Brian Truong

12/6/2021

Stat 2332 Final Project

Coded in Jupyter(Python)

Q1: Read the Final.csv data from D2L to R and Python and denote this data by d1.

import numpy as np

import pandas as pd

#q1

d1 = pd.read_csv("final.csv")

Q2: How many observations (number of rows) and Variables (columns) in the d1 data?

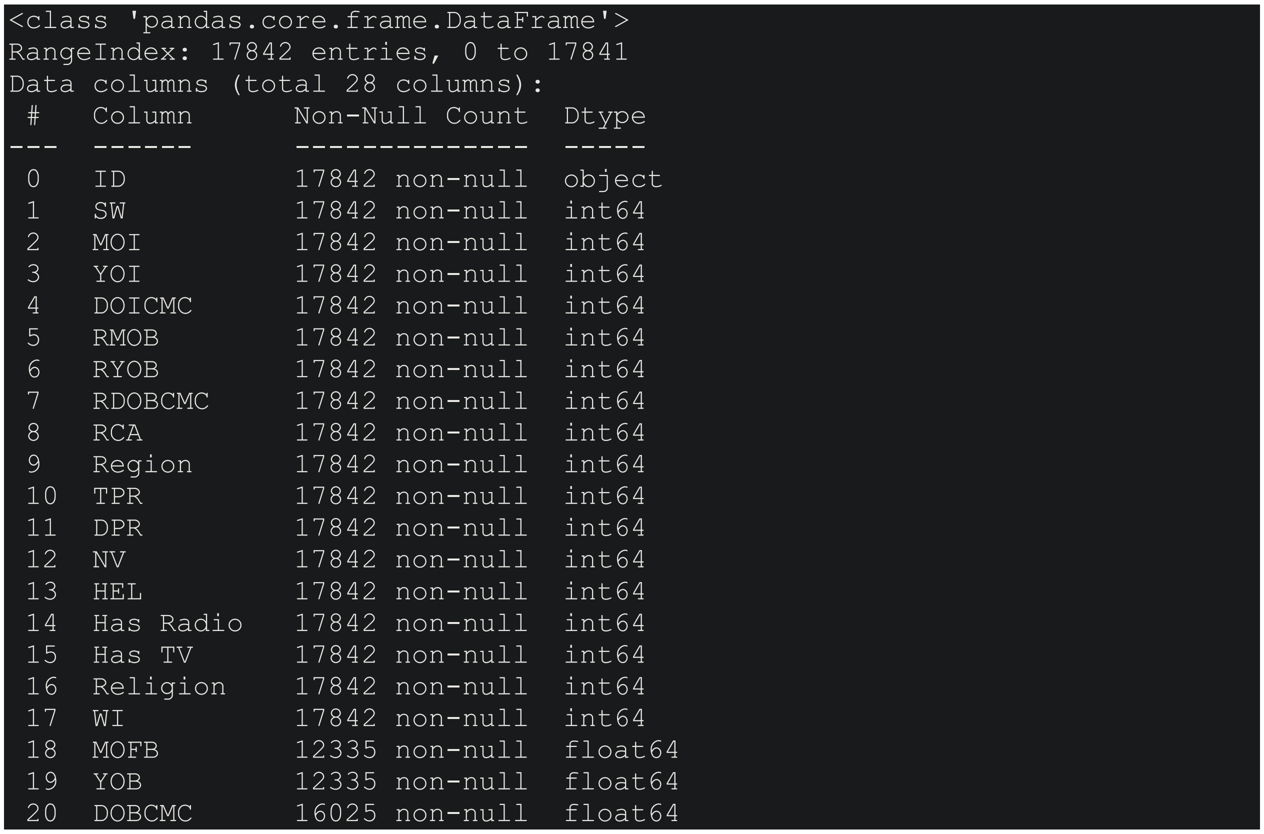
np.shape(d1)

Output: 17842 rows, 28 columns

Q3: How many variables are numerical/continuous and how many are them are integers/discrete?

d1.info()

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17842 entries, 0 to 17841
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          17842 non-null  object
 1   SW          17842 non-null  int64
 2   MOI         17842 non-null  int64
 3   YOI         17842 non-null  int64
 4   DOICMC      17842 non-null  int64
 5   RMOB        17842 non-null  int64
 6   RYOB        17842 non-null  int64
 7   RDOBCMC     17842 non-null  int64
 8   RCA         17842 non-null  int64
 9   Region      17842 non-null  int64
 10  TPR         17842 non-null  int64
 11  DPR         17842 non-null  int64
 12  NV          17842 non-null  int64
 13  HEL         17842 non-null  int64
 14  Has Radio   17842 non-null  int64
 15  Has TV      17842 non-null  int64
 16  Religion    17842 non-null  int64
 17  WI          17842 non-null  int64
 18  MOFB        12335 non-null  float64
 19  YOB         12335 non-null  float64
 20  DOBCMC      16025 non-null  float64
 21  DOFBCMC     16025 non-null  float64
 22  AOR         16025 non-null  float64
 23  MTFBI       16025 non-null  float64
 24  DSOUOM.CMC  10279 non-null  float64
 25  RW          17842 non-null  int64
 26  RH          17842 non-null  int64
 27  RBMI        17842 non-null  int64
dtypes: float64(7), int64(20), object(1)
memory usage: 3.8+ MB
```

Output:

Q4: Delete ID variable from the d1 data

d1.dtypes

del d1['ID']

d1.info()

Output:

```
RangeIndex: 17842 entries, 0 to 17841
Data columns (total 27 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   SW          17842 non-null  int64
 1   MOI         17842 non-null  int64
 2   YOI         17842 non-null  int64
 3   DOICMC      17842 non-null  int64
 4   RMOB        17842 non-null  int64
 5   RYOB        17842 non-null  int64
 6   RDOBCMC     17842 non-null  int64
 7   RCA         17842 non-null  int64
 8   Region      17842 non-null  int64
 9   TPR         17842 non-null  int64
 10  DPR         17842 non-null  int64
 11  NV          17842 non-null  int64
 12  HEL         17842 non-null  int64
 13  Has Radio   17842 non-null  int64
 14  Has TV      17842 non-null  int64
 15  Religion    17842 non-null  int64
 16  WI          17842 non-null  int64
 17  MOFB        12335 non-null  float64
 18  YOB         12335 non-null  float64
 19  DOBCMC      16025 non-null  float64
 20  DOFBCMC     16025 non-null  float64
 21  AOR         16025 non-null  float64
 22  MTFBI       16025 non-null  float64
 23  DSOUOM.CMC  10279 non-null  float64
 24  RW          17842 non-null  int64
 25  RH          17842 non-null  int64
 26  RBMI        17842 non-null  int64
dtypes: float64(7), int64(20)
```

```
memory usage: 3.7 MB
```

Q5: Report the number of missing values for the variables MOFB, YOB, and AOR.

temp = d1[['MOFB', 'YOB', 'AOR']].copy()

num_nan = temp.isna().sum()

print(num_nan)

Output:

```
MOFB    5507
YOB     5507
AOR     1817
dtype: int64
```

Q6: Create d2 data from d1 data by selecting variables RMOB, WI, RCA, Religion, Region, AOR, HEL, DOBCMC, DOFBCMC, MTFBI, RW, RH, and RBMI variables.

