# CS 584-04: Machine Learning

## Fall 2019: Assignment 1

Amitdeb Prasad Bhattacharya A20402789

Importing required libraries:

In [88]:
```
#importing required python libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading datasets:

In [90]:
```
#importing given datasets
fraud_dataset = pd.read_csv("C:\\Users\\Machine Learning\\Assignments & Projects\\Assignment 1\\Fraud.csv")
sample_dataset = pd.read_csv("C:\\Users\\Machine Learning\\Assignments & Projects\\Assignment 1\\NormalSample.csv")
```

Displaying few rows from top of Fraud dataset

In [91]:
```
#displaying top few rows of fraud dataset
print("Fraud Dataset:")
fraud_dataset.head()
```

Fraud Dataset:

Out[91]:

| | CASE_ID | FRAUD | TOTAL_SPEND | DOCTOR_VISITS | NUM_CLAIMS | MEMBER_DURATION | OPTOM_PRESC | NUM_MEMBERS |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1100 | 11 | 0 | 94 | 1 | 2 |
| 1 | 2 | 1 | 1300 | 7 | 2 | 122 | 0 | 1 |
| 2 | 3 | 1 | 1500 | 4 | 0 | 149 | 1 | 3 |
| 3 | 4 | 1 | 1500 | 9 | 0 | 180 | 1 | 1 |
| 4 | 5 | 0 | 1700 | 3 | 0 | 93 | 0 | 1 |

Displaying few rows from top of NormalSample dataset

In [86]:
```
#displaying top few rows of sample dataset
print("NormalSample Dataset:")
sample_dataset.head()
```

NormalSample Dataset:

Out[86]:

| | i | group | x |
|---|---|---|---|
| 0 | 0 | 1 | 29.8 |
| 1 | 1 | 1 | 33.3 |
| 2 | 2 | 0 | 30.9 |
| 3 | 3 | 1 | 32.2 |
| 4 | 4 | 0 | 31.1 |

## Question 1 (40 points)

**Write a Python program to calculate the density estimator of a histogram. Use the field *x* in the NormalSample.csv file.**

**a) (5 points) According to Izenman (1991) method, what is the recommended bin-width for the histogram of x?**

In [87]:
```
#checking the total length of the sample_dataset
print("Length of sample_dataset: {}".format(len(sample_dataset['x'])))
```

```
Length of sample_dataset: 1001
```

In [92]:
```
iqr = sample_dataset['x'].describe()[6] - sample_dataset['x'].describe()[4]
print("Interquartile range of X values: {}".format(iqr))
```

```
Interquartile range of X values: 2.0
```

In [93]:
```
h = 2*iqr*(np.power(len(sample_dataset['x']),-1/3))
print("According to Lzenmanh the recomended bin-width for histogram of x: {}".format(h))
```

```
According to Lzenmanh the recomended bin-width for histogram of x: 0.3998667554864774
```

In [94]:
```
u =np.log10(h)
v =np.sign(u)*np.ceil(np.abs(u))
bin_width = np.power(10,v)
print("Nice h value in Practice: {}".format(bin_width))
```

```
Nice h value in Practice: 0.1
```

**b) (5 points) What are the minimum and the maximum values of the field x?**

In [95]:
```
min_value = sample_dataset['x'].min()
print("Minimum value of field x: {}".format(min_value))
```

```
Minimum value of field x: 26.3
```

In [96]:
```
max_value = sample_dataset['x'].max()
print("Maximum Value of field x: {}".format(max_value))
```

```
Maximum Value of field x: 35.4
```

**c) (5 points) Let a be the largest integer less than the minimum value of the field x, and b be the smallest integer greater than the maximum value of the field x. What are the values of a and b?**

In [97]:
```
a = np.floor(min_value)
print("Largest integer less than the minimum value of the field x: {}".format(a))
```

```
Largest integer less than the minimum value of the field x: 26.0
```

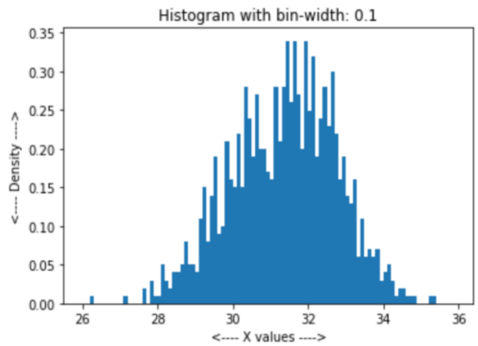
In [98]:
```
b = np.ceil(max_value)
print("The smallest integer greater than the maximum value of the field x : {}".format(b))
```

