In [1]:
```
import math
import numpy as np
from numpy.random import normal as normal
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
import matplotlib.animation as animation
import seaborn as sns
from scipy import stats as sts
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
pd.options.display.float_format = '{:.5f}'.format
```

In [2]:
```
df = pd.read_csv("Inc_Exp_Data.csv")
df
```

Out[2]:

	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_Amt	Annual_HH_Ir
0	5000	8000	3	2000	
1	6000	7000	2	3000	
2	10000	4500	2	0	1
3	10000	2000	1	0	
4	12500	12000	2	3000	1
5	14000	8000	2	0	1
6	15000	16000	3	35000	1
7	18000	20000	5	8000	2
8	19000	9000	2	0	2
9	20000	9000	4	0	2
10	20000	18000	4	8000	2
11	22000	25000	6	12000	2
12	23400	5000	3	0	2
13	24000	10500	6	0	3
14	24000	10000	4	0	2
15	25000	12300	3	0	2
16	25000	20000	3	3500	2
17	25000	10000	6	0	2
18	29000	6600	2	2000	3
19	30000	13000	4	0	3
20	30500	25000	5	5000	3
21	32000	15000	4	0	4
22	34000	19000	6	0	3
23	34000	25000	3	4000	4
24	35000	12000	3	0	4
25	35000	25000	4	0	4
26	39000	8000	4	0	5
27	40000	10000	4	0	4
28	42000	15000	4	0	4
29	43000	12000	4	0	6
30	45000	25000	6	0	5
31	45000	40000	6	3500	5
32	45000	10000	2	1000	4

	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_Amt	Annual_HH_In
33	45000	22000	4	2500	6
34	46000	25000	5	3500	5
35	47000	15000	7	0	4
36	50000	20000	4	0	5
37	50500	20000	3	0	5
38	55000	45000	6	12000	6
39	60000	10000	3	0	5
40	60000	50000	6	10000	5
41	65000	20000	4	5000	6
42	70000	9000	2	0	7
43	80000	20000	4	0	10
44	85000	25000	5	0	11
45	90000	48000	7	0	8
46	98000	25000	5	0	11
47	100000	30000	6	0	14
48	100000	50000	4	20000	10
49	100000	40000	6	10000	13

In [3]:
```
df["Inc_After_Exp"] = df["Mthly_HH_Income"] - df["Mthly_HH_Expense"]
df["Qualification_Num"] = df["Highest_Qualified_Member"]
num_values = {
    "Illiterate" : "0",
    "Under-Graduate" : "1",
    "Graduate" : "2",
    "Post-Graduate" : "3",
    "Professional" : "4"
}
for i in range(len(df["Qualification_Num"])):
    df["Qualification_Num"][i] = num_values[df["Qualification_Num"][i]]
df = df.astype({'Qualification_Num':'int64'})
df
```

```
C:\Users\test\AppData\Local\Temp\ipykernel_11936\2876986243.py:11: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  df["Qualification_Num"][i] = num_values[df["Qualification_Num"][i]]
```

Out[3]:

	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_Amt	Annual_HH_Incor
0	5000	8000	3	2000	642
1	6000	7000	2	3000	799
2	10000	4500	2	0	1128
3	10000	2000	1	0	972
4	12500	12000	2	3000	1470
5	14000	8000	2	0	1965
6	15000	16000	3	35000	1674
7	18000	20000	5	8000	2160
8	19000	9000	2	0	2188
9	20000	9000	4	0	2208
10	20000	18000	4	8000	2784
11	22000	25000	6	12000	2798
12	23400	5000	3	0	2920
13	24000	10500	6	0	3168
14	24000	10000	4	0	2448
15	25000	12300	3	0	2460
16	25000	20000	3	3500	2610
17	25000	10000	6	0	2580
18	29000	6600	2	2000	3480
19	30000	13000	4	0	3852

	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_Amt	Annual_HH_Incor
20	30500	25000	5	5000	3513
21	32000	15000	4	0	4454
22	34000	19000	6	0	3304
23	34000	25000	3	4000	4692
24	35000	12000	3	0	4662
25	35000	25000	4	0	4494
26	39000	8000	4	0	5569
27	40000	10000	4	0	4128
28	42000	15000	4	0	4888
29	43000	12000	4	0	6192
30	45000	25000	6	0	5238
31	45000	40000	6	3500	5076
32	45000	10000	2	1000	4374
33	45000	22000	4	2500	6102
34	46000	25000	5	3500	5961
35	47000	15000	7	0	4568
36	50000	20000	4	0	5700
37	50500	20000	3	0	5817
38	55000	45000	6	12000	6006
39	60000	10000	3	0	5904
40	60000	50000	6	10000	5904
41	65000	20000	4	5000	6474
42	70000	9000	2	0	7560
43	80000	20000	4	0	10752
44	85000	25000	5	0	11424
45	90000	48000	7	0	8856
46	98000	25000	5	0	11524
47	100000	30000	6	0	14040
48	100000	50000	4	20000	10320
49	100000	40000	6	10000	13200

