []
        transactions += 1
        for word in line.split(','):
            T.append(word)
            if word not in C1.keys():
                C1[word] = 1
            else:
                count = C1[word]
                C1[word] = count + 1
        D.append(T)
print ("Dataset:",D)

#Computing frequent dataitems-1
L1 = []
for key in C1:
    if (100 * C1[key]/transactions) >= support:
        list = []
        list.append(key)
        L1.append(list)
print ("\nFrequent ItemSet: 1")
print (L1)

def apriori_gen(Lk_1, k):
    length = k
    Ck = []
    for list1 in Lk_1:
        for list2 in Lk_1:
            count = 0
            c = []
            if list1 != list2:
                while count < length-1:
                    if list1[count] != list2[count]:
                        break
```

```
                else:
                    count += 1
            else:
                if list1[length-1] < list2[length-1]:
                    for item in list1:
                        c.append(item)
                    c.append(list2[length-1])
                    if not has_infrequent_subset(c, Lk_1, k):
                        Ck.append(c)
                        c = []
    return Ck

def findsubsets(S,m):
    return set(itertools.combinations(S, m))

def has_infrequent_subset(c, Lk_1, k):
    list = []
    list = findsubsets(c,k)
    for item in list:
        s = []
        for l in item:
            s.append(l)
        s.sort()
        if s not in Lk_1:
            return True
    return False

def frequent_itemsets():
    k = 2
    Lk_1 = []
    Lk = []
    L = []
    count = 0
    transactions = 0
    for item in L1:
        Lk_1.append(item)
    while Lk_1 != []:
        Ck = []
        Lk = []
        Ck = apriori_gen(Lk_1, k-1)
        for c in Ck:
            count = 0
            transactions = 0
            s = set(c)
            for T in D:
                transactions += 1
                t = set(T)
                if s.issubset(t) == True:
```

```
                count += 1
            if (100 * count/transactions) >= support:
                c.sort()
                Lk.append(c)
        Lk_1 = []
        print ("\nFrequent Itemset: %d" % k)
        print (Lk)

        for l in Lk:
            Lk_1.append(l)
        k += 1
        if Lk != []:
            L.append(Lk)

    return L

def generate_association_rules():
    s = []
    r = []
    length = 0
    count = 1
    inc1 = 0
    inc2 = 0
    num = 1
    m = []
    L= frequent_itemsets()
    print ("\nAssociation Rules:\n\n")
    for list in L:
        for l in list:
            length = len(l)
            count = 1
            while count < length:
                s = []
                r = findsubsets(l,count)
                count += 1
                for item in r:
                    inc1 = 0
                    inc2 = 0
                    s = []
                    m = []
                    for i in item:
                        s.append(i)
                    for T in D:
                        if set(s).issubset(set(T)) == True:
                            inc1 += 1
                        if set(l).issubset(set(T)) == True:
                            inc2 += 1
                    if 100*inc2/inc1 >= confidence:
```

```
                for index in l:
                    if index not in s:
                        m.append(index)
                print ("%d. %s ==> %s\nSupport: %d\nConfidence: %d\n" %(num,s, m, 100*inc2/len(D),
100*inc2/inc1))

                num += 1

generate_association_rules()
```

Output:

```
Please enter support value in %: 50
Please enter confidence value in %: 50
Dataset: [['Cheese', 'Milk', 'Cookies\t\n'], ['Butter', 'Milk', 'Bread\t\
n'], ['Cheese', 'Butter', 'Milk', 'Bread\n'], ['Butter', 'Bread\t\t']]

Frequent ItemSet: 1
[['Cheese'], ['Milk'], ['Butter']]

Frequent Itemset: 2
[['Cheese', 'Milk'], ['Butter', 'Milk']]

Frequent Itemset: 3
[]

Association Rules:


1. ['Milk'] ==> ['Cheese']
Support: 50
Confidence: 66

2. ['Cheese'] ==> ['Milk']
Support: 50
Confidence: 100

3. ['Milk'] ==> ['Butter']
Support: 50
Confidence: 66

4. ['Butter'] ==> ['Milk']
Support: 50
Confidence: 66
```