```
d2 = d1[['RMOB', 'WI', 'RCA', 'Religion', 'Region',
    'AOR', 'HEL', 'DOBCMC', 'DOFBCMC', 'MTFBI', 'RW',
    'RH', 'RBMI']].copy()
```

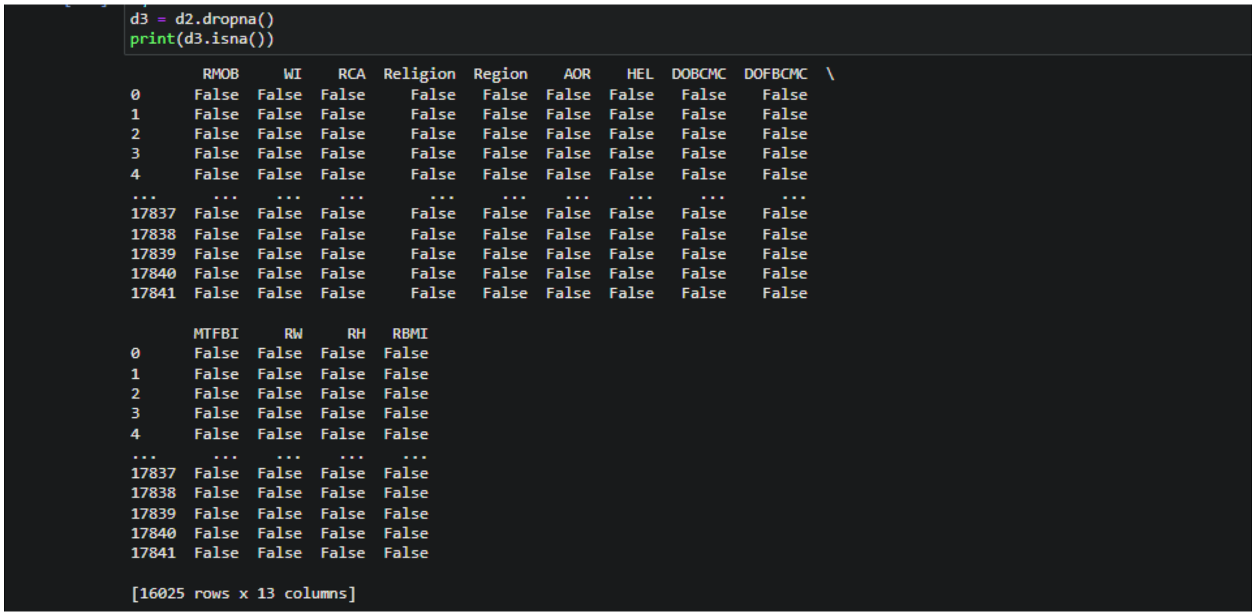
Output:

```
RangeIndex: 17842 entries, 0 to 17841
Data columns (total 13 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   RMOB      17842 non-null  int64
 1   WI        17842 non-null  int64
 2   RCA       17842 non-null  int64
 3   Religion  17842 non-null  int64
 4   Region    17842 non-null  int64
 5   AOR       16025 non-null  float64
 6   HEL       17842 non-null  int64
 7   DOBCMC    16025 non-null  float64
 8   DOFBCMC   16025 non-null  float64
 9   MTFBI     16025 non-null  float64
 10  RW        17842 non-null  int64
 11  RH        17842 non-null  int64
 12  RBMI      17842 non-null  int64
dtypes: float64(4), int64(9)
```

Q7: Delete rows that have missing values for any variable in the d2 data and denote this new data by d3.

d3 = d2.dropna()

Output:



```
d3 = d2.dropna()
print(d3.isna())

        RMOB     WI    RCA  Religion  Region    AOR    HEL  DOBCMC  DOFBCMC  \
0      False  False  False     False   False  False  False   False    False
1      False  False  False     False   False  False  False   False    False
2      False  False  False     False   False  False  False   False    False
3      False  False  False     False   False  False  False   False    False
4      False  False  False     False   False  False  False   False    False
...      ...    ...    ...       ...     ...    ...    ...     ...      ...
17837  False  False  False     False   False  False  False   False    False
17838  False  False  False     False   False  False  False   False    False
17839  False  False  False     False   False  False  False   False    False
17840  False  False  False     False   False  False  False   False    False
17841  False  False  False     False   False  False  False   False    False

       MTFBI     RW     RH   RBMI
0      False  False  False  False
1      False  False  False  False
2      False  False  False  False
3      False  False  False  False
4      False  False  False  False
...      ...    ...    ...    ...
17837  False  False  False  False
17838  False  False  False  False
17839  False  False  False  False
17840  False  False  False  False
17841  False  False  False  False

[16025 rows x 13 columns]
```

Q8: Find the summary statistics of the d3 data.

d3.describe()

output:

	RMOB	WI	RCA	Religion	Region	AOR	HEL	DOBCMC	DOFBCMC	MTFBI	RW	RH	RBMI
count	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000	16025.000000
mean	6.437067	3.128799	31.841373	1.120562	3.934727	17.902902	1.210109	1252.649360	1174.190078	35.436505	702.477566	1700.955257	2343.204431
std	3.491742	1.424111	8.886835	0.356198	1.901758	3.323532	0.930617	75.324979	107.317852	83.808245	1398.123557	1263.857453	1243.569493
min	1.000000	1.000000	13.000000	1.000000	1.000000	11.000000	0.000000	921.000000	893.000000	0.000000	229.000000	1044.000000	1245.000000

	RMOB	WI	RCA	Religion	Region	AOR	HEL	DOBCMC	DOFBCMC	MTFBI	RW	RH	RBMI
25%	3.000000	2.000000	24.000000	1.000000	2.000000	16.000000	0.000000	1212.000000	1091.000000	12.000000	423.00000	1474.000000	1876.000000
50%	6.000000	3.000000	31.000000	1.000000	4.000000	17.000000	1.000000	1273.000000	1188.000000	22.000000	482.00000	1510.000000	2114.000000
75%	10.000000	4.000000	39.000000	1.000000	6.000000	19.000000	2.000000	1311.000000	1264.000000	38.000000	556.00000	1547.000000	2420.000000
max	12.000000	5.000000	49.000000	4.000000	7.000000	40.000000	3.000000	1344.000000	1344.000000	996.00000	9999.000000	9999.000000	9999.000000

Q9: Add a new variable in the d3 data by finding the average of DOBCMC, DOFBCMC and MTFBI.

x = ['DOBCMC', 'DOFBCMC', 'MTFBI']

d3["average"] = d3[x].mean(axis = 1)

d3['average']

output:

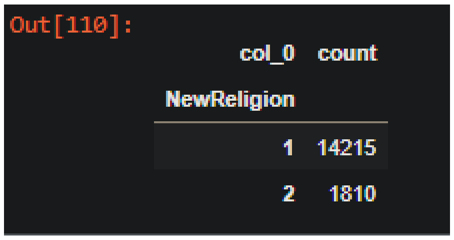
```
0        813.333333
1        907.000000
2        867.666667
3        793.666667
4        860.333333
            ...
17837    815.000000
17838    870.333333
17839    766.333333
17840    737.333333
17841    885.333333
```

Q10: Create a new variable named “Newreligion” by recoding ‘1’ as ‘1’ and rest as ‘2’ from the Religion Variable.

d3["NewReligion"] = d3['Religion']

d3.loc[d3['NewReligion'] != 1] = 2

Output:

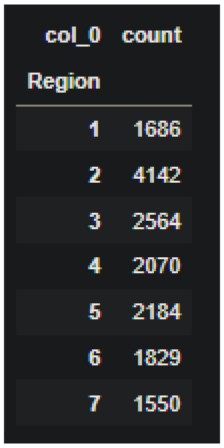


```
Out[110]:
                col_0   count
           NewReligion
                     1  14215
                     2   1810
```

Q11: Find the frequency table for the Region variable

pd.crosstab(index = d3["Region"], columns = "count")

Output:

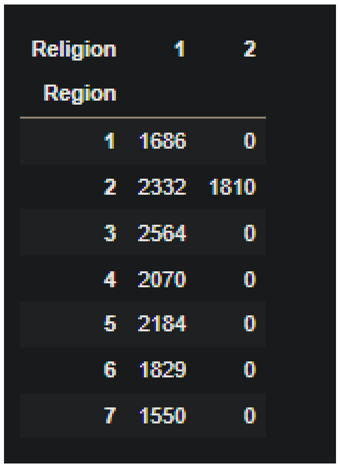