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Mthly_HH_Income           50 non-null     int64
 1   Mthly_HH_Expense          50 non-null     int64
 2   No_of_Fly_Members         50 non-null     int64
 3   Emi_or_Rent_Amt           50 non-null     int64
 4   Annual_HH_Income          50 non-null     int64
 5   Highest_Qualified_Member  50 non-null     object
 6   No_of_Earning_Members     50 non-null     int64
 7   Inc_After_Exp             50 non-null     int64
 8   Qualification_Num         50 non-null     int64
dtypes: int64(8), object(1)
memory usage: 3.6+ KB
```

In [6]:
```
df.shape
```

Out[6]: (50, 9)

In [7]:
```
numdf = df.select_dtypes(include = ["int64"])
numdf.head()
```

Out[7]:

	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_Amt	Annual_HH_Incom
0	5000	8000	3	2000	6420
1	6000	7000	2	3000	7992
2	10000	4500	2	0	11280
3	10000	2000	1	0	9720
4	12500	12000	2	3000	14700

In [8]:
```
stats = numdf.describe()
stats.loc["iqr"] = list(stats.loc["75%"] - stats.loc["25%"])
stats.loc["mode"] = list(numdf.mode().loc[0])
stats.loc["range"] = list(numdf.apply(lambda x: x.max() - x.min()))
stats.loc["var"] = list(stats.loc["std"].apply(lambda x: x**2))
stats.loc["mad"] = list(numdf.mad())
stats
```

Out[8]:

	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_Amt	Annual_HH_I
count	50.00000	50.00000	50.00000	50.00000	50
mean	41558.00000	18818.00000	4.06000	3060.00000	490019
std	26097.90898	12090.21682	1.51738	6241.43495	320135
min	5000.00000	2000.00000	1.00000	0.00000	64200
25%	23550.00000	10000.00000	3.00000	0.00000	258750
50%	35000.00000	15500.00000	4.00000	0.00000	447420
75%	50375.00000	25000.00000	5.00000	3500.00000	594720
max	100000.00000	50000.00000	7.00000	35000.00000	1404000