```
The smallest integer greater than the maximum value of the field x : 36.0
```

**d) (5 points) Use h = 0.1, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.**

```
In [99]:  h = 0.1
          minimum = 26
          maximum = 36
```

```
In [107]: #plotting a histogram with = 0.1
          mid_point = [e + 0.1/2 for e in np.arange(26,36,0.1)][:-1]
          density = plt.hist(x=sample_dataset['x'],bins=np.arange(26,36,0.1),density=True)
          plt.xlabel("<---- X values ---->")
          plt.ylabel("<---- Density ---->")
          plt.title("Histogram with bin-width: 0.1")
          plt.show()
```




```
In [108]: #storing the co-ordinates of density estimator
          co_ord = []
          for i in range(len(mid_point)):
              co_ord.append((np.round(mid_point[i],2),np.round(density[0][i],5)))
```

```
In [109]: #displaying first 10 co-ordinates
          print("Displaying First 20 Co-ordinates of Density estimator: ")
          co_ord[:20]
```

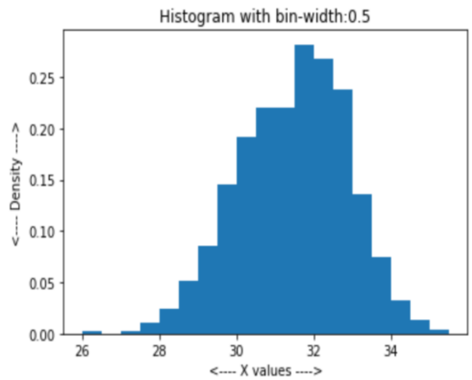
```
Displaying First 20 Co-ordinates of Density estimator:

Out[109]: [(26.05, 0.0),
           (26.15, 0.0),
           (26.25, 0.00999),
           (26.35, 0.0),
           (26.45, 0.0),
           (26.55, 0.0),
           (26.65, 0.0),
           (26.75, 0.0),
           (26.85, 0.0),
           (26.95, 0.0),
           (27.05, 0.0),
           (27.15, 0.00999),
           (27.25, 0.0),
           (27.35, 0.0),
           (27.45, 0.0),
           (27.55, 0.0),
           (27.65, 0.01998),
           (27.75, 0.0),
           (27.85, 0.02997),
           (27.95, 0.00999)]
```

**e) (5 points) Use h = 0.5, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.**

In [110]:

```python
#plotting a histogram with bin-width 0.5
h = 0.5
mid_point = [e + h/2 for e in np.arange(26,36,h)][:-1]
density = plt.hist(x=sample_dataset['x'],bins=np.arange(26,36,h),density=True)
plt.xlabel("<---- X values ---->")
plt.ylabel("<---- Density ---->")
plt.title("Histogram with bin-width:{}".format(h))
plt.show()
```




In [111]:

```python
#storing the co-ordinates of density estimator
co_ord = []
for i in range(len(mid_point)):
    co_ord.append((np.round(mid_point[i],2),np.round(density[0][i],5)))
```

In [112]:

```python
#displaying first few co-ordinates
print("Displaying First 20 Co-ordinates of Density estimator:")
co_ord[:20]
```

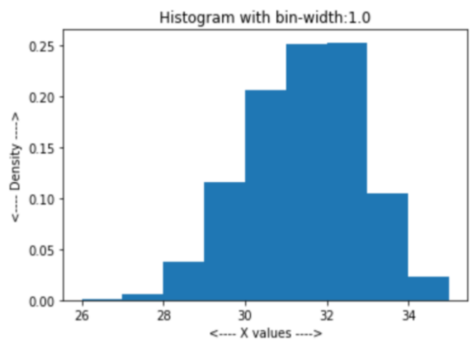
```
Displaying First 20 Co-ordinates of Density estimator:
```

Out[112]:

```
[(26.25, 0.002),
 (26.75, 0.0),
 (27.25, 0.002),
 (27.75, 0.00999),
 (28.25, 0.02398),
 (28.75, 0.05195),
 (29.25, 0.08591),
 (29.75, 0.14585),
 (30.25, 0.19181),
 (30.75, 0.21978),
 (31.25, 0.21978),
 (31.75, 0.28172),
 (32.25, 0.26773),
 (32.75, 0.23776),
 (33.25, 0.13586),
 (33.75, 0.07393),
 (34.25, 0.03197),
 (34.75, 0.01399),
 (35.25, 0.004)]
```

**f) (5 points) Use h = 1, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.**

```
In [113]:
#plotting a histogram with bin-width 1.0
h = 1.0
mid_point = [e + h/2 for e in np.arange(26,36,h)][:-1]
density = plt.hist(x=sample_dataset['x'],bins=np.arange(26,36,h),density=True)
plt.xlabel("<---- X values ---->")
plt.ylabel("<---- Density ---->")
plt.title("Histogram with bin-width:{}".format(h))
plt.show()
```




```
In [114]:
#storing the co-ordinates of density estimator
co_ord = []
for i in range(len(mid_point)):
    co_ord.append((np.round(mid_point[i],2),np.round(density[0][i],5)))
```

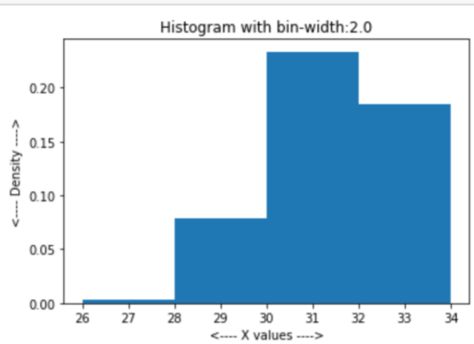
```
In [115]:
#displaying first few co-ordinates
print("Displaying First few Co-ordinates of Density estimator:" )
co_ord[:10]
```

```
Displaying First few Co-ordinates of Density estimator:

Out[115]: [(26.5, 0.001),
 (27.5, 0.00601),
 (28.5, 0.03804),
 (29.5, 0.11612),
 (30.5, 0.20621),
 (31.5, 0.25125),
 (32.5, 0.25325),
 (33.5, 0.10511),
 (34.5, 0.02302)]
```

**g) (5 points) Use h = 2, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.**

```
In [117]:
#plotting a histogram with bin-width 2.0
h = 2.0
mid_point = [e + h/2 for e in np.arange(26,36,h)][:-1]
density = plt.hist(x=sample_dataset['x'],bins=np.arange(26,36,h),density=True)
plt.xlabel("<---- X values ---->")
plt.ylabel("<---- Density ---->")
plt.title("Histogram with bin-width:{}".format(h))
plt.show()
```



In [118]:
```python
#storing the co-ordinates of density estimator
co_ord = []
for i in range(len(mid_point)):
    co_ord.append((np.round(mid_point[i],2),np.round(density[0][i],5)))
```

In [120]:
```python
#displaying first few co-ordinates
print("Displaying First few Co-ordinates of Density estimator: ")
co_ord
```

```
Displaying First few Co-ordinates of Density estimator:
```

Out[120]:
```
[(27.0, 0.00358), (29.0, 0.07865), (31.0, 0.2334), (33.0, 0.18437)]
```

**h) (5 points) Among the four histograms, which one, in your honest opinions, can best provide your insights into the shape and the spread of the distribution of the field x? Please state your arguments.**

**Ans:** Histogram with bin width 0.5 graph looks more uniform and we can easily interpret the highest point in graph, spread which includes mean, median and mode of the data very well though histogram with bin width 0.1 looks close to histogram with bin width 0.5. So histogram with bin-width of 0.5 can provide best insights.

## Question 2 (20 points)

**Use in the NormalSample.csv to generate box-plots for answering the following questions.**

**a) (5 points) What is the five-number summary of x? What are the values of the 1.5 IQR whiskers?**

In [163]:
```python
#five-number summary of x
print("Five number summary:\n{}".format(sample_dataset['x'].describe()))
```

```
Five number summary:
count    1001.000000
mean       31.414585
std         1.397672
min        26.300000
25%        30.400000
50%        31.500000
75%        32.400000
max        35.400000
Name: x, dtype: float64
```

In [164]:
```python
#Calculating the 1.5 IQR Whisker values
iqr = sample_dataset['x'].describe()[6] - sample_dataset['x'].describe()[4]
upper_limit = sample_dataset['x'].describe()[6] + 1.5*iqr
lower_limit = sample_dataset['x'].describe()[4] - 1.5*iqr
```

In [165]:
```python
print("Value of 1.5 IQR upperlimit: {} and lowerlimit: {}".format(upper_limit,lower_limit))
```

```
Value of 1.5 IQR upperlimit: 35.4 and lowerlimit: 27.4
```

**b) (5 points) What is the five-number summary of x for each category of the group? What are the values of the 1.5 IQR whiskers for each category of the group?**

In [166]:
```
#five-number summary of x for each category of the group
sample_dataset.groupby('group').describe()['x']
```

Out[166]:

| group | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| 0 | 315.0 | 30.004127 | 0.973935 | 26.3 | 29.4 | 30.0 | 30.6 | 32.2 |
| 1 | 686.0 | 32.062245 | 1.040236 | 29.1 | 31.4 | 32.1 | 32.7 | 35.4 |

In [167]:
```
#Calculating Q1,Q3,IQR of group 0
q1_0 = sample_dataset.groupby('group').describe()['x']['25%'][0]
q3_0 = sample_dataset.groupby('group').describe()['x']['75%'][0]
iqr_0 = round(q3_0 - q1_0,2)
iqr_0
```

Out[167]: 1.2

In [168]:
```
upper_limit_group0 = q3_0 + 1.5*iqr_0
lower_limit_group0 = round(q1_0 - 1.5*iqr_0,2)
print("Values of the 1.5 IQR whiskers for group 0 Upperlimit: {} and Lowerlimit: {}"
.format(upper_limit_group0,lower_limit_group0))
```

```
Values of the 1.5 IQR whiskers for group 0 Upperlimit: 32.4 and Lowerlimit: 27.6
```

In [169]:
```
#Calculating Q1,Q3,IQR of group 1
q1_1 = sample_dataset.groupby('group').describe()['x']['25%'][1]
q3_1 = sample_dataset.groupby('group').describe()['x']['75%'][1]
iqr_1 = round(q3_1 - q1_1,2)
iqr_1
```

Out[169]: 1.3

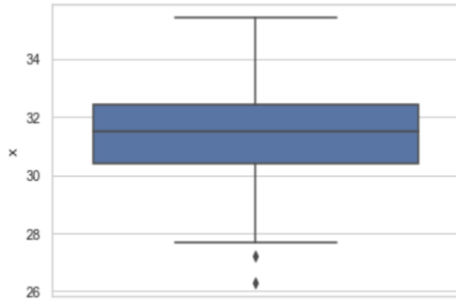
In [170]:
```
upper_limit_group1 = round(q3_1 + 1.5*iqr_1,2)
lower_limit_group1 = round(q1_1 - 1.5*iqr_1,2)
print("Values of the 1.5 IQR whiskers for group 1 Upperlimit: {} and LowerLimit:{}"
.format(upper_limit_group1,lower_limit_group1))
```

```
Values of the 1.5 IQR whiskers for group 1 Upperlimit: 34.65 and LowerLimit:29.45
```

Values of the 1.5 IQR whiskers for group 1 Upperlimit: 34.65 and LowerLimit:29.45

c) **(5 points) Draw a boxplot of x (without the group) using the Python boxplot function. Can you tell if the Python's boxplot has displayed the 1.5 IQR whiskers correctly?**

In [171]:
```
#Code to draw boxplot
sns.set(style="whitegrid")
sns.boxplot(sample_dataset['x'],orient="v")
plt.show()
```




In [172]:
```
upper_limit = sample_dataset['x'].describe()[6] + 1.5*iqr
lower_limit = sample_dataset['x'].describe()[4] - 1.5*iqr
```

In [173]:
```
#Checking the upper limit and lower limit values
(upper_limit,lower_limit)
```

Out[173]:
```
(35.4, 27.4)
```

Yes, python's boxplot has displayed 1.5 IQR correctly as we can clearly see from above values which we got from calculation.

**d) (5 points) Draw a graph where it contains the boxplot of x, the boxplot of x for each category of Group (i.e., three boxplots within the same graph frame). Use the 1.5 IQR whiskers, identify the outliers of x, if any, for the entire data and for each category of the group.**
***Hint: Consider using the CONCAT function in the PANDA module to append observations***

In [174]:
```
#Extracting x-values of group 0
grouped0 = sample_dataset.groupby('group').get_group(0)
```

In [175]:
```
#Extracting x-values of group 1
grouped1 = sample_dataset.groupby('group').get_group(1)
```

In [176]:
```
#Renaming x cloumn of group0 to x_group0
grouped0 = grouped0.rename(columns={'x':'x_group0'})
```

In [177]:
```
#Renaming x cloumn of group1 to x_group1
grouped1 = grouped1.rename(columns={'x':'x_group1'})
```

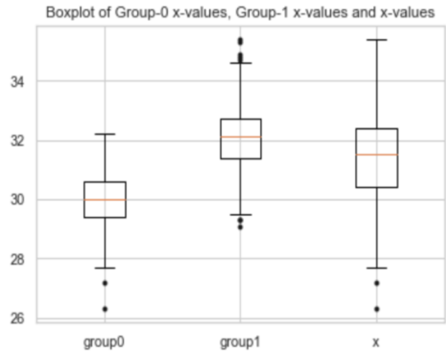
In [178]:
```
#Extracting x-values of group 0 to a list
group0 = grouped0['x_group0'].tolist()
```

In [179]:
```
#Extracting x-values of group 1 to a list
group1 = grouped1['x_group1'].tolist()
```

In [180]:
```
#Extracting x-values of both groups from original dataframe to a list
xvalues = sample_dataset['x'].tolist()
```

In [182]:

```
#Plotting the box plot of Group-0 x-values, Group-1 x-values and x-values
plt.boxplot([group0,group1,xvalues],labels=['group0','group1','x'],sym='k.')
plt.title("Boxplot of Group-0 x-values, Group-1 x-values and x-values")
plt.show()
```




For group-0 we have 2 outliers below lower whisker, for group-1 we have 4 outlier above upper whisker and below lower whisker and for only x-values we have 2 outliers below lower whisker.

## Question 3 (40 points)

**The data, FRAUD.csv, contains results of fraud investigations of 5,960 cases. The binary variable FRAUD indicates the result of a fraud investigation: 1 = Fraudulent, 0 = Otherwise. The other interval variables contain information about the cases.**

1. **TOTAL_SPEND: Total amount of claims in dollars**
2. **DOCTOR_VISITS: Number of visits to a doctor**
3. **NUM_CLAIMS: Number of claims made recently**
4. **MEMBER_DURATION: Membership duration in number of months**
5. **OPTOM_PRESC: Number of optical examinations**
6. **NUM_MEMBERS: Number of members covered**

**You are asked to use the Nearest Neighbors algorithm to predict the likelihood of fraud.**

a) **(5 points) What percent of investigations are found to be fraudulent? Please give your answer up to 4 decimal places.**

In [219]:

```
#displaying top few rows of dataset
fraud_dataset.head()
```

Out[219]:

| | CASE_ID | FRAUD | TOTAL_SPEND | DOCTOR_VISITS | NUM_CLAIMS | MEMBER_DURATION | OPTOM_PRESC | NUM_MEMBERS |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1100 | 11 | 0 | 94 | 1 | 2 |
| 1 | 2 | 1 | 1300 | 7 | 2 | 122 | 0 | 1 |
| 2 | 3 | 1 | 1500 | 4 | 0 | 149 | 1 | 3 |
| 3 | 4 | 1 | 1500 | 9 | 0 | 180 | 1 | 1 |
| 4 | 5 | 0 | 1700 | 3 | 0 | 93 | 0 | 1 |

In [220]:
```
#info of dataframe
fraud_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5960 entries, 0 to 5959
Data columns (total 8 columns):
CASE_ID            5960 non-null int64
FRAUD              5960 non-null int64
TOTAL_SPEND        5960 non-null int64
DOCTOR_VISITS      5960 non-null int64
NUM_CLAIMS         5960 non-null int64
MEMBER_DURATION    5960 non-null int64
OPTOM_PRESC        5960 non-null int64
NUM_MEMBERS        5960 non-null int64
dtypes: int64(8)
memory usage: 372.6 KB
```

In [221]:
```
print("The fraudulent investigations percentage: {}".format(round(np.mean(fraud_dataset['FRAUD'])*100,4)))
```

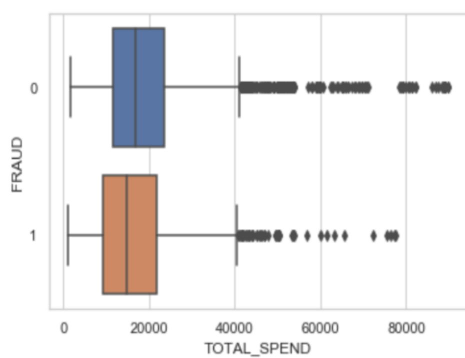
```
The fraudulent investigations percentage: 19.9497
```

The fraudulent investigation percentage is 19.9497

**b) (5 points) Use the BOXPLOT function to produce horizontal box-plots. For each interval variable, one box-plot for the fraudulent observations, and another box-plot for the non-fraudulent observations. These two box-plots must appear in the same graph for each interval variable.**

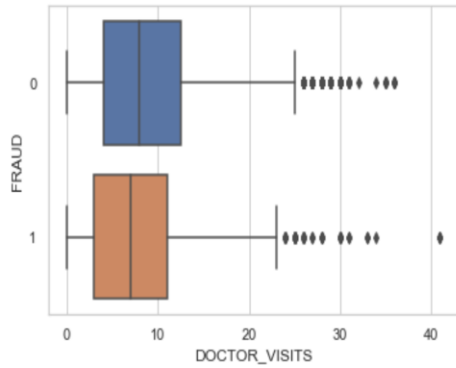
*Note: We are ignoring Case_Id, as it is unique field and is not considered as interval variables.*

In [222]:
```
#Fraud vs Total_spend
sns.boxplot(x="TOTAL_SPEND",y="FRAUD",orient='h',data=fraud_dataset)
plt.show()
```



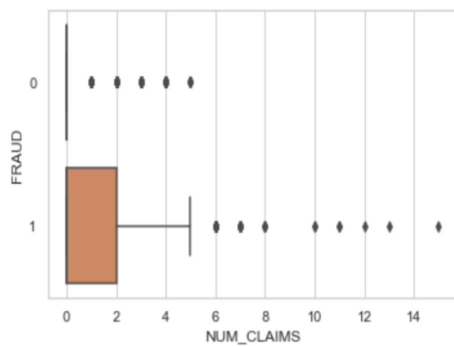
In [223]:

```python
#Fraud VS Doctor visits
sns.boxplot(x="DOCTOR_VISITS",y="FRAUD",orient='h',data=fraud_dataset)
plt.show()
```



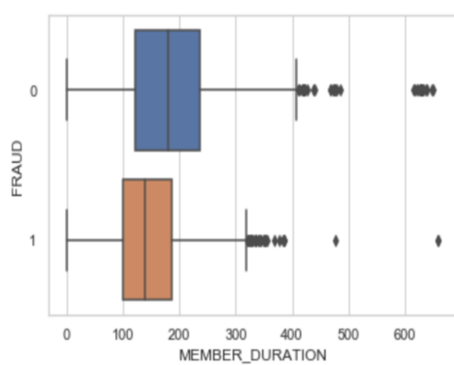

In [224]:

```python
#Fraud vs Num_claims
sns.boxplot(x="NUM_CLAIMS",y="FRAUD",orient='h',data=fraud_dataset)
plt.show()
```

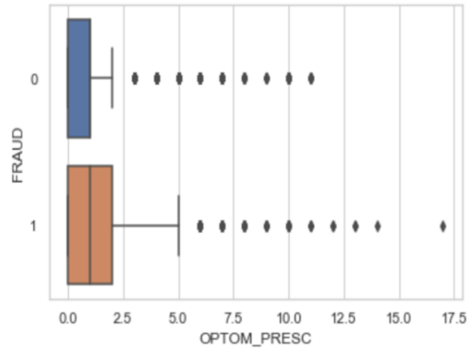



In [225]:

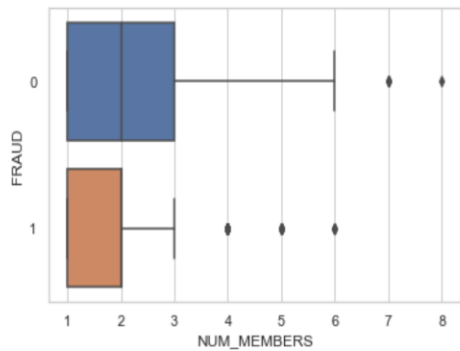
```python
#Fraud vs Member_Duration
sns.boxplot(x="MEMBER_DURATION",y="FRAUD",orient='h',data=fraud_dataset)
plt.show()
```



In [226]:
```python
#Fraud vs Optom_Presc
sns.boxplot(x="OPTOM_PRESC",y="FRAUD",orient='h',data=fraud_dataset)
plt.show()
```




In [227]:
```python
#Fraud vs Num_Memebers
sns.boxplot(x="NUM_MEMBERS",y="FRAUD",orient='h',data=fraud_dataset)
plt.show()
```




**c) (10 points) Orthonormalize interval variables and use the resulting variables for the nearest neighbor analysis. Use only the dimensions whose corresponding eigenvalues are greater than one.**

In [228]:
```python
#displaying top 5 rows of dataset
fraud_dataset.head()
```

Out[228]:

| | CASE_ID | FRAUD | TOTAL_SPEND | DOCTOR_VISITS | NUM_CLAIMS | MEMBER_DURATION | OPTOM_PRESC | NUM_MEMBERS |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1100 | 11 | 0 | 94 | 1 | 2 |
| **1** | 2 | 1 | 1300 | 7 | 2 | 122 | 0 | 1 |
| **2** | 3 | 1 | 1500 | 4 | 0 | 149 | 1 | 3 |
| **3** | 4 | 1 | 1500 | 9 | 0 | 180 | 1 | 1 |
| **4** | 5 | 0 | 1700 | 3 | 0 | 93 | 0 | 1 |

In [245]:
```
# Orthonormalizing interval variables from the dataframe
'''
- Ignoring case ID and fraud because fraud is our target variable and cases ID is not a interval variable.
- Extracting the fields from total spend to num_member converting it into a matrix and storing them in x.
'''
x = np.matrix(fraud_dataset.iloc[:,2:].values)
```

In [246]:
```
#checking the shape of x value
x.shape
```

Out[246]:
```
(5960, 6)
```

In [247]:
```
#multiplying x transpose with x
xtx = x.transpose() * x
```

In [248]:
```
#eigen value decomposition
eigen_value, eigen_vector = np.linalg.eigh(xtx)
```

In [250]:
```
print("Eigen values of x = \n", eigen_value)
print("Eigen vectors of x = \n",eigen_vector)
```

```
Eigen values of x =
 [6.84728061e+03 8.38798104e+03 1.80639631e+04 3.15839942e+05
 8.44539131e+07 2.81233324e+12]
Eigen vectors of x =
 [[-5.37750046e-06 -2.20900379e-05  3.62806809e-05 -1.36298664e-04
  -7.26453432e-03  9.99973603e-01]
 [ 6.05433402e-03 -2.69942162e-02  1.27528313e-02  9.99013423e-01
   3.23120126e-02  3.69879256e-04]
 [-9.82198935e-01  1.56454700e-01 -1.03312781e-01  1.14463687e-02
   1.62110700e-03  1.52596881e-05]
 [ 1.59310591e-04 -4.91894718e-03  3.11864824e-03 -3.25018102e-02
   9.99428355e-01  7.25592222e-03]
 [ 6.90939783e-02 -2.10615119e-01 -9.75101628e-01  6.26672294e-03
   2.19857585e-03  4.79234486e-05]
 [ 1.74569737e-01  9.64577791e-01 -1.95782843e-01  2.73038995e-02
   6.21788707e-03  7.82430481e-05]]
```

In [251]:
```
#checking if all interval variable eigen values are greater than 1
eigen_value > 1
```

Out[251]:
```
array([ True,  True,  True,  True,  True,  True])
```

i. **(5 points) How many dimensions are used?**

Ans: Total 6 dimensions are used i.e one for each feature variable ("TOTAL_SPEND", "DOCTOR_VISITS", "NUM_CLAIMS", "MEMBER_DURATION", "OPTOM_PRESC", "NUM_MEMBERS")

ii. **(5 points) Please provide the transformation matrix? You must provide proof that the resulting variables are actually orthonormal.**

In [253]:
```python
#Transformation matrix
transformation = eigen_vector * np.linalg.inv(np.sqrt(np.diagflat(eigen_value)));
print("Transformation Matrix = \n", transformation)
```

```
Transformation Matrix =
 [[-6.49862374e-08 -2.41194689e-07  2.69941036e-07 -2.42525871e-07
  -7.90492750e-07  5.96286732e-07]
 [ 7.31656633e-05 -2.94741983e-04  9.48855536e-05  1.77761538e-03
   3.51604254e-06  2.20559915e-10]
 [-1.18697179e-02  1.70828329e-03 -7.68683456e-04  2.03673350e-05
   1.76401304e-07  9.09938972e-12]
 [ 1.92524315e-06 -5.37085514e-05  2.32038406e-05 -5.78327741e-05
   1.08753133e-04  4.32672436e-09]
 [ 8.34989734e-04 -2.29964514e-03 -7.25509934e-03  1.11508242e-05
   2.39238772e-07  2.85768709e-11]
 [ 2.10964750e-03  1.05319439e-02 -1.45669326e-03  4.85837631e-05
   6.76601477e-07  4.66565230e-11]]
```

In [255]:
```python
# Here is the transformed X
transformation_x = x * transformation;
print("The Transformed x = \n", transformation_x)
```

```
The Transformed x =
 [[ 5.96859502e-03  1.02081629e-02 -6.64664861e-03  1.39590283e-02
   9.39352141e-03  6.56324665e-04]
 [-2.09672310e-02  5.01932025e-03  8.51930607e-04  5.16174400e-03
   1.22658834e-02  7.75702220e-04]
 [ 7.64597676e-03  1.97528525e-02 -7.38335310e-03 -1.71350853e-03
   1.50348109e-02  8.95075830e-04]
 ...
 [-7.18408819e-05 -1.62580211e-02  2.75078514e-02 -7.13245766e-03
  -4.74021952e-02  5.31896971e-02]
 [-1.80147801e-04 -1.62154130e-02  2.76213381e-02 -9.17125411e-03
  -4.76625006e-02  5.35474776e-02]
 [-2.21157680e-03 -2.73884697e-02  2.93391341e-02 -7.81347172e-03
  -4.70861917e-02  5.36071324e-02]]
```

In [260]:
```python
# Proof to check resulting varaibles are othonormal
xtx = transformation_x.transpose() * transformation_x;
print("Expect an Identity Matrix = \n", xtx)
```

```
Expect an Identity Matrix =
 [[ 1.00000000e+00 -2.99781901e-16 -4.56882795e-16  5.45884952e-15
   1.20129601e-15 -1.27176915e-16]
 [-2.99781901e-16  1.00000000e+00 -6.56592836e-16 -2.76891140e-14
  -1.22818422e-15  7.71951947e-16]
 [-4.56882795e-16 -6.56592836e-16  1.00000000e+00  3.50132250e-15
   1.14491749e-16 -2.32452946e-16]
 [ 5.45884952e-15 -2.76891140e-14  3.50132250e-15  1.00000000e+00
   1.14821347e-14 -3.47768689e-15]
 [ 1.20129601e-15 -1.22818422e-15  1.14491749e-16  1.14821347e-14
   1.00000000e+00 -6.27969898e-16]
 [-1.27176915e-16  7.71951947e-16 -2.32452946e-16 -3.47768689e-15
  -6.27969898e-16  1.00000000e+00]]
```

In [261]:
```python
#Checking whether 1.00000000e+00 is equal to 1
1.00000000e+00 == 1
```

Out[261]:
```
True
```

The result variables are orthonormal.

**d) (10 points) Use the NearestNeighbors module to execute the Nearest Neighbors algorithm using exactly <u>five</u> neighbors and the resulting variables you have chosen in c). The KNeighborsClassifier module has a score function.**

```
In [267]: #Loading the NearestNeighbors library
          from sklearn.neighbors import KNeighborsClassifier
```

```
In [268]: #Extracting target values from fraud dataframe
          target = fraud_dataset.iloc[:,1]
```

```
In [269]: #implementing the Nearest neighbours algorithm by using 5 neighbours
          neigh = KNeighborsClassifier(n_neighbors=5,algorithm = 'brute', metric = 'euclidean')
```

```
In [270]: #training the model
          nbrs = neigh.fit(transformation_x,target)
```

i. **(5 points) Run the score function, provide the function return value**

```
In [276]: #result using score function
          print("Score function result: {}".format(nbrs.score(transformation_x,target)))

          Score function result: 0.8778523489932886
```

Score function result: 0.8778523489932886

ii. **(5 points) Explain the meaning of the score function return value.**

Ans: Scores in classification algorithms represents accuracy of given dataset which means that our model has 87% accuracy on training data.

**e) (5 points) For the observation which has these input variable values: TOTAL_SPEND = 7500, DOCTOR_VISITS = 15, NUM_CLAIMS = 3, MEMBER_DURATION = 127, OPTOM_PRESC = 2, and NUM_MEMBERS = 2, find its five neighbors. Please list their input variable values and the target values. *Reminder: transform the input observation using the results in c) before finding the neighbors*.**

```
In [283]: #Transforming i/p obsevations as scaling is required for accuracy for KNN algorithm.
          focal = [[7500, 15, 3, 127, 2, 2]]
          trans_focal = focal * transformation
```

```
In [284]: test_set = nbrs.kneighbors(trans_focal,n_neighbors = 5)
          print("Distance values: {} \nNearest Neighbours: {}".format(test_set[0],test_set[1]))

          Distance values: [[0.         0.01045716 0.01206995 0.0121862  0.0140379 ]]
          Nearest Neighbours: [[ 588 2897 1199 1246  886]]
```

We got two output arrays among them 1st array gives distance value and 2nd one gives nearest neighbors element in trained dataset.

**f) (5 points) Follow-up with e), what is the predicted probability of fraudulent (i.e., FRAUD = 1)? If your predicted probability is greater than or equal to your answer in a), then the observation will be classified as fraudulent. Otherwise, non-fraudulent. Based on this criterion, will this observation be misclassified?**

```
In [291]: #Prediction label of test data
          print("Predicted label of KNN for Test data:{}".format(nbrs.predict(trans_focal)))

          Predicted label of KNN for Test data:[1]
```

In [294]:
```
#probability of testdata being class 0 or class 1
print("Checking the probability values of test set: {}".format(nbrs.predict_proba(trans_focal)))
```

```
Checking the probability values of test set: [[0. 1.]]
```

It has predicted probability of being class 1 as 1.0 hence it is Fraud since 1 >= 0.19 the observation is fraudulent and it’s not misclassified.