```
  col_0  count
 Region
      1   1686
      2   4142
      3   2564
      4   2070
      5   2184
      6   1829
      7   1550
```

Q12: Find the joint frequency table for the variables Region and Religion.

pd.crosstab(index = d3["Region"], columns = d3["Religion"])

Output:



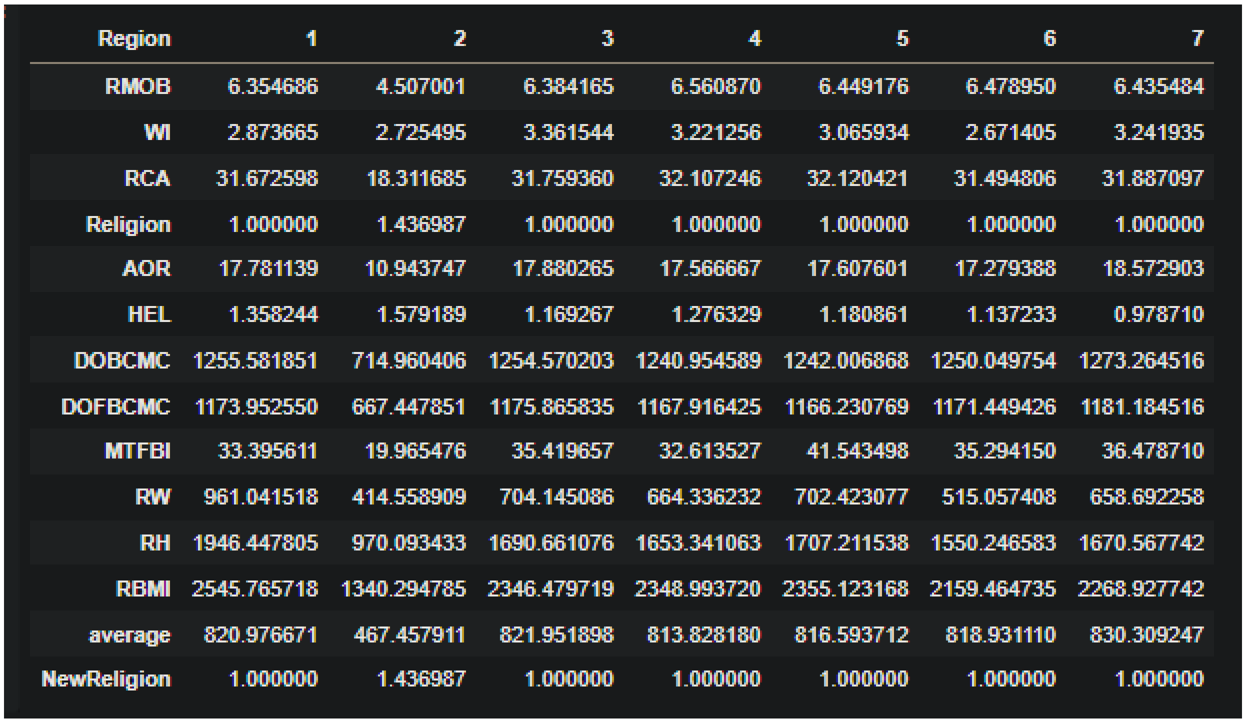
```
 Religion     1     2
   Region
        1  1686     0
        2  2332  1810
        3  2564     0
        4  2070     0
        5  2184     0
        6  1829     0
        7  1550     0
```

Q13: Find the mean values of AOR variable corresponding to each label of Region variable.

d3_byRegion = d3.groupby('Region')

d3_byRegion.mean().transpose()

Output:



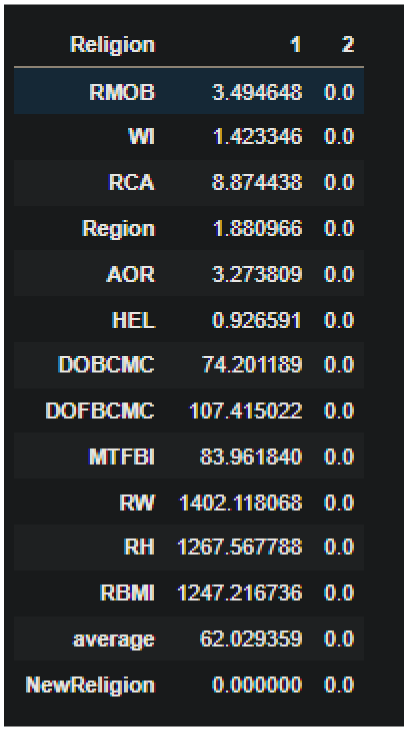
Region	1	2	3	4	5	6	7
RMOB	6.354686	4.507001	6.384165	6.560870	6.449176	6.478950	6.435484
WI	2.873665	2.725495	3.361544	3.221256	3.065934	2.671405	3.241935
RCA	31.672598	18.311685	31.759360	32.107246	32.120421	31.494806	31.887097
Religion	1.000000	1.436987	1.000000	1.000000	1.000000	1.000000	1.000000
AOR	17.781139	10.943747	17.880265	17.566667	17.607601	17.279388	18.572903
HEL	1.358244	1.579189	1.169267	1.276329	1.180861	1.137233	0.978710
DOBCMC	1255.581851	714.960406	1254.570203	1240.954589	1242.006868	1250.049754	1273.264516
DOFBCMC	1173.952550	667.447851	1175.865835	1167.916425	1166.230769	1171.449426	1181.184516
MTFBI	33.395611	19.965476	35.419657	32.613527	41.543498	35.294150	36.478710
RW	961.041518	414.558909	704.145086	664.336232	702.423077	515.057408	658.692258
RH	1946.447805	970.093433	1690.661076	1653.341063	1707.211538	1550.246583	1670.567742
RBMI	2545.765718	1340.294785	2346.479719	2348.993720	2355.123168	2159.464735	2268.927742
average	820.976671	467.457911	821.951898	813.828180	816.593712	818.931110	830.309247
NewReligion	1.000000	1.436987	1.000000	1.000000	1.000000	1.000000	1.000000

Q14: Find the variances of AOR variable corresponding to each label of Religion variable.

d3_byReligion = d3.groupby('Religion')

d3_byReligion.std().transpose()

Output:

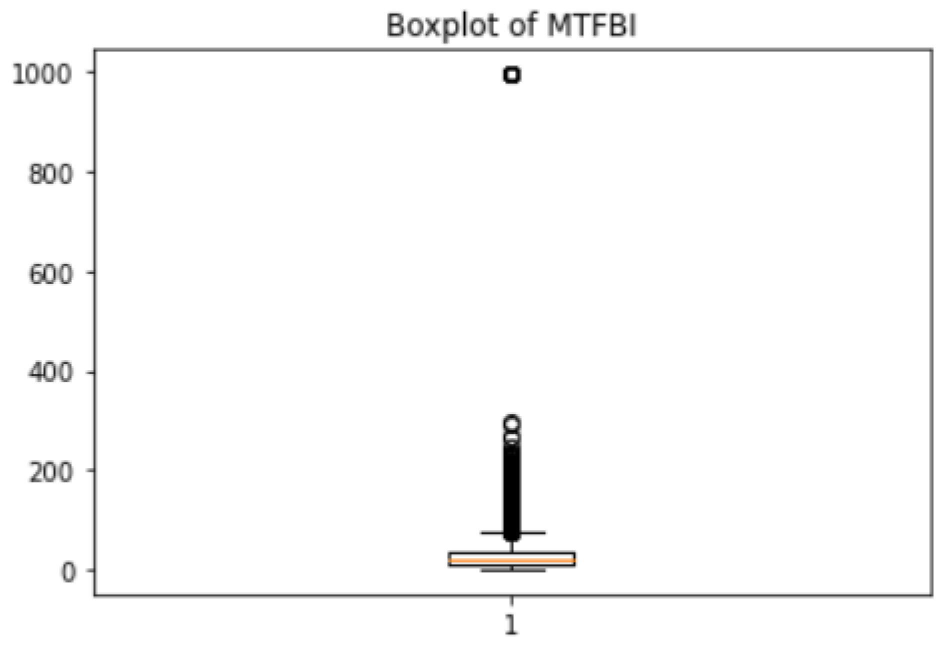


Religion	1	2
RMOB	3.494648	0.0
WI	1.423346	0.0
RCA	8.874438	0.0
Region	1.880966	0.0
AOR	3.273809	0.0
HEL	0.926591	0.0
DOBCMC	74.201189	0.0
DOFBCMC	107.415022	0.0
MTFBI	83.961840	0.0
RW	1402.118068	0.0
RH	1267.567788	0.0
RBMI	1247.216736	0.0
average	62.029359	0.0
NewReligion	0.000000	0.0

Q15: Draw a boxplot for the MTFBI variable.