iqr	26825.00000	15000.00000	2.00000	3500.00000	335970
mode	45000.00000	25000.00000	4.00000	0.00000	590400
range	95000.00000	48000.00000	6.00000	35000.00000	1339800
var	681100853.06122	146173342.85714	2.30245	38955510.20408	102486925397
mad	20288.96000	9247.44000	1.19920	3866.40000	238469

In [9]:
```
stats.cov()
```

Out[9]:

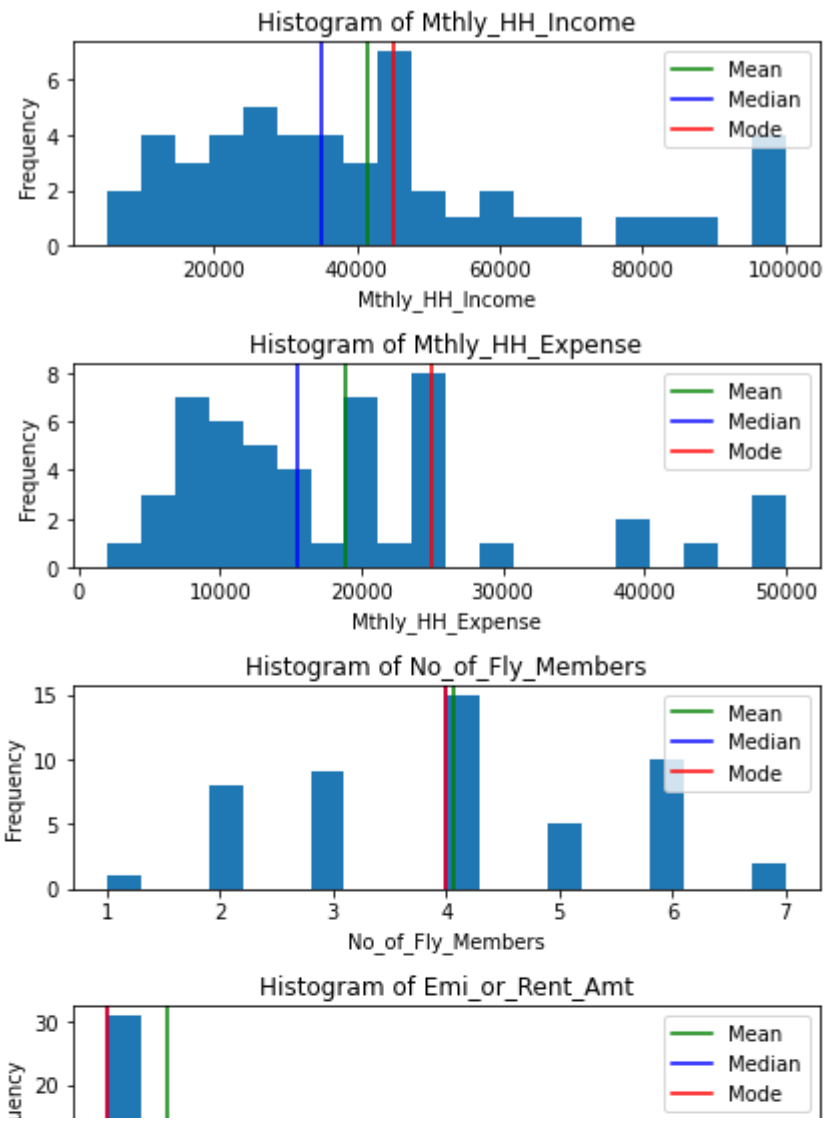
	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Membe
Mthly_HH_Income	35680398050115344.00000	7656922369684353.00000	-267060364.559
Mthly_HH_Expense	7656922369684353.00000	1643156079505701.25000	-57338864.797
No_of_Fly_Members	-267060364.55961	-57338864.79739	170.284
Emi_or_Rent_Amt	2040461037668922.25000	437877813348610.06250	-15279655.207
Annual_HH_Income	5369198756433674240.00000	1152216316104727040.00000	-40172830775.834
No_of_Earning_Members	-263525971.75815	-56588360.45132	175.601
Inc_After_Exp	21875531308876448.00000	4694433235706356.00000	-163730736.196
Qualification_Num	-235384000.44812	-50547070.38775	173.976

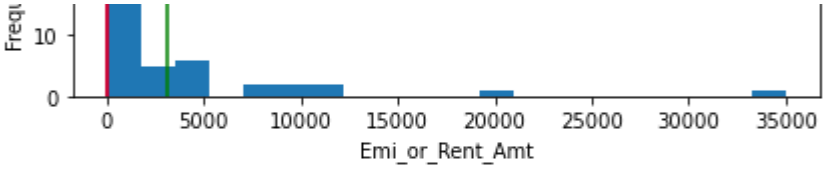
In [10]:
```
stats.corr()
```

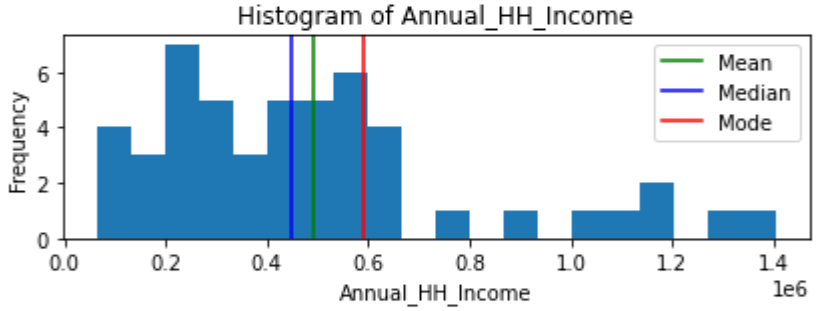
Out[10]:

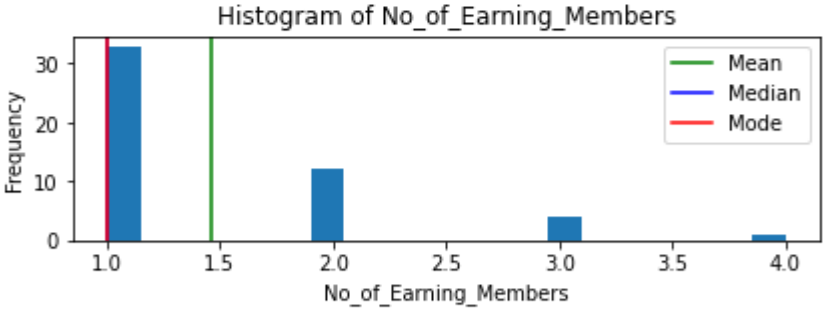
	Mthly_HH_Income	Mthly_HH_Expense	No_of_Fly_Members	Emi_or_Rent_A
Mthly_HH_Income	1.00000	1.00000	-0.10834	1.00
Mthly_HH_Expense	1.00000	1.00000	-0.10840	1.00
No_of_Fly_Members	-0.10834	-0.10840	1.00000	-0.10
Emi_or_Rent_Amt	1.00000	1.00000	-0.10840	1.00
Annual_HH_Income	1.00000	1.00000	-0.10831	1.00
No_of_Earning_Members	-0.10331	-0.10337	0.99645	-0.10
Inc_After_Exp	1.00000	1.00000	-0.10834	1.00
Qualification_Num	-0.09323	-0.09330	0.99750	-0.09

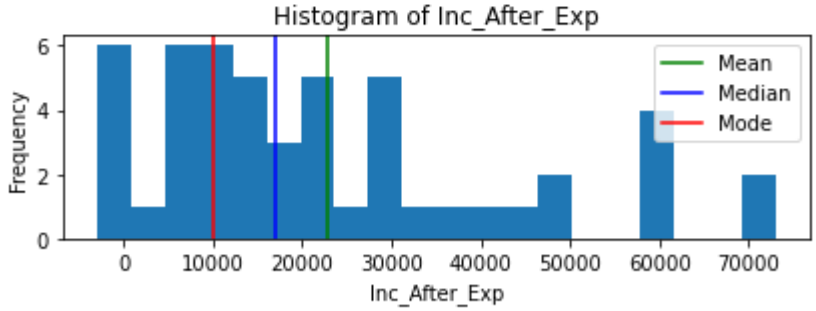
In [11]:
```
fig, axes = plt.subplots(nrows = len(numdf.columns), ncols = 1, figsize = (6, 18)

i = 0
for ax in axes:
    mean = numdf[numdf.columns[i]].mean()
    median = numdf[numdf.columns[i]].median()
    mode = numdf[numdf.columns[i]].mode()

    ax.hist(numdf[numdf.columns[i]], bins = 20)

    mean_line = ax.axvline(mean, color='green', label='Mean')
    median_line = ax.axvline(median, color='blue', label='Median')
    mode_line = ax.axvline(mode[0], color='red', label='Mode')

    ax.legend(handles=[mean_line, median_line, mode_line])

    ax.set_xlabel(numdf.columns[i])
    ax.set_ylabel("Frequency")
    ax.set_title("Histogram of " + numdf.columns[i])
    i = i + 1

plt.tight_layout()
plt.show()
```

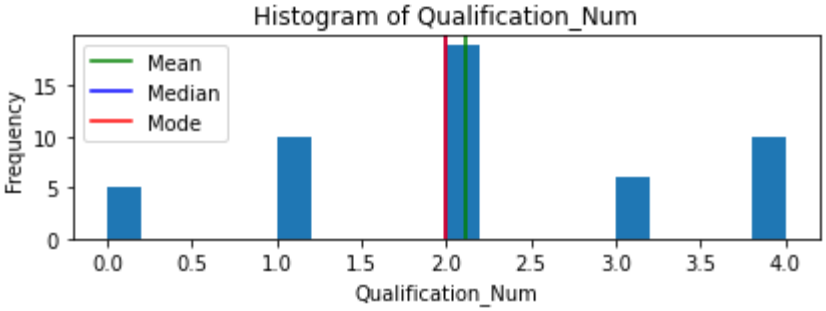



Freq
10
0
0
5000
10000
15000
20000
25000
30000
35000
Emi_or_Rent_Amt

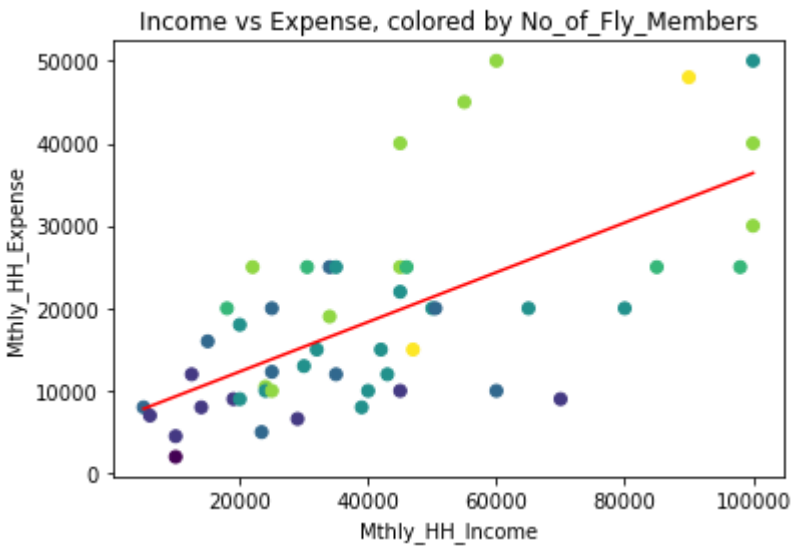

Histogram of Annual_HH_Income
Mean
Median
Mode
Frequency
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
Annual_HH_Income


Histogram of No_of_Earning_Members
Mean
Median
Mode
Frequency
30
20
10
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
No_of_Earning_Members


Histogram of Inc_After_Exp
Mean
Median
Mode
Frequency
6
4
2
0
0
10000
20000
30000
40000
50000
60000
70000
Inc_After_Exp


Histogram of Qualification_Num
Mean
Median
Mode
Frequency
15
10
5
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Qualification_Num

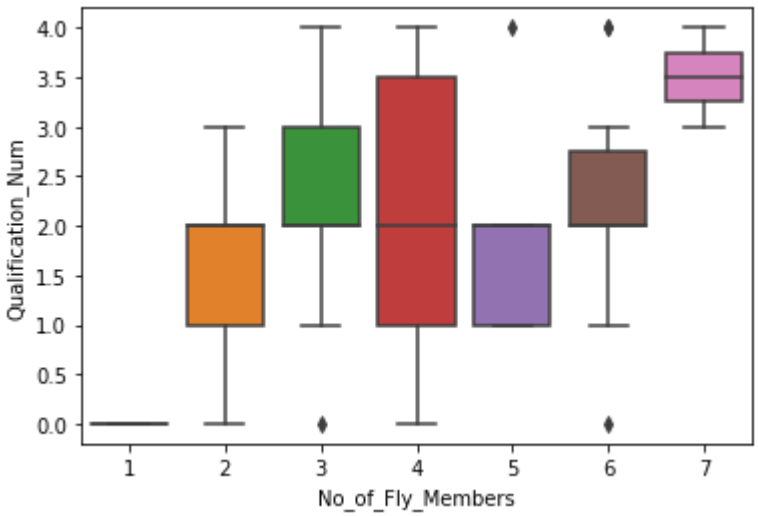
In [13]:
```
plt.scatter(df["Mthly_HH_Income"], df["Mthly_HH_Expense"], c=df["No_of_Fly_Member
gradient, intercept, r_value, p_value, std_err = sts.linregress(df["Mthly_HH_Inco
x1 = np.linspace(np.min(df["Mthly_HH_Income"]),np.max(df["Mthly_HH_Income"]),500)
y1 = gradient * x1 + intercept
plt.xlabel("Mthly_HH_Income")
plt.ylabel("Mthly_HH_Expense")
plt.title("Income vs Expense, colored by No_of_Fly_Members")
plt.plot(x1,y1,'-r')
plt.show()
```



In [66]:
```
sns.boxplot(df["No_of_Fly_Members"], df["Qualification_Num"])
```