```
import matplotlib.pyplot as plt

y = d3['MTFBI']

plt.boxplot(y)

plt.title("Boxplot of MTFBI")
```

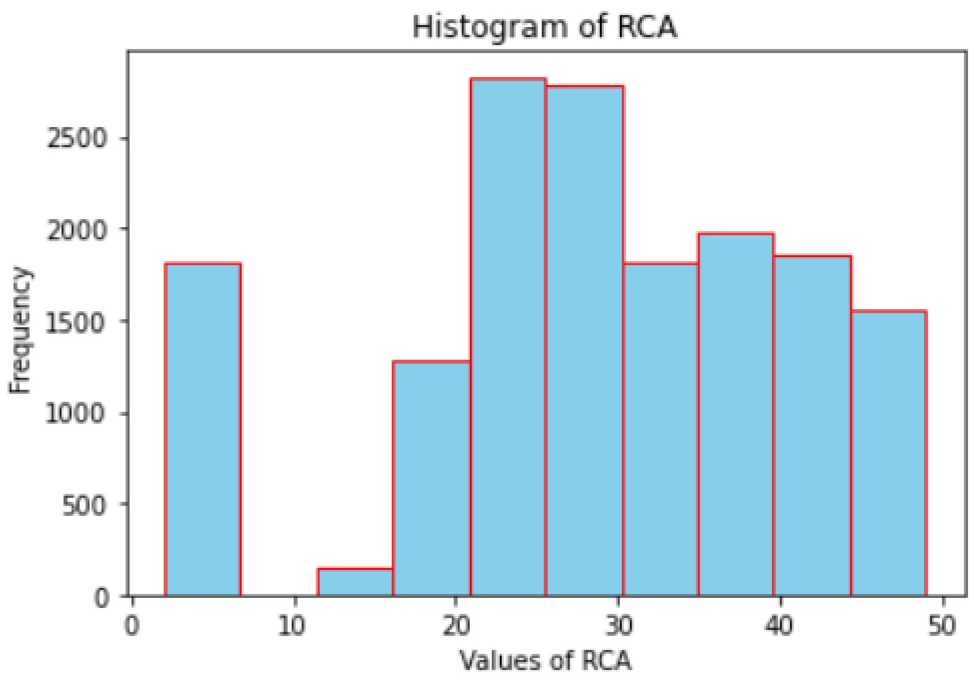
Output:



Q16: Draw a histogram for the RCA variable.

```
r = d3['RCA']

plt.hist(r, color = "skyblue", ec = "red")

plt.title("Histogram of RCA")

plt.xlabel("Values of RCA")

plt.ylabel("Frequency")
```

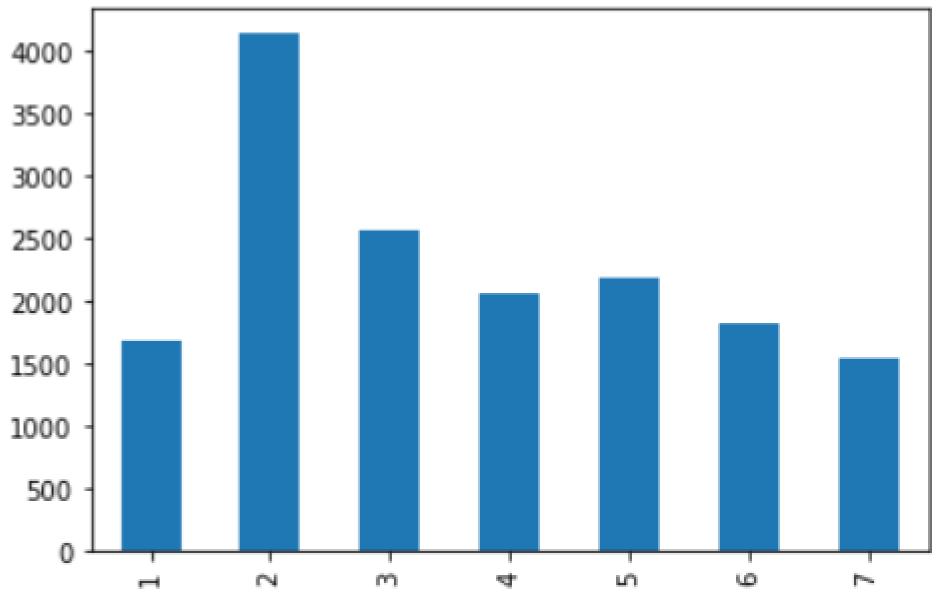
Output:



Q17: Draw a bar chart for the Region variable

d3['Region'].value_counts(sort = False).plot.bar()

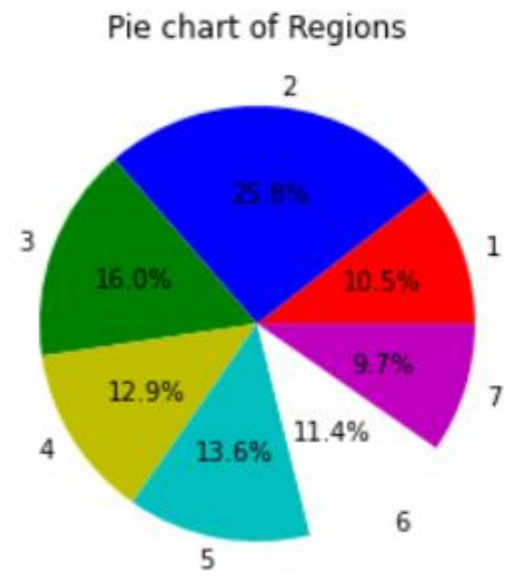
Output:



Q18: Draw a pie chart for the Region variable

```
labels = ['1', '2', '3', '4', '5', '6', '7']

cols = ['r', 'b', 'g', 'y', 'c', 'w', 'm']

sizes = ['1686', '4142', '2564', '2070', '2184', '1829', '1550']

plt.pie(sizes, explode = None, labels = labels, autopct = '%1.1f%%', colors = cols)

plt.title("Pie chart of Regions")

plt.show()
```

Output:



Q19: Put above four figures (question 15 to question 18) in a 2 by 2 grid

```
b = d3['MTFBI']

h = d3['RCA']

reg = d3['Region']

ba = d3['Region'].value_counts(sort = False)

labels = ['1', '2', '3', '4', '5', '6', '7']

cols = ['r', 'b', 'g', 'y', 'c', 'w', 'm']

sizes = ['1686', '4142', '2564', '2070', '2184', '1829', '1550']


plt.subplot(2,2,2)
```

plt.hist(h, color = "skyblue", ec = "red")

plt.subplot(2,2,1)

plt.boxplot(b)

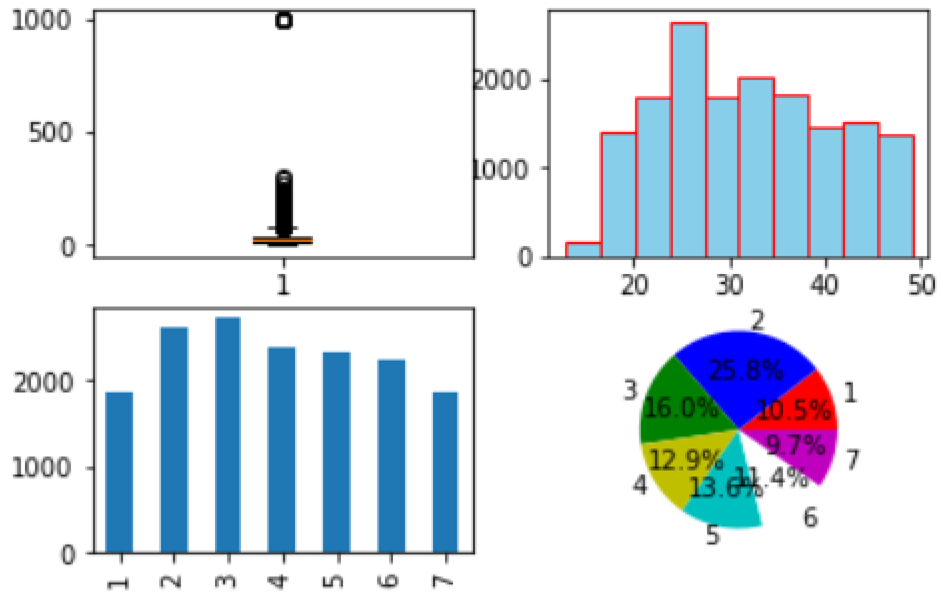
plt.subplot(2,2,3)

d3['Region'].value_counts(sort = False).plot.bar()

plt.subplot(2,2,4)

plt.pie(sizes, explode = None, labels = labels, autopct = '%1.1f%%', colors = cols)

Output:



Q20: Split the d3 data by WI variable and denote it by d4

d4 = d3.groupby('WI', as_index = False)

Output:

Q21: For each split data in d4 write a single loop to find the mean, minimum, maximum, standard deviation of MTFBI.

import statistics as ss

ds = [rows for _, rows in d4]

datalist = []

for i in range(len(ds)):

```
    datalist.append(pd.DataFrame({'WI' : [ss.mean(ds[i].WI)], 'MTFBI mean' : [ss.mean(ds[i].MTFBI)],
                    'MTFBI min' : [min(ds[i].MTFBI)], 'MTFBI Max' : [max(ds[i].MTFBI)],
                    'MTFBI Variance' : [ss.variance(ds[i].MTFBI)]
                    ,'MTFBI Median' : [ss.median(ds[i].MTFBI)]}))
new_data = pd.concat(datalist)
print(new_data)
```

Output:

```
WI  MTFBI mean  MTFBI min  MTFBI Max  MTFBI Variance  MTFBI Median
0   1   37.295088        0.0      996.0     7890.267841          22.0
0   2   21.975794        0.0      996.0     4433.464318           8.0
0   3   34.084646        0.0      996.0     6400.766659          22.0
0   4   34.814576        0.0      996.0     7217.257909          22.0
0   5   36.252577        0.0      996.0     6859.687171          23.0
```

Q22: Conduct a one sample mean test of hypothesis to check whether MTFBI has a mean of 30 or not.

```
import scipy.stats as st
st.stats.ttest_1samp(d3.MTFBI, 30)
```

Output: Ttest_1sampResult(statistic=2.8121680810452396, pvalue=0.004926859651115069)

Q23: Conduct a normality test of the MTFBI variable

```
import scipy
scipy.stats.shapiro(d3.MTFBI)
```

Output: ShapiroResult(statistic=0.24056196212768555, pvalue=0.0)

Q24: Check the equality of mean for MTFBI variable corresponding to two labels of "Newreligion" variable.

```
scipy.stats.ttest_ind(d3[d3.NewReligion == 1].MTFBI, d3[d3.NewReligion == 2].MTFBI)
```

Output: Ttest_indResult(statistic=17.0063732678908, pvalue=2.698791769280193e-64)

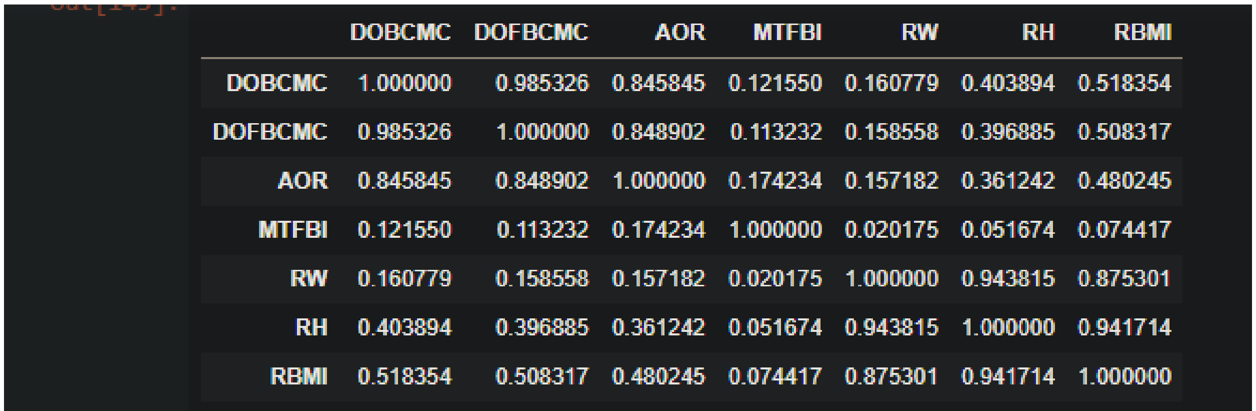
Q25: Find the correlation matrix of the variables DOBCMC, DOFBCMC, AOR, MTFBI, RW, RH and RBMI from the d3 data.

```
columns = ["DOBCMC","DOFBCMC","AOR","MTFBI","RW","RH", "RBMI"]
c1=d3[columns]
import matplotlib.pyplot as plt
plt.matshow(c1.corr())
```

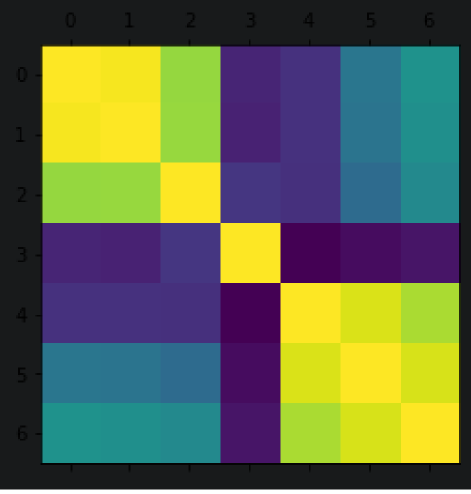
# correltion matrix

c1.corr()

Output:



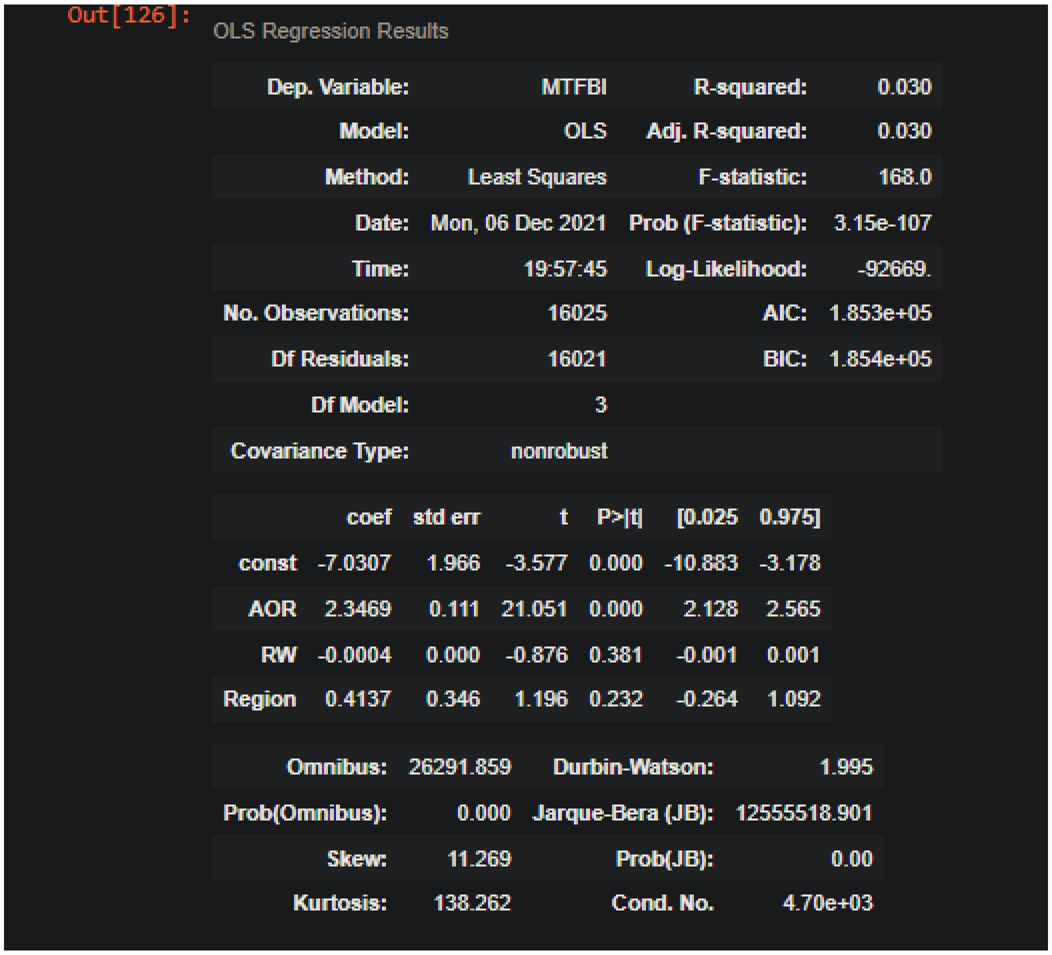
	DOBCMC	DOFBCMC	AOR	MTFBI	RW	RH	RBMI
DOBCMC	1.000000	0.985326	0.845845	0.121550	0.160779	0.403894	0.518354
DOFBCMC	0.985326	1.000000	0.848902	0.113232	0.158558	0.396885	0.508317
AOR	0.845845	0.848902	1.000000	0.174234	0.157182	0.361242	0.480245
MTFBI	0.121550	0.113232	0.174234	1.000000	0.020175	0.051674	0.074417
RW	0.160779	0.158558	0.157182	0.020175	1.000000	0.943815	0.875301
RH	0.403894	0.396885	0.361242	0.051674	0.943815	1.000000	0.941714
RBMI	0.518354	0.508317	0.480245	0.074417	0.875301	0.941714	1.000000



Q27: Fit a multiple regression model by considering MTFBI as dependent variable and AOR, RW, Region as independent variables

```
#q27

import statsmodels.tools as sm

from statsmodels.api import OLS

y=d3.MTFBI

x=d3[['AOR','RW','Region']]

x1=sm.add_constant(x)

model = OLS(y, x1).fit()

model.summary()
```

Output:



```
Out[126]:
OLS Regression Results

           Dep. Variable:            MTFBI          R-squared:      0.030
                  Model:              OLS      Adj. R-squared:      0.030
                 Method:    Least Squares         F-statistic:      168.0
                   Date:  Mon, 06 Dec 2021  Prob (F-statistic):  3.15e-107
                   Time:         19:57:45      Log-Likelihood:    -92669.
       No. Observations:            16025                 AIC:  1.853e+05
           Df Residuals:            16021                 BIC:  1.854e+05
               Df Model:                3
        Covariance Type:        nonrobust

             coef  std err       t   P>|t|   [0.025   0.975]
   const  -7.0307    1.966  -3.577   0.000  -10.883   -3.178
     AOR   2.3469    0.111  21.051   0.000    2.128    2.565
      RW  -0.0004    0.000  -0.876   0.381   -0.001    0.001
  Region   0.4137    0.346   1.196   0.232   -0.264    1.092

        Omnibus:  26291.859    Durbin-Watson:        1.995
 Prob(Omnibus):      0.000  Jarque-Bera (JB):  12555518.901
           Skew:     11.269         Prob(JB):         0.00
       Kurtosis:    138.262         Cond. No.     4.70e+03
```

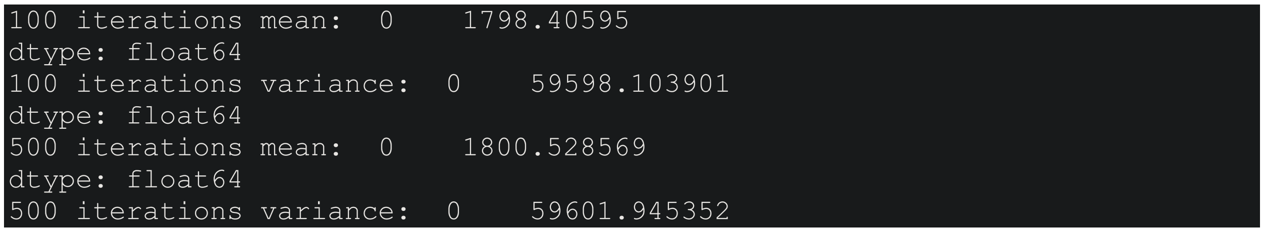
Q28 – 32: Simulate one data from the following equation y=50+10X+20U+100N+E. Where X is binomial with n=20, p=.70. U is uniform between 15 and 30 (inclusive). N is normal with mean 0 and standard deviation 5. E is random uniform between -1 and 1. True mean is 640. True variance is 257920. Repeat the procedure 100 times and check the true mean with the simulated mean. Repeat the procedure 100 times and check the true variance with the simulated variance. Repeat the procedure 500 times and check the true mean with the simulated mean. Repeat the procedure 500 times and check the true variance with the simulated variance.

```
import numpy as np

import pandas as pd

import scipy.stats as stats

import statsmodels.stats.api as sms

#q28

def sim(n):

   x = np.random.binomial(20,.70,n)

   u = np.random.normal(0,5,n)
```

```
    N = np.random.uniform(15,30,n)

    E = np.random.uniform(-1, 1, n)

    y= 50 + 10*x + 20 * u + 100 * N + E

    y1=pd.DataFrame(y)

    return y1


a = sim(1000)


# mean and variance

np.mean(a)

np.var(a)

stats.ttest_1samp(a,1000)

# repeating 100 times

B=100

repeat=[sim(1000) for i in range(B)]

alldata=pd.concat(repeat)

# computing mean and variance from simulated data

simulated_mean=np.mean(alldata)

simulated_variance=np.var(alldata)

# Theoretical Mean and Variance

theoretical_mean= 640

theoretical_variance= 257920

# difference between them

#29

print("100 iterations mean: " , abs(simulated_mean-theoretical_mean))

#30

print("100 iterations variance: " , abs(simulated_variance-theoretical_variance))


C=500
```

```
repeat=[sim(1000) for i in range(C)]

alldata=pd.concat(repeat)

# computing mean and variance from simulated data

simulated_mean=np.mean(alldata)

simulated_variance=np.var(alldata)

# Theoretical Mean and Variance

theoretical_mean= 640

theoretical_variance= 257920

# difference between them

#31

print("500 iterations mean: " , abs(simulated_mean-theoretical_mean))

#32

print("500 iterations variance: ", abs(simulated_variance-theoretical_variance))
```

Output:



```
100 iterations mean:  0    1798.40595
dtype: float64
100 iterations variance:  0    59598.103901
dtype: float64
500 iterations mean:  0    1800.528569
dtype: float64
500 iterations variance:  0    59601.945352
```

Q33: For five values of x=1:5, y=2:6, and z=3:7, compute 5 values for f(x)=e^x−log⁡(z^2)/ (5+y)

```
import math

def fun(x, y, z):

   return (math.exp(x) - math.log(z**2))/(5+y)


x = 1

y = 2

z = 3

for i in range(5):
   print(fun(x,y,z))
   x += 1
   y += 1
   z += 1
```

Output:

```
0.07443675016040364
0.5770584220863586
1.8740734553688299
5.1014631094688125
13.138303527678726
```

Q34: Solve the following system of linear equations: 70x+100y+40z=900; 120x+450y+340z=1000; 230x+230y+1230z=3000

```
a=np.array([[70,100,40],[120,450,340],[230,230,1230]])
b=np.array([900,1000,3000])
x=np.linalg.solve(a,b)
print(x)
```

Output:

```
[15.53852758 -1.8270015  -0.12491951]
```

Q35: Find the inverse of the following matrix: A=(20, 30, 30

20,80,120

40,90,360)

```
a = np.array([[20, 30, 30],
        [20, 80, 120],
        [40, 90, 360]])
print(np.linalg.inv(a))
```

Output:

```
[[ 0.07317073 -0.03292683  0.00487805]
 [-0.0097561   0.02439024 -0.00731707]
 [-0.00569106 -0.00243902  0.00406504]]
```

Q36: Suppose b=(10

20

30).

Then find (A'A)^−1 A'b. Here A’ means A transpose.

```
a_2 = a.transpose()

b = np.array([[10],

        [20],

        [30]])

v1 = np.linalg.inv(a_2.dot(a))

v2 = a_2.dot(b)

print(v1.dot(v2))
```

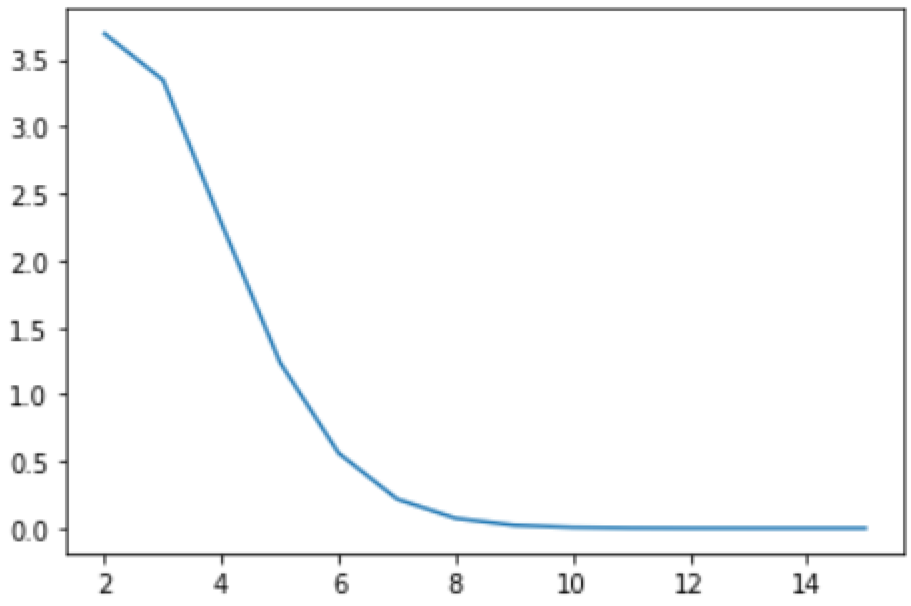
Output:

```
[[0.2195122 ]
 [0.17073171]
 [0.01626016]]
```

Q37: Draw the graph for the function f(x)= e^x/x! ;for2≤x≤15.

```
import numpy as np

from matplotlib import pyplot as plt

def f(x):

    return math.exp(x) / math.factorial(x)


x = [2,3,4,5,6,7,8,9,10,11,12,13,14,15]

y = []

for i in range(len(x)):

    y.append(f(x[i]))


plt.plot(x,y)

plt.show()
```

Output:



Q38: Draw the graph for the step functions Consider the continuous function

f(x)={2x^2+e^x+3; ifx<0

9x+log(20); if0≤x<10

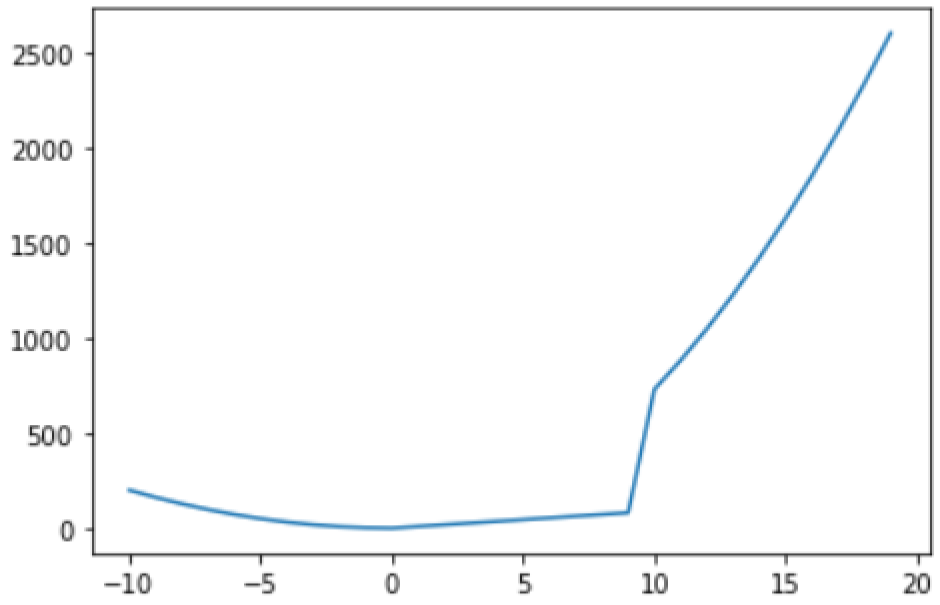
7x^2+5x−17;I f10≤x.

```
def g(x):
    out = 0
    if x < 0:
        out = 2 * (x**2) + math.exp(x) + 3
    elif x >= 0 and x < 10:
        out = 9 * x + math.log(20)
    else:
        out = 7 * (x**2) + 5 * x - 17
    return out


x = [i for i in range(-10, 20)]
y = []
for i in range(len(x)):
    y.append(g(x[i]))
```

plt.plot(x, y)

plt.show()

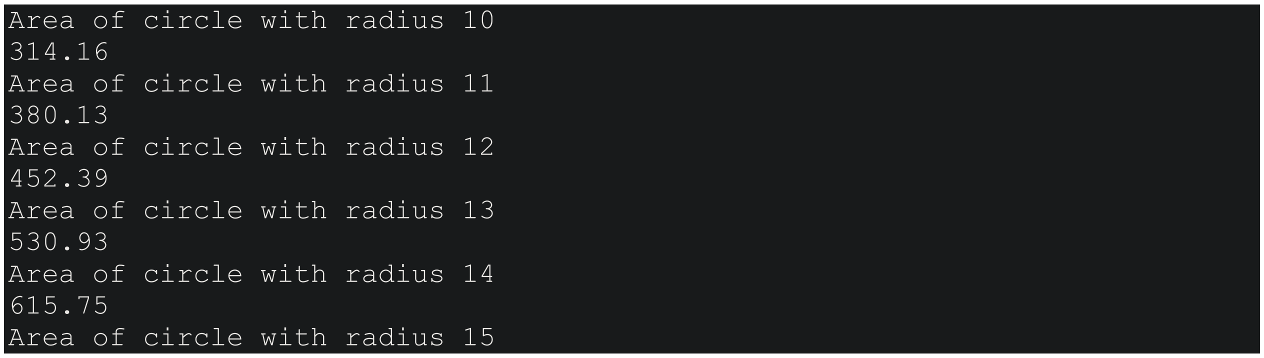
Output:



Q39: Find the areas of 10 circles, which have radii 10:19. The Area of a circle is given πr^2.

```
def circ_area(r):
    return math.pi * r**2


for i in range(10, 20):
    print("Area of circle with radius " + str(i))
    print(round(circ_area(i),2))
```

Output:



```
Area of circle with radius 10
314.16
Area of circle with radius 11
380.13
Area of circle with radius 12
452.39
Area of circle with radius 13
530.93
Area of circle with radius 14
615.75
Area of circle with radius 15
```

```
706.86
Area of circle with radius 16
804.25
Area of circle with radius 17
907.92
Area of circle with radius 18
1017.88
Area of circle with radius 19
1134.11
```

Q40: Find sum(1/logx) for x = 2 to 10000

```
def h(x):
    return 1/(math.log(x))


sum = 0
for i in range(2, 10000):
    sum += h(i)


print(sum)
```

Output:

```
1245.839690648031
```

Q41: Find sum(i^10/3+j) for i = 1 to 30 and j = 1 to 10

```
sum = 0
for i in range(30):
    for j in range(10):
        sum += (i ** 10)/ (3 + j)
print(sum)
```

Output:

```
2134775903297949.8
```

Q42: Compute the integral ∫ from 0 to positive infinity for equation: x^15 * e^−40xdx.

```
from scipy.integrate import quad
def f(x):
    return (x ** 15) * (math.exp(-(40 * x)))


res, err = quad(f, 0, math.inf)

print(res, err)
```

Output:

```
3.044666888955981e-14 2.766447362769 91e-14
```

Q43: Compute the integral ∫ from 0 to 1 for equation: x^150 * (1−x)^30dx

```
def g(x):

    return (x ** 150) * (math.pow(1-x,30))


res,err = quad(g, 0 ,1)

print(res, err)
```

Output:

```
4.167698831230213e-37 6.182522311694294e-37
```

Q44: For five values of x=1:5, y=2:6, and z=3:7, compute 5 values for f(x)=e^x−log(z^2)/ (5+y)

```
import math

def fun(x, y, z):

    return (math.exp(x) - math.log(z**2))/(5+y)


x = 1

y = 2

z = 3

for i in range(5):

    print(fun(x,y,z))

    x += 1

    y += 1

    z += 1
```

Output:

```
0.07443675016040364
0.5770584220863586
1.8740734553688299
5.1014631094688125
13.138303527678726
```

Q45: Solve the equation x^2−33x+1=0.

```
import sympy as sym

solution = sym.solve('x**2 - 33 * x + 1','x')

solution[0]
```

Output:

$$\frac{33}{2} - \frac{\sqrt{1085}}{2}$$

Q47: If $40 is invested today for 50 years with interest rate .10, the find the total amount of money in 50 years. The formula is p*(1+r)^t. p=40, t=50, and r=.10.

```
def interest(p, t, r):
    return p * ((1 + r) ** t)


print(interest(40, 50, 0.1))
```

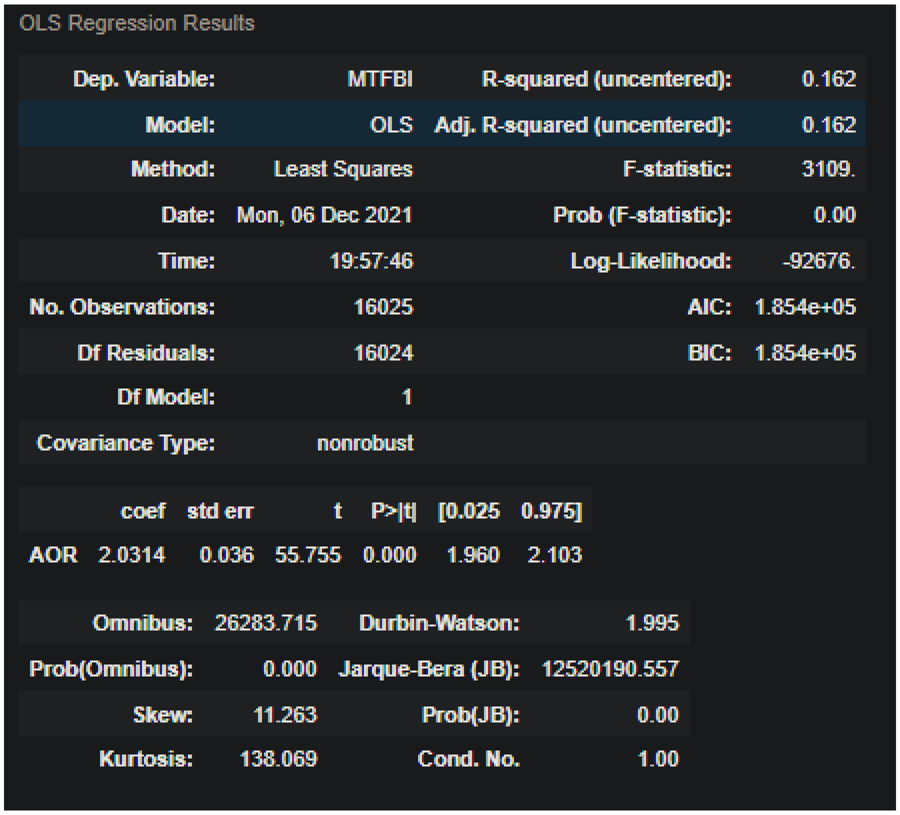
Output:

```
4695.634115187831
```

Q48: Fit a simple regression model by using MTFBI as dependent variable and AOR as independent variable.

```
import statsmodels.api as sm

model = sm.OLS(d3.MTFBI, d3.AOR).fit()

predictions = model.predict(d3.AOR)

model.summary()
```

Output:



```
OLS Regression Results
          Dep. Variable:            MTFBI        R-squared (uncentered):        0.162
                 Model:               OLS   Adj. R-squared (uncentered):        0.162
                Method:     Least Squares                   F-statistic:        3109.
                  Date:  Mon, 06 Dec 2021             Prob (F-statistic):        0.00
                  Time:          19:57:46                Log-Likelihood:      -92676.
     No. Observations:             16025                           AIC:    1.854e+05
          Df Residuals:            16024                           BIC:    1.854e+05
              Df Model:                 1
       Covariance Type:         nonrobust

        coef  std err       t  P>|t|  [0.025  0.975]
AOR   2.0314    0.036  55.755  0.000   1.960   2.103

       Omnibus:  26283.715    Durbin-Watson:        1.995
Prob(Omnibus):       0.000  Jarque-Bera (JB):  12520190.557
          Skew:     11.263         Prob(JB):         0.00
      Kurtosis:    138.069         Cond. No.         1.00
```

Q49: Check whether AOR and MTFBI are correlated or not.

```
import scipy

scipy.stats.pearsonr(d3.AOR, d3.MTFBI)
```

Output:

```
(0.1742337620135717, 1.9819270155456687e-109)
```

Q50: Check whether variance of AOR is 10 or not.

```
def chi_sq_test_for_variance(variable, h0):
    sample_variance = variable.var()
    n = variable.notnull().sum()
    degrees_of_freedom = n - 1
    x_sq_stat = (n-1) * sample_variance / h0
    p = stats.chi2.cdf(x_sq_stat, degrees_of_freedom)


    if p > 0.05:
        p = 1 - p
    return (x_sq_stat,p,degrees_of_freedom)


aor_variance = round(d3["AOR"].var(),2)
x_sq_stat,pval,dof = chi_sq_test_for_variance(d3["AOR"], h0 = 10)
print(round(x_sq_stat, 2),pval,dof)
```

Output:

```
55214.29 0.0 16024
```