```
C:\Users\test\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

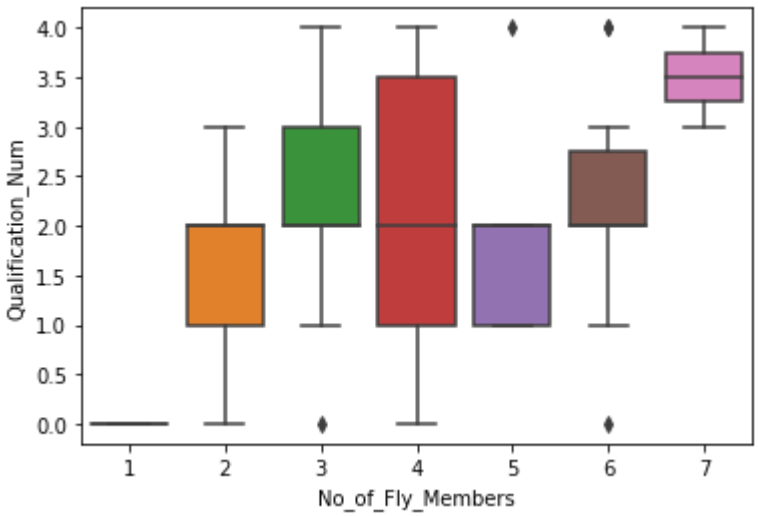
Out[66]:
```
<AxesSubplot:xlabel='No_of_Fly_Members', ylabel='Qualification_Num'>
```



In [66]:
```
sns.boxplot(df["No_of_Fly_Members"], df["Qualification_Num"])
```

```
C:\Users\test\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

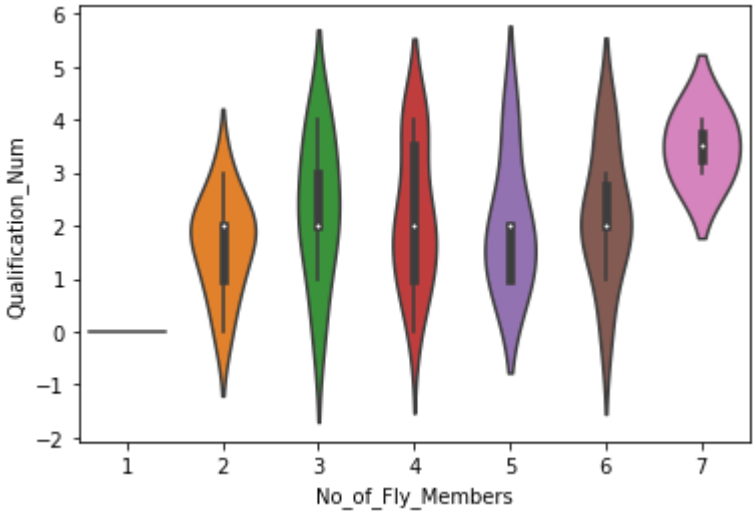
Out[66]:
```
<AxesSubplot:xlabel='No_of_Fly_Members', ylabel='Qualification_Num'>
```



In [67]:
```
sns.violinplot(df["No_of_Fly_Members"], df["Qualification_Num"])
```

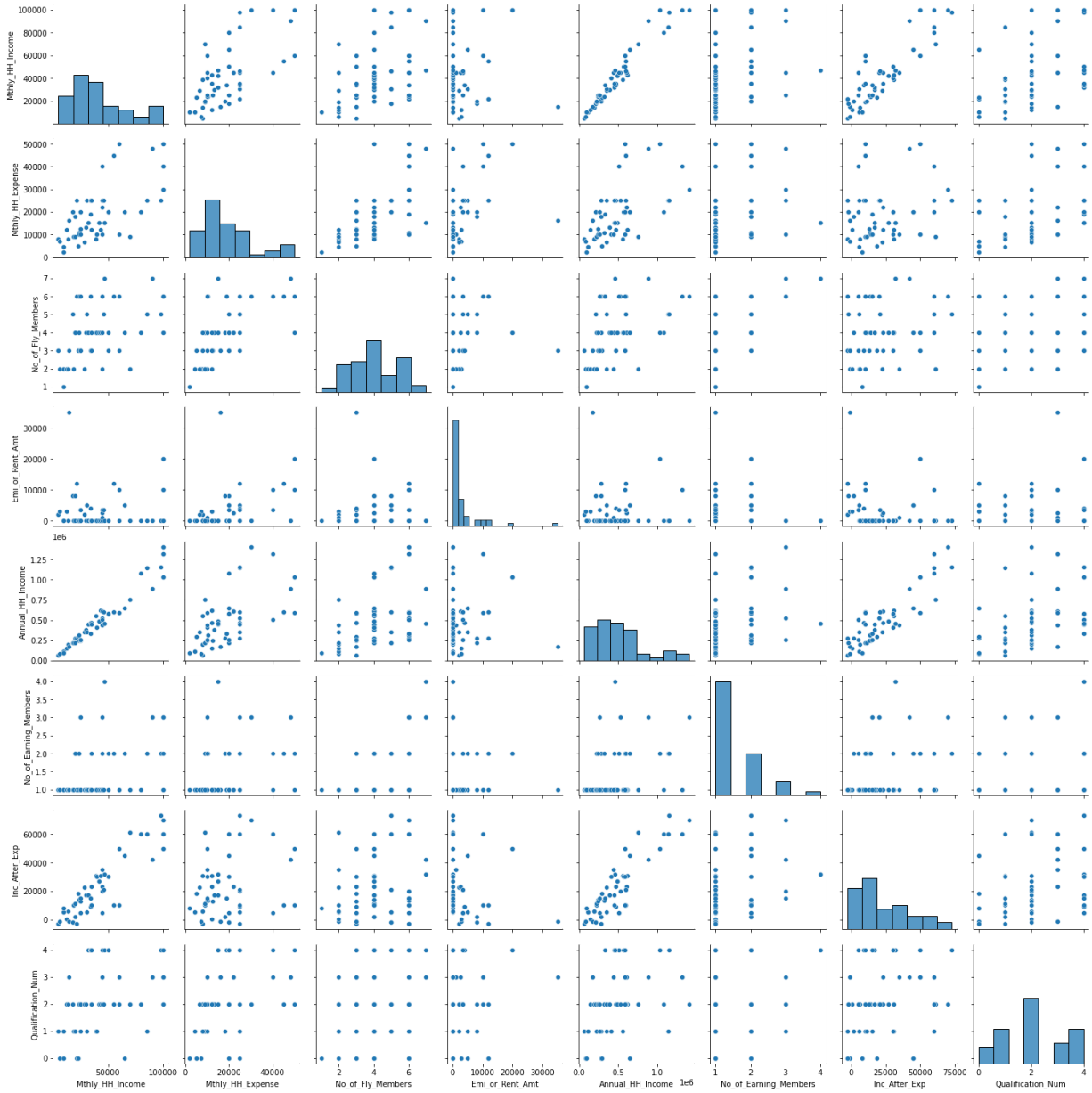
```
C:\Users\test\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarn
ing: Pass the following variables as keyword args: x, y. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[67]:
```
<AxesSubplot:xlabel='No_of_Fly_Members', ylabel='Qualification_Num'>
```



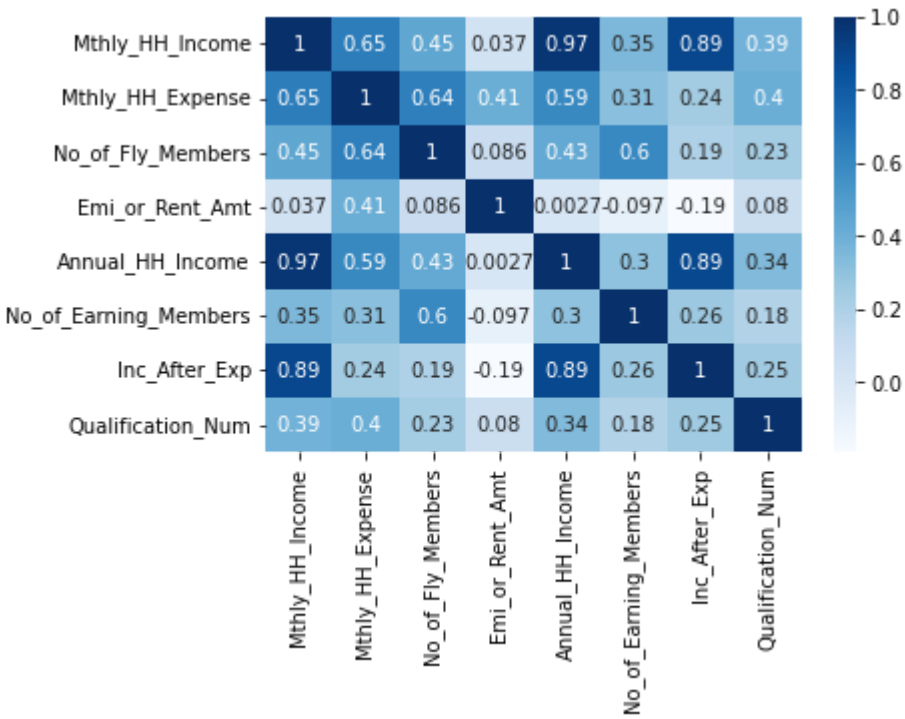
In [68]:
```
sns.pairplot(df)
```

Out[68]:
```
<seaborn.axisgrid.PairGrid at 0x481e7c45e0>
```

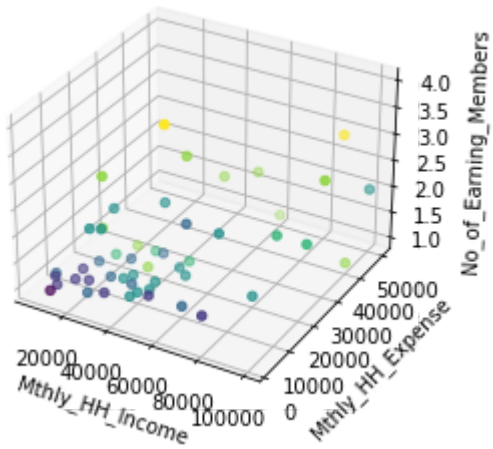


In [65]:
```
sns.heatmap(df.corr(), cmap="Blues", annot=True)
```

Out[65]: <AxesSubplot:>

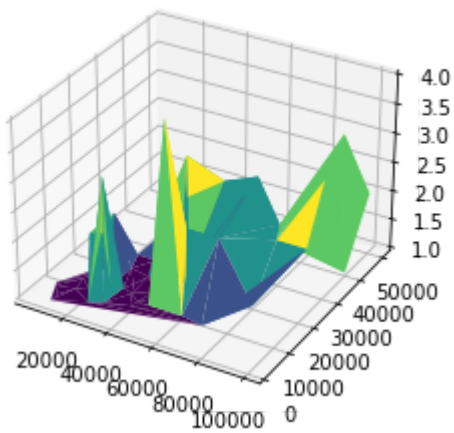


In [15]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

xs = df["Mthly_HH_Income"]
ys = df["Mthly_HH_Expense"]
zs = df["No_of_Earning_Members"]
ax.scatter(xs, ys, zs, c=df["No_of_Fly_Members"])

ax.set_xlabel("Mthly_HH_Income")
ax.set_ylabel("Mthly_HH_Expense")
ax.set_zlabel("No_of_Earning_Members")

plt.show()
```



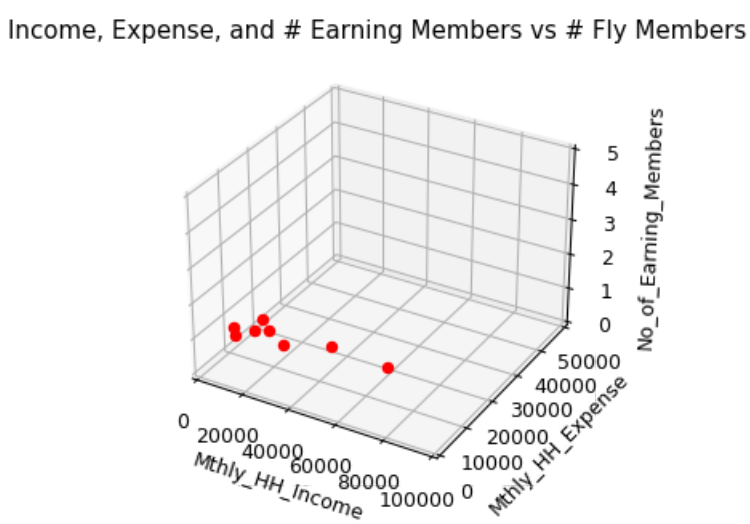
In [16]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(df["Mthly_HH_Income"], df["Mthly_HH_Expense"], df["No_of_Earning_
plt.show()
```

```
C:\Users\test\AppData\Local\Temp\ipykernel_5324\3654048881.py:2: MatplotlibDepr
ecationWarning: Calling gca() with keyword arguments was deprecated in Matplotl
ib 3.4. Starting two minor releases later, gca() will take no keyword argument
s. The gca() function should only be used to get the current axes, or if no axe
s exist, create new axes with default keyword arguments. To create a new axes w
ith non-default arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```



In [19]:
```
%matplotlib notebook
nfr = 20 # Number of frames
fps = 10 # Frame per sec
xs = []
ys = []
zs = []
ss = np.arange(1,nfr,0.5)

for s in ss:
    for i in range(1,8):
        xs.append(df[df["No_of_Fly_Members"] == i]["Mthly_HH_Income"])
        ys.append(df[df["No_of_Fly_Members"] == i]["Mthly_HH_Expense"])
        zs.append(df[df["No_of_Fly_Members"] == i]["No_of_Earning_Members"])

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sct, = ax.plot([], [], [], "o", c="r", markersize=5)

def update(ifrm, xa, ya, za):
    sct.set_data(xa[ifrm], ya[ifrm])
    sct.set_3d_properties(za[ifrm])

ax.set_xlim(0,100000)
ax.set_ylim(0,50000)
ax.set_zlim(0,5)

ax.set_xlabel("Mthly_HH_Income")
ax.set_ylabel("Mthly_HH_Expense")
ax.set_zlabel("No_of_Earning_Members")

plt.title("Income, Expense, and # Earning Members vs # Fly Members")

ani = animation.FuncAnimation(fig, update, nfr, fargs=(xs,ys,zs), interval=1000/f
```

```
<IPython.core.display.Javascript object>
```



In [79]:
```
%matplotlib notebook
nfr = 30
fps = 5
xs = []
ys = []
zs = []
ss = np.arange(1,nfr,0.5)
A = df["Mthly_HH_Income"]/1000
B = df["Mthly_HH_Expense"]/1000
C = df["No_of_Earning_Members"]
D = df["No_of_Fly_Members"]

for s in ss:
    for i in range(1,8):
        xs.append(df[D == i]["Mthly_HH_Income"]/1000)
        ys.append(df[D == i]["Mthly_HH_Expense"]/1000)
        zs.append(df[D == i]["No_of_Earning_Members"])

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sct, = ax.plot([], [], [], "o", c="blue", markersize=4, zorder=1000)

def update(ifrm, xa, ya, za):
    sct.set_data(xa[ifrm], ya[ifrm])
    sct.set_3d_properties(za[ifrm])

ax.set_xlim(0,100)
ax.set_ylim(0,50)
ax.set_zlim(0.5,4.5)

ax.set_xlabel("Income, in 1000s")
ax.set_ylabel("Expense, in 1000s")
ax.set_zlabel("No_of_Earning_Members")

plt.title("Income, Expense, and # Earning Members vs # Fly Members")

ani = animation.FuncAnimation(fig, update, nfr, fargs=(xs,ys,zs), interval=1000/f

ax.scatter(df["Mthly_HH_Income"]/1000, df["Mthly_HH_Expense"]/1000, df["No_of_Ear

ax = fig.gca(projection='3d')
ax.plot_trisurf(A, B, C, cmap="Wistia", linewidth=0.5, zorder = 0, edgecolors = '
ax.view_init(16, -37)
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
C:\Users\test\AppData\Local\Temp\ipykernel_5324\4226936226.py:41: MatplotlibDep
recationWarning: Calling gca() with keyword arguments was deprecated in Matplot
lib 3.4. Starting two minor releases later, gca() will take no keyword argument
s. The gca() function should only be used to get the current axes, or if no axe
s exist, create new axes with default keyword arguments. To create a new axes w
ith non-default arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
```

In [ ]:

```
%matplotlib notebook
nfr = 30
fps = 5
xs = []
ys = []
zs = []
ss = np.arange(1,nfr,0.5)
A = df["Inc_After_Exp"]/1000
B = df["Qualification_Num"]
C = df["No_of_Earning_Members"]
D = df["No_of_Fly_Members"]

for s in ss:
    for i in range(1,8):
        xs.append(df[D == i]["Inc_After_Exp"]/1000)
        ys.append(df[D == i]["Qualification_Num"])
        zs.append(df[D == i]["No_of_Earning_Members"])

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sct, = ax.plot([], [], [], "o", c="blue", markersize=4, zorder=1000)

def update(ifrm, xa, ya, za):
    sct.set_data(xa[ifrm], ya[ifrm])
    sct.set_3d_properties(za[ifrm])

ax.set_xlim(0,70)
ax.set_ylim(0,5)
ax.set_zlim(0.5,4.5)

ax.set_xlabel("Income After Expense, in 1000s")
ax.set_ylabel("Qualification Number")
ax.set_zlabel("No_of_Earning_Members")

plt.title("Income, Qualification, and # Earning Members vs # Fly Members")

ani = animation.FuncAnimation(fig, update, nfr, fargs=(xs,ys,zs), interval=1000/f

ax.scatter(df["Inc_After_Exp"]/1000, df["Qualification_Num"], df["No_of_Earning_M

ax = fig.gca(projection='3d')
ax.plot_trisurf(A, B, C, cmap="Wistia", linewidth=0.5, zorder = 0, edgecolors = '
ax.view_init(30, -70)
plt.show()
```

In [83]:
```
tempx = pd.read_html("https://data.cityofchicago.org/Environment-Sustainable-Deve
tempx
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [83], in <cell line: 1>()
----> 1 tempx = pd.read_html("https://data.cityofchicago.org/Environment-Sustai
nable-Development/Chicago-Energy-Benchmarking-2014-Data-Reported-in-/tepd-j7h5/
data")
      2 tempx

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:311, in deprecate
_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\io\html.py:1113, in read_html(io, mat
ch, flavor, header, index_col, skiprows, attrs, parse_dates, thousands, encodin
g, decimal, converters, na_values, keep_default_na, displayed_only)
   1109 validate_header_arg(header)
   1111 io = stringify_path(io)
-> 1113 return _parse(
   1114     flavor=flavor,
   1115     io=io,
   1116     match=match,
   1117     header=header,
   1118     index_col=index_col,
   1119     skiprows=skiprows,
   1120     parse_dates=parse_dates,
   1121     thousands=thousands,
   1122     attrs=attrs,
   1123     encoding=encoding,
   1124     decimal=decimal,
   1125     converters=converters,
   1126     na_values=na_values,
   1127     keep_default_na=keep_default_na,
   1128     displayed_only=displayed_only,
   1129 )

File ~\anaconda3\lib\site-packages\pandas\io\html.py:939, in _parse(flavor, io,
match, attrs, encoding, displayed_only, **kwargs)
    937 else:
    938     assert retained is not None  # for mypy
--> 939     raise retained
    941 ret = []
    942 for table in tables:

File ~\anaconda3\lib\site-packages\pandas\io\html.py:919, in _parse(flavor, io,
match, attrs, encoding, displayed_only, **kwargs)
    916 p = parser(io, compiled_match, attrs, encoding, displayed_only)
```

```
    918 try:
--> 919     tables = p.parse_tables()
    920 except ValueError as caught:
    921     # if `io` is an io-like object, check if it's seekable
    922     # and try to rewind it before trying the next parser
    923     if hasattr(io, "seekable") and io.seekable():

File ~\anaconda3\lib\site-packages\pandas\io\html.py:239, in _HtmlFrameParser.p
arse_tables(self)
    231 def parse_tables(self):
    232     """
    233     Parse and return all tables from the DOM.
    234
   (...)
    237     list of parsed (header, body, footer) tuples from tables.
    238     """
--> 239     tables = self._parse_tables(self._build_doc(), self.match, self.att
rs)
    240     return (self._parse_thead_tbody_tfoot(table) for table in tables)

File ~\anaconda3\lib\site-packages\pandas\io\html.py:569, in _BeautifulSoupHtml
5LibFrameParser._parse_tables(self, doc, match, attrs)
    566 tables = doc.find_all(element_name, attrs=attrs)
    568 if not tables:
--> 569     raise ValueError("No tables found")
    571 result = []
    572 unique_tables = set()

ValueError: No tables found
```

In [ ]: