MM20B007 : Data Analytics Lab Assignment 1

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Getting necessary packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

Reading the data

```
mdf = pd.read_csv('/content/drive/MyDrive/sem 7/EE5708/Assignment
1/Data/merged_data.csv', index_col = 0)
```

Visualizing the data

```
mdf.head()

  State                            AreaName  All_Poverty  M_Poverty  \
0    AK      Aleutians East Borough, Alaska          553        334
1    AK  Aleutians West Census Area, Alaska          499        273
2    AK      Anchorage Municipality, Alaska        23914      10698
3    AK          Bethel Census Area, Alaska         4364       2199
4    AK         Bristol Bay Borough, Alaska           69         33

   F_Poverty  FIPS  Med_Income  Med_Income_White  Med_Income_Black  \
0        219  2013     61518.0           72639.0           31250.0
1        226  2016     84306.0           97321.0           93750.0
2      13216  2020     78326.0           87235.0           50535.0
3       2165  2050     51012.0           92647.0           73661.0
4         36  2060     79750.0           88000.0               NaN

   Med_Income_Nat_Am  ...  F_Without  All_With  All_Without  fips_x  \
0            54750.0  ...        540      1442         1857    2013
1            48750.0  ...        564      4177         1333    2016
```

```
2          53935.0  ...      21393    243173          44638    2020
3          41594.0  ...       1774     13023           4482    2050
4          63333.0  ...         67       768            191    2060

   Incidence_Rate  Avg_Ann_Incidence  recent_trend  fips_y
Mortality_Rate  \
0               *         3 or fewer             *    2013
*
1               *         3 or fewer             *    2016
*
2            61.5                131        stable    2020
47.3
3            62.7                  6        stable    2050
58.3
4               *         3 or fewer             *    2060
*

  Avg_Ann_Deaths
0              *
1              *
2             96
3              5
4              *

[5 rows x 25 columns]
```

Preprocessing and Data Cleaning

```
mdf.columns

Index(['State', 'AreaName', 'All_Poverty', 'M_Poverty', 'F_Poverty',
'FIPS',
       'Med_Income', 'Med_Income_White', 'Med_Income_Black',
       'Med_Income_Nat_Am', 'Med_Income_Asian', 'Hispanic', 'M_With',
       'M_Without', 'F_With', 'F_Without', 'All_With', 'All_Without',
'fips_x',
       'Incidence_Rate', 'Avg_Ann_Incidence', 'recent_trend',
'fips_y',
       'Mortality_Rate', 'Avg_Ann_Deaths'],
      dtype='object')
```

Dropping unwanted columns

```
mdf.drop(['fips_x', 'fips_y'], axis = 1, inplace = True)
mdf.columns

Index(['State', 'AreaName', 'All_Poverty', 'M_Poverty', 'F_Poverty',
'FIPS',
       'Med_Income', 'Med_Income_White', 'Med_Income_Black',
       'Med_Income_Nat_Am', 'Med_Income_Asian', 'Hispanic', 'M_With',
       'M_Without', 'F_With', 'F_Without', 'All_With', 'All_Without',
       'Incidence_Rate', 'Avg_Ann_Incidence', 'recent_trend',
'Mortality_Rate',
       'Avg_Ann_Deaths'],
      dtype='object')
```

```
mdf.shape
```

```
(3134, 23)
```

```
df = mdf.copy()
```

```
df[['Med_Income', 'Med_Income_White', 'Med_Income_Black',
       'Med_Income_Nat_Am', 'Med_Income_Asian', 'Hispanic']].mean()
```

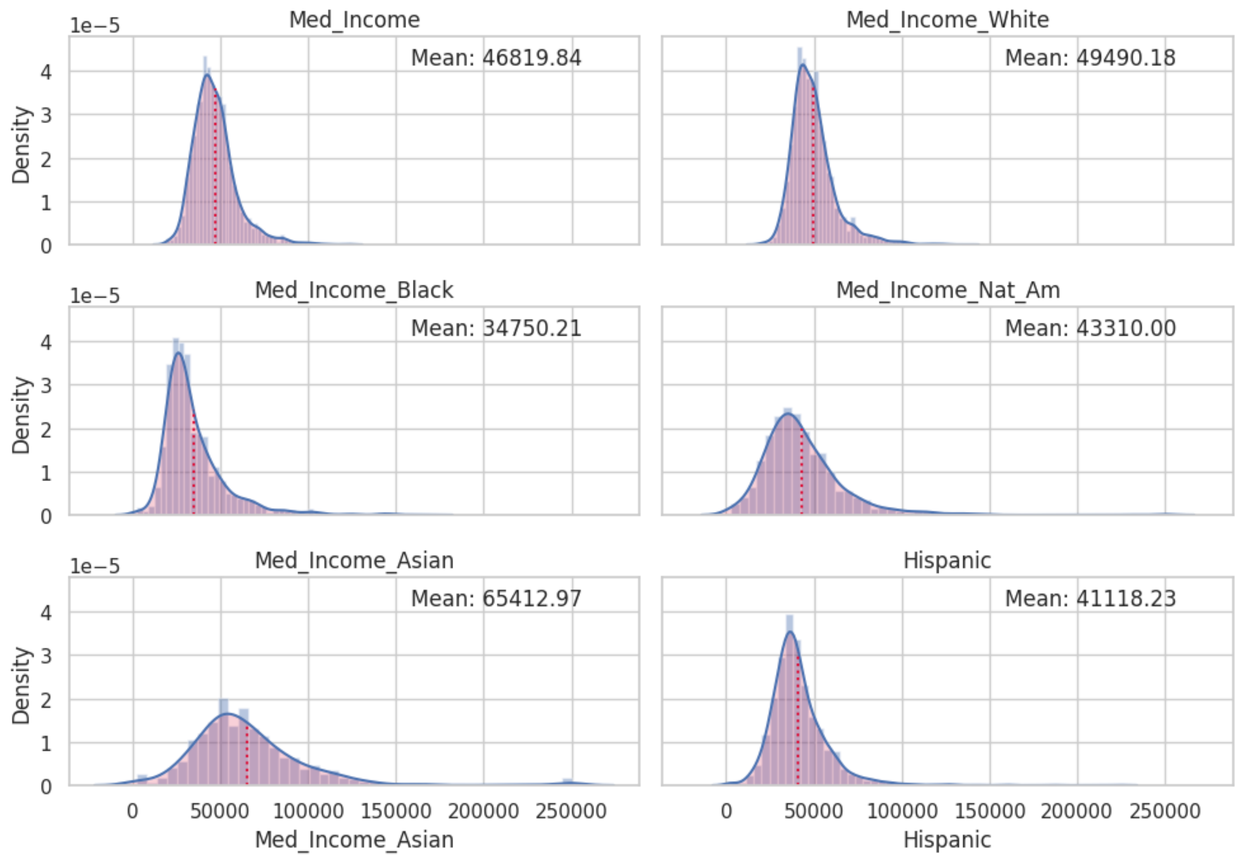
```
Med_Income           46819.837855
Med_Income_White     49490.181992
Med_Income_Black     34750.214137
Med_Income_Nat_Am    43309.998643
Med_Income_Asian     65412.969499
Hispanic             41118.231553
dtype: float64
```

Function to plot distributions as subplots

```
def dist_plot(features, rows, cols, data, figsize=(10, 7)):
    """
    Plot distributions of features in subplots and display the mean
value in each subplot.

    Parameters:
        features (list): List of feature names to plot.
        rows (int): Number of subplot rows.
        cols (int): Number of subplot columns.
        data (DataFrame): The data containing the features.
        figsize (tuple): Figure size (width, height).

    Returns:
        None
    """
    fig, axes = plt.subplots(rows, cols, figsize=figsize, sharex=True,
sharey=True)

    # Flatten the 2D array of subplots for easier iteration
    for i, ax in enumerate(axes.flatten()):
        feature_name = features[i]
        feature_data = data[feature_name]

        # Plot the distribution using Seaborn
        sns.distplot(feature_data, ax=ax)

        # Set title for the subplot
        ax.set_title(feature_name)

        # Get the KDE line and calculate mean height
        kde_line = ax.lines[0]
        mean = feature_data.mean()
        xs = kde_line.get_xdata()
        ys = kde_line.get_ydata()
        height = np.interp(mean, xs, ys)

        # Add a vertical line at the mean and fill area under the KDE
curve
        ax.vlines(mean, 0, height, color='crimson', linestyle=':')
        ax.fill_between(xs, 0, ys, facecolor='crimson', alpha=0.2)

        # Add text to display the mean value
        ax.text(0.75, 0.9, "Mean: {:.2f}".format(mean),
                horizontalalignment='center',
verticalalignment='center',
                transform=ax.transAxes)

    plt.tight_layout()
    plt.show()

income = ['Med_Income', 'Med_Income_White', 'Med_Income_Black',
       'Med_Income_Nat_Am', 'Med_Income_Asian', 'Hispanic']
dist_plot(income, 3, 2, df)
```

```
<ipython-input-137-ddcc233c517a>:23: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feature_data, ax=ax)
<ipython-input-137-ddcc233c517a>:23: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feature_data, ax=ax)
<ipython-input-137-ddcc233c517a>:23: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feature_data, ax=ax)
<ipython-input-137-ddcc233c517a>:23: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feature_data, ax=ax)
<ipython-input-137-ddcc233c517a>:23: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feature_data, ax=ax)
<ipython-input-137-ddcc233c517a>:23: UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(feature_data, ax=ax)
```



These graphs show that different social groups have different mean incomes for other states; hence, if we can prove that the median income is a valid factor in determining the average annual incidence or death, we can also assume that socioeconomic status would be a valid factor!

Checking the number of null values in each of the features

```
mdf.isnull().sum()

State                       0
AreaName                    0
```

```
All_Poverty              0
M_Poverty                0
F_Poverty                0
FIPS                     0
Med_Income               1
Med_Income_White         2
Med_Income_Black      1210
Med_Income_Nat_Am     1660
Med_Income_Asian      1757
Hispanic               681
M_With                   0
M_Without                0
F_With                   0
F_Without                0
All_With                 0
All_Without              0
Incidence_Rate           0
Avg_Ann_Incidence        0
recent_trend             0
Mortality_Rate           0
Avg_Ann_Deaths           0
dtype: int64
```

The columns 'Med_Income_White', 'Med_Income_Black', 'Med_Income_Nat_Am', 'Med_Income_Asian', and 'Hispanic' have too many null values hence we will be dropping them.

```
df.drop(['Med_Income_White', 'Med_Income_Black', 'Med_Income_Nat_Am',
'Med_Income_Asian', 'Hispanic'], axis = 1, inplace = True)
```

Checking, what all columns have numeric data type

```
df.apply(lambda s: pd.to_numeric(s, errors='coerce').notnull().all())
```

```
State                 False
AreaName              False
All_Poverty            True
M_Poverty              True
F_Poverty              True
FIPS                   True
Med_Income            False
M_With                 True
M_Without              True
F_With                 True
F_Without              True
All_With               True
All_Without            True
Incidence_Rate        False
Avg_Ann_Incidence     False
recent_trend          False
```

```
Mortality_Rate        False
Avg_Ann_Deaths        False
dtype: bool

df.describe()

        All_Poverty      M_Poverty      F_Poverty          FIPS  \
count  3.134000e+03    3134.000000    3134.000000   3134.000000
mean   1.522966e+04    6828.800893    8400.855775  30426.019145
std    5.457122e+04   24719.078097   29865.855831  15124.491165
min    1.000000e+01       5.000000       5.000000   1001.000000
25%    1.731250e+03     758.750000     957.000000  19001.500000
50%    4.294000e+03    1925.000000    2372.000000  29180.000000
75%    1.034550e+04    4697.500000    5812.500000  45080.500000
max    1.800265e+06  823612.000000  976653.000000  56045.000000

          Med_Income        M_With      M_Without        F_With  \
count    3133.000000  3.134000e+03    3134.000000  3.134000e+03
mean    46819.837855  4.158963e+04    6930.955329  4.487357e+04
std     12246.380184  1.293894e+05   28686.089548  1.406455e+05
min     19328.000000  3.200000e+01       4.000000  3.300000e+01
25%     38826.000000  4.506750e+03     750.000000  4.657500e+03
50%     45075.000000  1.040450e+04    1763.000000  1.110800e+04
75%     52224.000000  2.788775e+04    4407.250000  2.976475e+04
max    123453.000000  3.904322e+06  997326.000000  4.230137e+06

           F_Without      All_With   All_Without
count    3134.000000  3.134000e+03  3.134000e+03
mean     5968.701021  8.646320e+04  1.289966e+04
std     24657.276997  2.699985e+05  5.331494e+04
min         4.000000  6.700000e+01  8.000000e+00
25%       633.000000  9.173500e+03  1.388250e+03
50%      1529.000000  2.144800e+04  3.323500e+03
75%      3834.000000  5.756150e+04  8.240000e+03
max    837175.000000  8.134459e+06  1.834501e+06
```

Thus we can see we have to treat the columns [Incidence_Rate, Avg_Ann_Incidence, recent_trend, Mortality_Rate, Avg_Ann_Deaths] for values that are not numeric.

Also, we can see, all the independent columns are not normalized by population and we also do not have population data, it is better to delete the Mortality rate and Incidence rate columns as these are just the average values normalized by population and hence can be dropped !!

```
df.drop(['Mortality_Rate', 'Incidence_Rate'], axis = 1, inplace =
True)

# we create a filter to convert the non-numeric data to either numeric
or to replace by NULL
def filter_(x):
    try:
        return float(str(x).split(' ')[0])
    except ValueError:
        return float('NaN')

# Using the filter on different columns
df['Avg_Ann_Incidence'] = df['Avg_Ann_Incidence'].map(filter_)

df['Avg_Ann_Deaths'] = df['Avg_Ann_Deaths'].map(filter_)
df['Med_Income'] = df['Med_Income'].map(filter_)

print([i for i in df['Avg_Ann_Incidence'].unique() if type(i)==str])
print([i for i in df['Avg_Ann_Deaths'].unique() if type(i)==str])

[]
[]

# creating columns with Rising and falling !!

def boo(col, chck):
    if col == chck:
        return 1
    return 0

df['Rising'] = df['recent_trend'].apply(lambda x: boo(x, 'rising'))
df['Falling'] = df['recent_trend'].apply(lambda x: boo(x, 'falling'))

df.select_dtypes(include=np.number)

      All_Poverty  M_Poverty  F_Poverty   FIPS  Med_Income  M_With
M_Without  \
0             553        334        219   2013     61518.0     876
1317
1             499        273        226   2016     84306.0    2470
769
2           23914      10698      13216   2020     78326.0  120747
23245
3            4364       2199       2165   2050     51012.0    6396
2708
4              69         33         36   2060     79750.0     419
124
...           ...        ...        ...    ...         ...     ...
...
3129         5058       2177       2881  56037     69022.0   19891
3318
3130         1638       1026        612  56039     75325.0    8948
2558
3131         2845       1453       1392  56041     56569.0    9132
1413
3132         1137        489        648  56043     47652.0    3349
691
3133          958        354        604  56045     57738.0    2927
```

```
454

      F_With  F_Without  All_With  All_Without  Avg_Ann_Incidence  \
0        566        540      1442         1857                3.0   
1       1707        564      4177         1333                3.0   
2     122426      21393    243173        44638              131.0   
3       6627       1774     13023         4482                6.0   
4        349         67       768          191                3.0   
...      ...        ...       ...          ...                ...   
3129   18600       2683     38491         6001               14.0   
3130    9555       1192     18503         3750                5.0   
3131    8711       1503     17843         2916                6.0   
3132    3490        703      6839         1394                6.0   
3133    3087        314      6014          768                4.0   

      Avg_Ann_Deaths  Rising  Falling  
0                NaN       0        0  
1                NaN       0        0  
2               96.0       0        0  
3                5.0       0        0  
4                NaN       0        0  
...              ...     ...      ...  
3129             9.0       0        0  
3130             5.0       0        0  
3131             4.0       0        0  
3132             5.0       0        0  
3133             4.0       0        0  

[3134 rows x 15 columns]
```

```
#checking for null values post the pre-processing
df.isnull().sum()
```

```
State                  0
AreaName               0
All_Poverty            0
M_Poverty              0
F_Poverty              0
FIPS                   0
Med_Income             1
M_With                 0
M_Without              0
F_With                 0
F_Without              0
All_With               0
All_Without            0
Avg_Ann_Incidence    209
recent_trend           0
Avg_Ann_Deaths       331
Rising                 0
```

```
Falling                0
dtype: int64

#Checking for mean values for different features, grouped by state
df.groupby(['State']).mean()

<ipython-input-150-94714262a348>:2: FutureWarning: The default value
of numeric_only in DataFrameGroupBy.mean is deprecated. In a future
version, numeric_only will default to False. Either specify
numeric_only or select only columns which should be valid for the
function.
  df.groupby(['State']).mean()

         All_Poverty     M_Poverty     F_Poverty          FIPS
Med_Income  \
State

AK       2978.782609   1425.260870   1553.521739   2138.217391
66812.565217
AL      13242.686567   5740.791045   7501.895522   1067.000000
37973.134328
AR       7381.920000   3303.280000   4078.640000   5075.000000
36626.480000
AZ      78712.666667  36501.133333  42211.533333   4013.866667
43252.200000
CA     105778.310345  48712.758621  57065.551724   6058.000000
56013.155172
CO      10218.265625   4729.312500   5488.953125   8062.234375
51263.187500
CT      45793.875000  20125.000000  25668.875000   9008.000000
71184.125000
DC     110365.000000  48069.000000  62296.000000  11001.000000
70848.000000
DE      36105.000000  15373.000000  20732.000000  10003.000000
58067.666667
FL      47464.313433  21485.835821  25978.477612  12067.910448
44046.477612
GA      11251.238994   4968.270440   6282.968553  13161.490566
40704.911950
HI      30788.800000  14142.200000  16646.600000  15005.000000
64879.000000
IA       3776.595960   1693.616162   2082.979798  19099.000000
50483.121212
ID       5572.204545   2591.636364   2980.568182  16044.000000
43607.750000
IL      17658.019608   7873.049020   9784.970588  17102.000000
50163.441176
IN      10630.902174   4719.782609   5911.119565  18092.000000
48745.402174
KS       3631.933333   1651.476190   1980.457143  20105.000000
```

```
47322.209524
KY        6715.341667   3007.733333   3707.608333  21120.000000
39137.300000
LA       13879.375000   5932.671875   7946.703125  22064.000000
41411.781250
MA       53493.214286  23007.857143  30485.357143  25014.000000
65974.428571
MD       24033.541667  10270.708333  13762.833333  24044.958333
69200.375000
ME       11267.375000   5047.500000   6219.875000  23016.000000
46141.750000
MI       19480.361446   8863.554217  10616.807229  26083.000000
44464.987952
MN        6858.183908   3145.494253   3712.689655  27087.000000
53926.988506
MO        7964.973913   3553.165217   4411.808696  29117.713043
41755.400000
MS        7945.670732   3430.256098   4515.414634  28082.000000
34938.926829
MT        2689.035714   1246.785714   1442.250000  30056.000000
44497.017857
NC       16674.650000   7399.320000   9275.330000  37100.000000
41784.200000
ND        1504.867925    669.962264    834.905660  38053.000000
55574.867925
NE        2485.107527   1088.548387   1396.559140  31093.000000
48646.129032
NH       11384.000000   5107.000000   6277.000000  33010.000000
60648.900000
NJ       44992.714286  19710.857143  25281.857143  34021.000000
73014.095238
NM       13010.939394   6034.939394   6976.000000  35030.151515
40183.666667
NV       25078.647059  11705.352941  13373.294118  32045.529412
53689.705882
NY       48482.951613  21388.532258  27094.419355  36062.000000
55275.693548
OH       20179.954545   8994.931818  11185.022727  39088.000000
48446.409091
OK        8104.454545   3612.857143   4491.597403  40077.000000
44097.376623
OR       17692.972222   8237.222222   9455.750000  41036.000000
45171.222222
PA       24874.164179  11000.119403  13874.044776  42067.000000
50316.253731
RI       28844.600000  12603.400000  16241.200000  44005.000000
65783.400000
SC       18063.065217   7840.521739  10222.543478  45046.000000
39756.695652
```

```
SD       1652.692308    756.615385    896.076923  46067.430769
47679.738462
TN      11764.147368   5240.600000   6523.547368  47095.000000
40168.031579
TX      17608.074803   7883.543307   9724.531496  48254.000000
46745.778656
UT      12124.172414   5646.620690   6477.551724  49029.000000
54687.034483
VA       6927.651515   3028.363636   3899.287879  51265.848485
53059.212121
VT       4945.214286   2213.642857   2731.571429  50014.000000
52653.500000
WA      23295.179487  10715.025641  12580.153846  53039.000000
50217.076923
WI      10060.388889   4519.638889   5540.750000  55071.097222
50649.000000
WV       5879.709091   2635.436364   3244.272727  54055.000000
39411.818182
WY       2825.869565   1255.739130   1570.130435  56023.000000
57042.304348

              M_With     M_Without         F_With     F_Without  \
State
AK      12366.565217   2917.739130   12147.608696   2411.217391
AL      29389.671642   4678.328358   32530.134328   4294.194030
AR      16012.226667   2881.893333   17237.906667   2603.880000
AZ     178389.533333  35436.066667  191685.466667  30056.200000
CA     269831.241379  52388.275862  287728.534483  43712.500000
CO      34745.546875   5534.781250   36322.625000   4447.265625
CT     194827.125000  19867.500000  212544.125000  15086.125000
DC     276285.000000  22198.000000  323314.000000  14813.000000
DE     132112.666667  13828.000000  146781.000000  11207.666667
FL     112310.552239  27464.761194  124417.328358  24393.179104
GA      24288.446541   5449.440252   26859.364780   5116.691824
HI     123945.400000   8997.200000  130480.400000   7047.000000
IA      14020.090909   1243.060606   14529.464646   1006.444444
ID      15305.886364   2773.363636   15626.613636   2562.863636
IL      53124.647059   7664.294118   57616.421569   6047.813725
IN      29740.010870   4733.456522   31571.097826   4270.260870
KS      11722.238095   1594.009524   12237.314286   1435.695238
KY      15362.841667   2174.991667   16486.958333   1910.525000
LA      28475.515625   5594.515625   31249.015625   5339.109375
MA     218643.642857  10231.071429  237913.857143   6623.428571
MD     104805.500000  12109.708333  116597.083333   9684.958333
ME      35496.687500   4614.687500   38427.937500   3637.187500
MI      51325.879518   6328.108434   55251.277108   5036.240964
MN      28082.068966   2476.908046   29202.885057   1857.229885
MO      21828.939130   3295.878261   23439.895652   3021.800000
MS      14163.085366   2915.890244   15824.743902   2720.536585
```

```
MT       7459.232143   1468.642857    7611.232143   1303.982143
NC      39094.400000   7360.030000   43500.040000   6541.590000
ND       6080.886792    679.924528    6035.056604    530.679245
NE       8726.322581   1086.344086    9035.322581    947.000000
NH      57839.500000   6636.700000   60861.000000   5568.500000
NJ     176838.857143  26562.190476  193311.333333  22173.523810
NM      25007.909091   5452.000000   26911.939394   4740.333333
NV      65350.588235  15792.000000   67480.411765  13947.588235
NY     134038.258065  17297.806452  149030.435484  12977.209677
OH      56197.590909   6936.954545   60783.000000   5660.829545
OK      19749.077922   4244.116883   21039.896104   3907.480519
OR      46073.055556   7253.416667   48939.194444   6089.083333
PA      82057.477612   9146.940299   89112.880597   7370.582090
RI      89317.200000  10941.000000   99037.000000   8290.800000
SC      41092.608696   7752.934783   45848.934783   7033.695652
SD       5532.461538    709.692308    5630.861538    609.646154
TN      27868.221053   4724.884211   30850.915789   3897.357895
TX      39586.787402  10816.748031   41903.157480  10301.303150
UT      42691.068966   6939.689655   43393.172414   6177.206897
VA      25542.757576   3658.446970   27926.901515   3227.848485
VT      20227.000000   1575.571429   21508.642857   1007.142857
WA      76236.564103  11122.410256   79952.794872   9069.102564
WI      35358.930556   3563.430556   37140.777778   2664.125000
WV      14296.309091   1975.145455   15042.254545   1820.836364
WY      10802.956522   1764.608696   10670.782609   1552.869565

            All_With   All_Without  Avg_Ann_Incidence  Avg_Ann_Deaths
\
State

AK      24514.173913   5328.956522          15.130435       19.833333

AL      61919.805970   8972.522388          59.597015       47.552239

AR      33250.133333   5485.773333          35.733333       28.546667

AZ     370075.000000  65492.266667         252.600000       91.214286

CA     557559.775862  96100.775862         294.396552      165.603774

CO      71068.171875   9982.046875          35.281250       39.075000

CT     407371.250000  34953.625000         332.875000      216.875000

DC     599599.000000  37011.000000         351.000000      240.000000

DE     278893.666667  25035.666667         257.666667      188.333333

FL     236727.880597  51857.940299         245.462687      178.029851
```

```
GA   51147.811321  10566.132075          39.823899        29.841060
HI  254425.800000  16044.200000         155.800000       134.250000
IA   28549.555556   2249.505051          24.131313        18.051546
ID   30932.500000   5336.227273          19.500000        20.586207
IL  110741.068627  13712.107843          91.254902        42.178218
IN   61311.108696   9003.717391          57.771739        43.586957
KS   23959.552381   3029.704762                NaN        20.826087
KY   31849.800000   4085.516667          40.083333        29.008403
LA   59724.531250  10933.625000          54.453125        42.500000
MA  456557.500000  16854.500000         358.785714       247.500000
MD  221402.583333  21794.666667         152.458333       114.083333
ME   73924.625000   8251.875000          82.750000        59.750000
MI  106577.156627  11364.349398          95.012048        58.592593
MN   57284.954023   4334.137931                NaN        28.719512
MO   45268.834783   6317.678261          46.243478        34.333333
MS   29987.829268   5636.426829          30.487805        23.975309
MT   15070.464286   2772.625000          13.321429        13.485714
NC   82594.440000  13901.620000          75.580000        54.840000
ND   12115.943396   1210.603774           9.094340        12.000000
NE   17761.645161   2033.344086          13.473118        15.035714
NH  118700.500000  12205.200000         105.600000        73.600000
NJ  370150.190476  48735.714286         280.904762       195.095238
NM   51919.848485  10192.333333          29.424242        27.653846
NV  132831.000000  29739.588235                NaN       108.500000
NY  283068.693548  30275.016129         219.532258       146.677419
OH  116980.590909  12597.784091         109.670455        84.204545
```

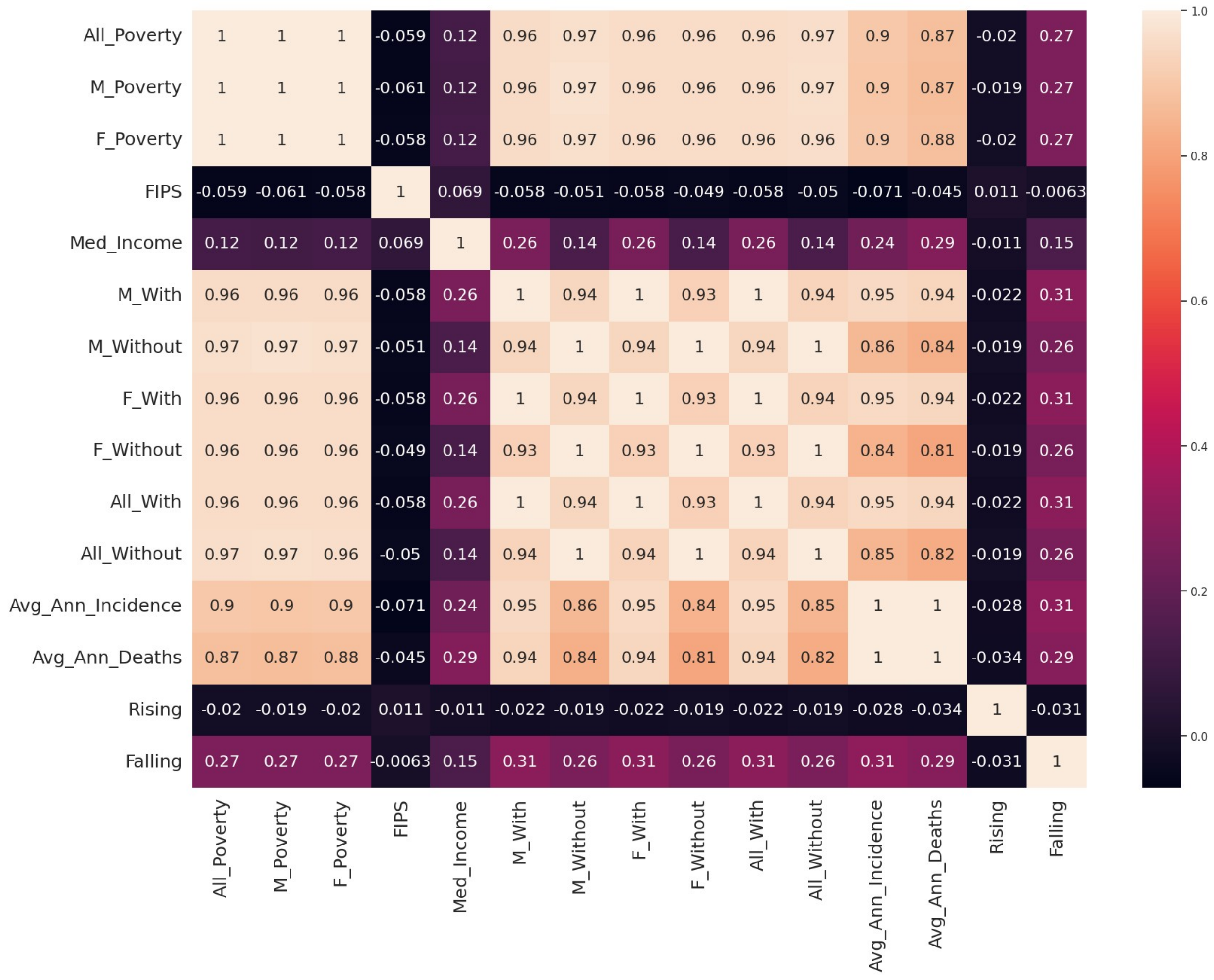
```
OK      40788.974026   8151.597403          39.272727         33.791667

OR      95012.250000  13342.500000          74.694444         62.363636

PA     171170.358209  16517.522388         159.074627        116.621212

RI     188354.200000  19231.800000         172.800000        124.600000

SC      86941.543478  14786.630435          81.413043         60.826087

SD      11163.323077   1319.338462           8.969231         10.882353

TN      58719.136842   8622.242105          59.515789         45.873684

TX      81489.944882  21118.051181          51.480315         41.954315

UT      86084.241379  13116.896552          22.241379         24.764706

VA      53469.659091   6886.295455          39.659091         30.387597

VT      41735.642857   2582.714286          37.357143         26.500000

WA     156189.358974  20191.512821         110.282051         84.837838

WI      72499.708333   6227.555556          55.888889         41.732394

WV      29338.563636   3795.981818          36.436364         27.436364

WY      21473.739130   3317.478261          12.739130         10.857143


         Rising   Falling
State
AK     0.000000  0.000000
AL     0.000000  0.074627
AR     0.000000  0.040000
AZ     0.066667  0.133333
CA     0.000000  0.258621
CO     0.015625  0.062500
CT     0.000000  0.375000
DC     0.000000  0.000000
DE     0.000000  0.000000
FL     0.000000  0.134328
GA     0.000000  0.050314
HI     0.000000  0.000000
IA     0.060606  0.030303
ID     0.000000  0.000000
IL     0.009804  0.127451
IN     0.032609  0.032609
KS     0.000000  0.000000
KY     0.016667  0.041667
```

```
LA      0.000000  0.031250
MA      0.000000  0.142857
MD      0.041667  0.125000
ME      0.000000  0.000000
MI      0.000000  0.072289
MN      0.000000  0.000000
MO      0.052174  0.034783
MS      0.012195  0.048780
MT      0.035714  0.017857
NC      0.010000  0.040000
ND      0.056604  0.037736
NE      0.010753  0.000000
NH      0.000000  0.100000
NJ      0.000000  0.666667
NM      0.000000  0.121212
NV      0.000000  0.000000
NY      0.032258  0.096774
OH      0.000000  0.113636
OK      0.000000  0.038961
OR      0.000000  0.055556
PA      0.029851  0.029851
RI      0.000000  0.000000
SC      0.043478  0.086957
SD      0.000000  0.030769
TN      0.010526  0.063158
TX      0.007874  0.031496
UT      0.000000  0.068966
VA      0.015152  0.106061
VT      0.000000  0.142857
WA      0.000000  0.282051
WI      0.027778  0.055556
WV      0.018182  0.018182
WY      0.000000  0.043478
```

```python
plt.figure(figsize=(20, 14))
corr_matrix = df.corr()
annot_kws = {"size": 16}

# Create the heatmap of Correlation matrix
heatmap = sns.heatmap(corr_matrix, annot=True, annot_kws=annot_kws,
xticklabels=True, yticklabels=True)

plt.xticks(fontsize=18)
plt.yticks(fontsize=18)

plt.show()
```

```
<ipython-input-151-4e706e5e56c5>:2: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the
```

```
value of numeric_only to silence this warning.
  corr_matrix = df.corr()
```



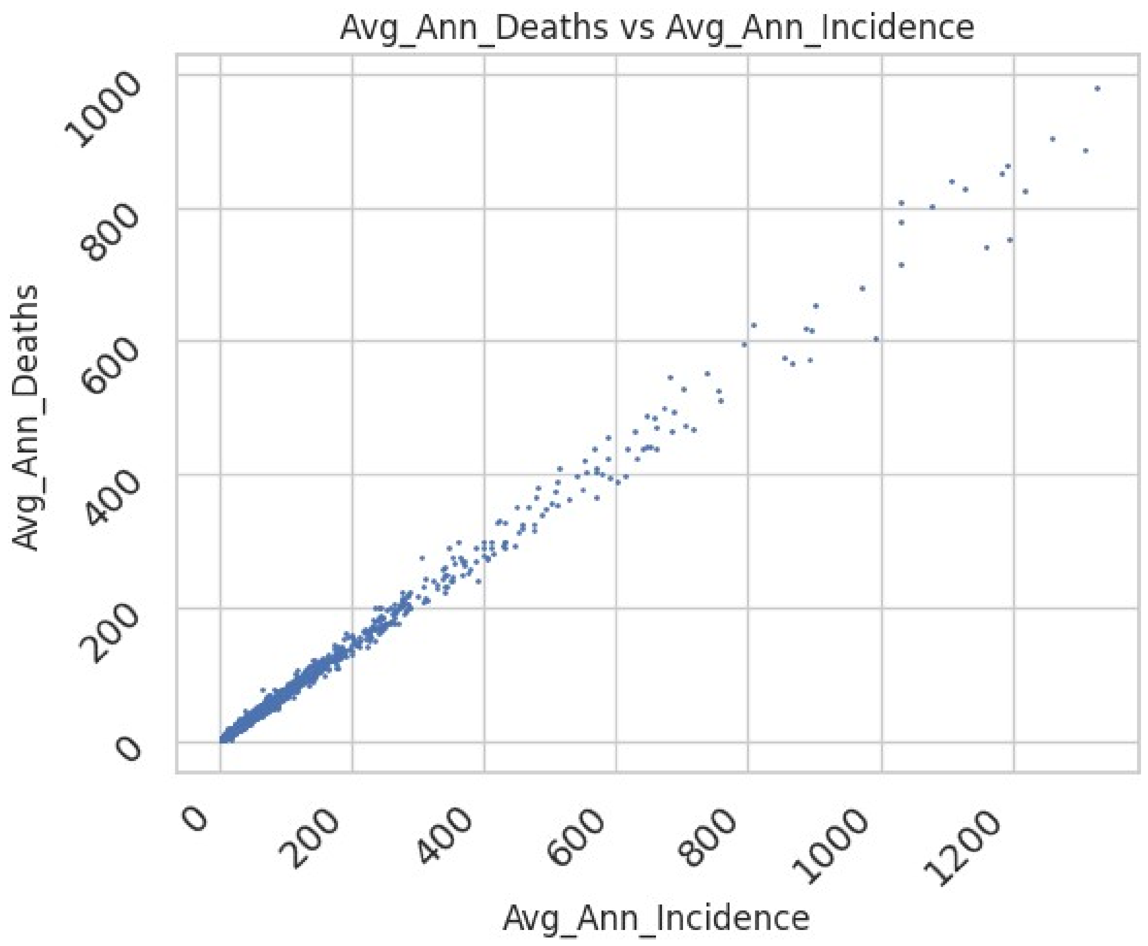
```
#Converting the Median Income column to numeric type
df['Med_Income'] = pd.to_numeric(df.Med_Income)

# Checks what all columns in the dataframe contain only numeric values
df.apply(lambda s: pd.to_numeric(s, errors='coerce').notnull().all())

State                False
AreaName             False
All_Poverty           True
M_Poverty             True
F_Poverty             True
FIPS                  True
Med_Income           False
M_With                True
M_Without             True
F_With                True
```

```
F_Without               True
All_With                True
All_Without             True
Avg_Ann_Incidence      False
recent_trend           False
Avg_Ann_Deaths         False
Rising                  True
Falling                 True
dtype: bool
```

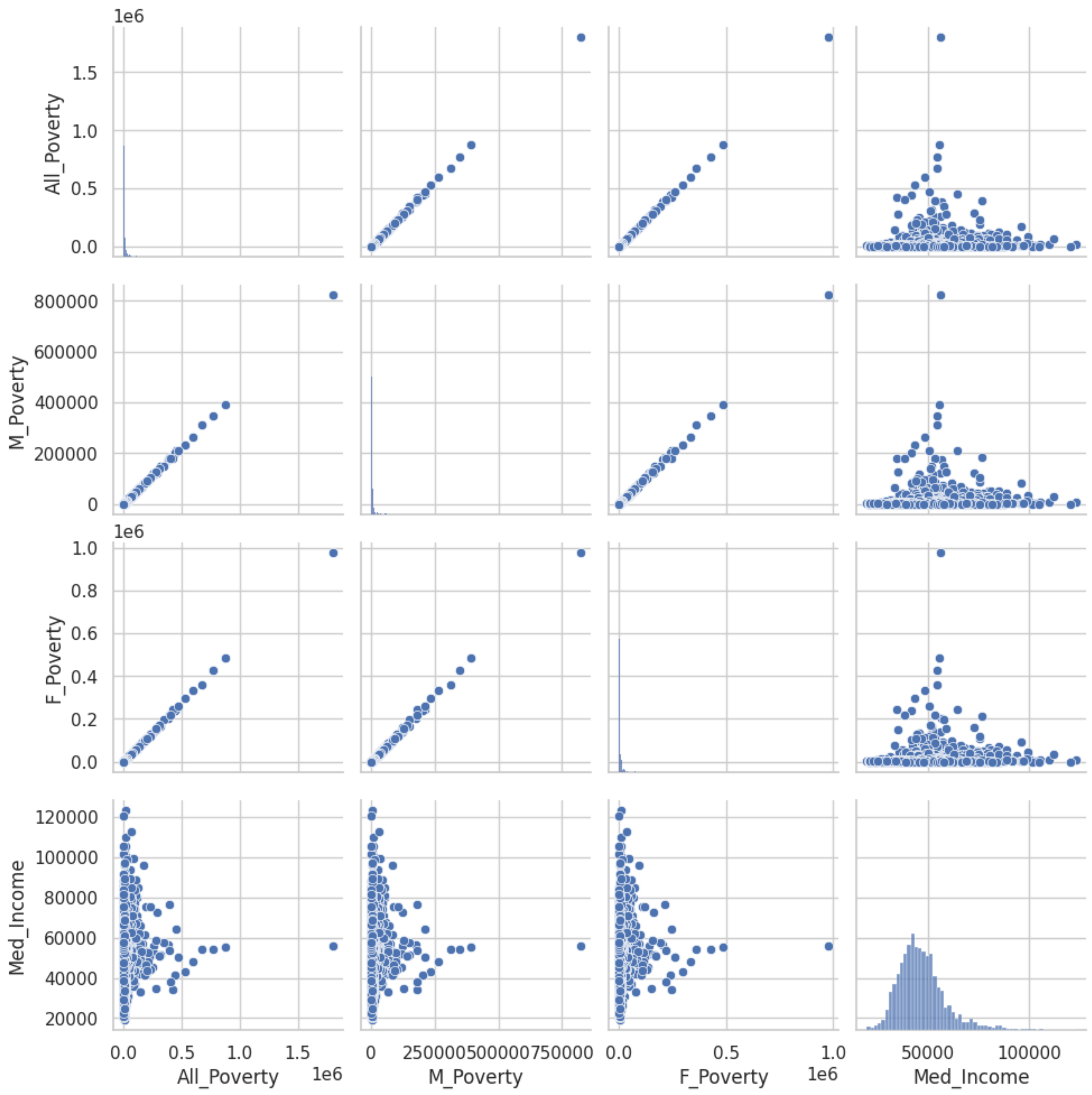
Visualization

```
# Scatter plot Avg_Ann_Incidence vs Avg_Ann_Deaths
df.plot(x = 'Avg_Ann_Incidence', y = 'Avg_Ann_Deaths', kind=
'scatter', s=0.9)
plt.grid(True)
plt.title('Avg_Ann_Deaths vs Avg_Ann_Incidence')
plt.xticks(rotation=45, ha="right", fontsize=14)
plt.yticks(rotation=45, ha="right", fontsize=14)
```

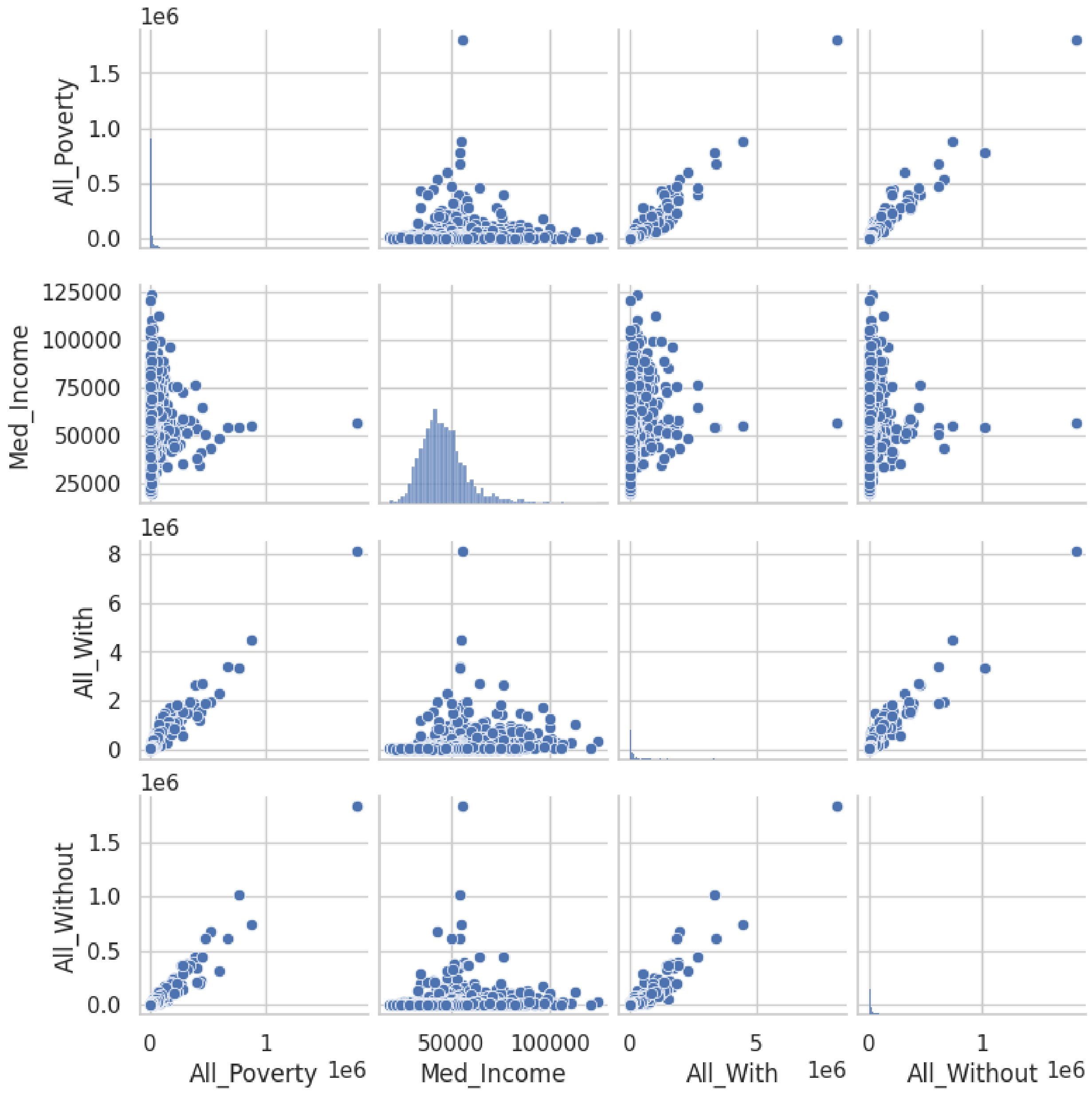
```
(array([-200.,    0.,  200.,  400.,  600.,  800., 1000., 1200.]),
 [Text(0, -200.0, '−200'),
  Text(0, 0.0, '0'),
  Text(0, 200.0, '200'),
  Text(0, 400.0, '400'),
  Text(0, 600.0, '600'),
  Text(0, 800.0, '800'),
  Text(0, 1000.0, '1000'),
  Text(0, 1200.0, '1200')])
```



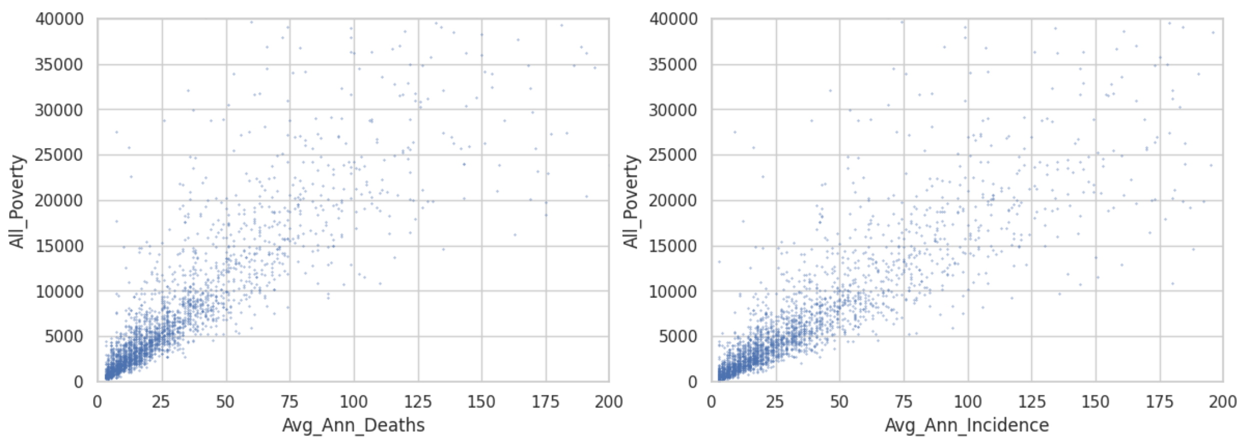
```
# Pair Plots
sns.set(style='whitegrid', context='notebook')

columns = ['All_Poverty', 'M_Poverty', 'F_Poverty', 'Med_Income']

pairplot = sns.pairplot(df[columns])

# pairplot.fig.suptitle('Pairplot of Poverty and Median Income',
y=1.08, fontsize=16)
plt.show()
```



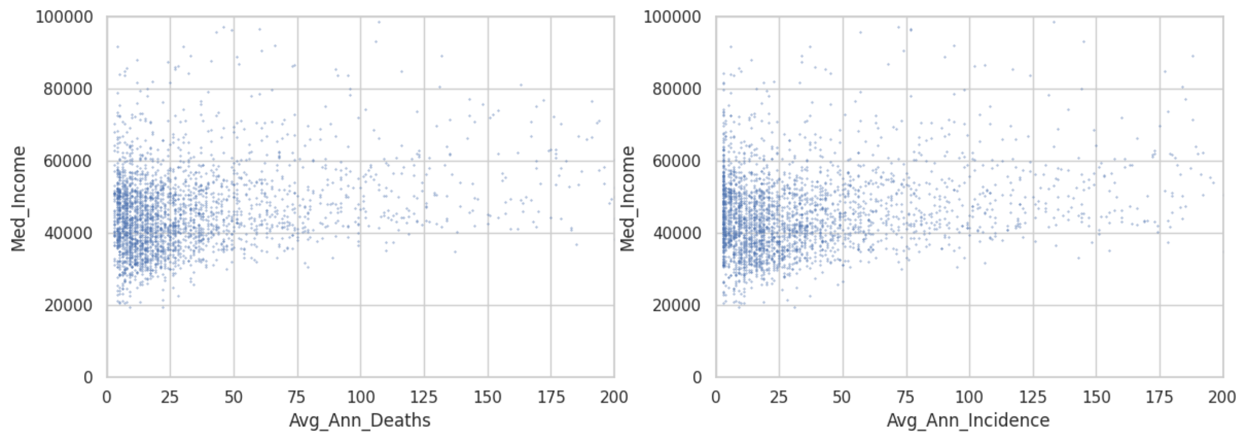
```
df.drop(['M_With', 'M_Without', 'F_With', 'F_Without'], axis=1,
inplace=True)

sns.set(style='whitegrid', context='notebook')
sns.pairplot(df[['All_Poverty', 'Med_Income', 'All_With',
       'All_Without']], height=2)

<seaborn.axisgrid.PairGrid at 0x7a8e0a4366e0>
```



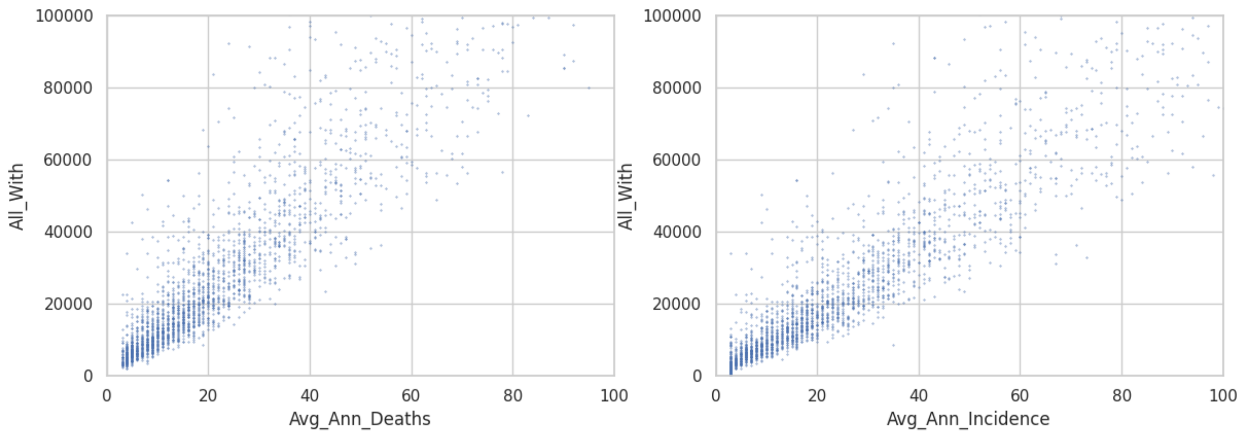
```
# Scatter Plot
def visualize_scatter_pov(col):
    fig1 = plt.figure(figsize = (14,10))

    ax3 = fig1.add_subplot(223)
    df.plot(x = 'Avg_Ann_Deaths', y = col, kind= 'scatter', s=0.1,
xlim = [0, 200], ylim = [0, 40000], ax = ax3)
    ax4 = fig1.add_subplot(224)
    df.plot(x = 'Avg_Ann_Incidence', y = col, kind= 'scatter',
s=0.1,xlim = [0, 200], ylim = [0, 40000], ax = ax4)

visualize_scatter_pov('All_Poverty')
```



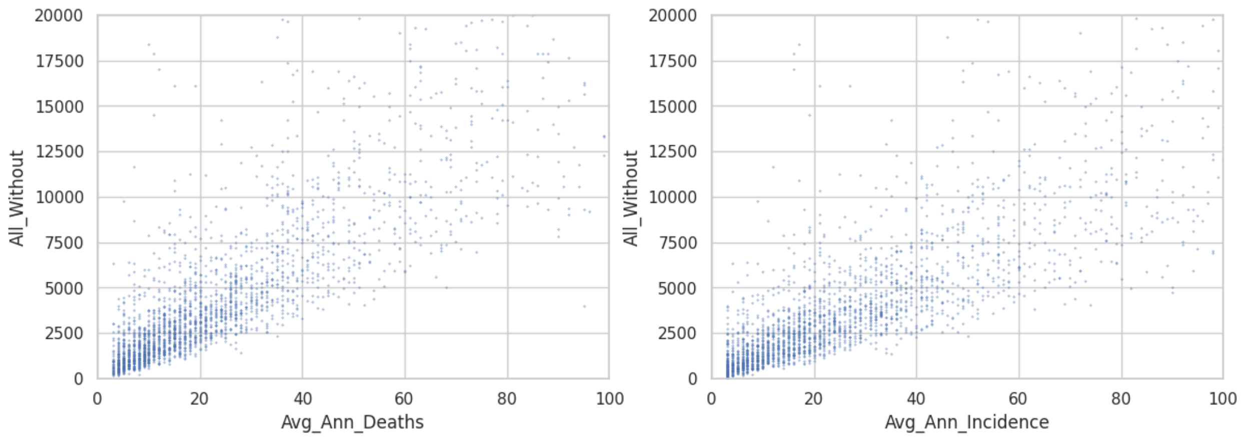
```
def visualize_scatter_inc(col):
    fig1 = plt.figure(figsize = (14,10))
    ax3 = fig1.add_subplot(223)
    df.plot(x = 'Avg_Ann_Deaths', y = col, kind= 'scatter',xlim = [0, 
200], ylim = [0, 100000], s=0.1, ax = ax3)
    ax4 = fig1.add_subplot(224)
    df.plot(x = 'Avg_Ann_Incidence', y = col, kind= 'scatter', 
s=0.1,xlim = [0, 200], ylim = [0, 100000], ax = ax4)

visualize_scatter_inc('Med_Income')
```



```
def visualize_scatter_with(col):
    fig1 = plt.figure(figsize = (14,10))
    ax3 = fig1.add_subplot(223)
    df.plot(x = 'Avg_Ann_Deaths', y = col, kind= 'scatter',xlim = [0, 
100], ylim = [0, 100000], s=0.1, ax = ax3)
    ax4 = fig1.add_subplot(224)
    df.plot(x = 'Avg_Ann_Incidence', y = col, kind= 'scatter', 
s=0.1,xlim = [0, 100], ylim = [0, 100000], ax = ax4)

visualize_scatter_with('All_With')
```



```python
def visualize_scatter_without(col):
    fig1 = plt.figure(figsize = (14,10))
    ax3 = fig1.add_subplot(223)
    df.plot(x = 'Avg_Ann_Deaths', y = col, kind= 'scatter',xlim = [0,
100], ylim = [0, 20000], s=0.1, ax = ax3)
    ax4 = fig1.add_subplot(224)
    df.plot(x = 'Avg_Ann_Incidence', y = col, kind= 'scatter',
s=0.1,xlim = [0, 100], ylim = [0, 20000], ax = ax4)

visualize_scatter_without('All_Without')
```



Statistical Linear Regression Modelling using Statsmodel Library

Unlike SKLearn, statsmodels does not automatically include a constant term when fitting models. Therefore, to incorporate a constant term, you should utilize the method sm.add_constant(X) within statsmodels. This addition of a constant term, although not obligatory, significantly enhances the quality of the fitted line. For instance, if your original line has an intercept of -2000 and you attempt to fit the same line through the origin, the resulting line will be of lower quality. However, upon introducing a constant term (intercept), you will observe that the coefficients align between SKLearn and statsmodels.

```python
res = ''
for i in df.columns:
    res += str(i) + ' + '
print(res)
```

```
State + AreaName + All_Poverty + M_Poverty + F_Poverty + FIPS +
Med_Income + All_With + All_Without + Avg_Ann_Incidence + recent_trend
+ Avg_Ann_Deaths + Rising + Falling +
```

```python
# Using statmodels library for Linear Regression Modelling
model1 = smf.ols(formula='Avg_Ann_Incidence ~ All_Poverty + Med_Income
+ All_With + All_Without + Rising + Falling', data=df).fit()
print(model1.summary())
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:      Avg_Ann_Incidence   R-squared:
0.922
Model:                            OLS   Adj. R-squared:
0.921
Method:                 Least Squares   F-statistic:
5707.
Date:                Fri, 01 Sep 2023   Prob (F-statistic):
0.00
Time:                        09:53:57   Log-Likelihood:
-15493.
No. Observations:                2924   AIC:
3.100e+04
Df Residuals:                    2917   BIC:
3.104e+04
Df Model:                           6

Covariance Type:            nonrobust

======================================================================
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
---------
Intercept      36.8125      3.931      9.365      0.000      29.105
44.520
All_Poverty     0.0005   8.38e-05      6.497      0.000       0.000
0.001
Med_Income     -0.0005   8.48e-05     -5.914      0.000      -0.001
-0.000
All_With        0.0008   1.36e-05     55.181      0.000       0.001
0.001
```

```
All_Without     -0.0014   6.39e-05    -22.047      0.000      -0.002
-0.001
Rising          -6.5225      7.452     -0.875      0.381     -21.134
8.089
Falling         11.5051      3.775      3.047      0.002       4.102
18.908
======================================================================
========
Omnibus:                     1643.837   Durbin-Watson:
1.614
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
599626.270
Skew:                           1.406   Prob(JB):
0.00
Kurtosis:                      73.098   Cond. No.
2.50e+06
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.5e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.

# Fitting to Average Deaths
model2 = smf.ols(formula='Avg_Ann_Deaths ~ All_Poverty + Med_Income  +
All_With + All_Without + Rising + Falling', data=df).fit()
print(model2.summary())

                            OLS Regression Results

======================================================================
========
Dep. Variable:         Avg_Ann_Deaths   R-squared:
0.893
Model:                            OLS   Adj. R-squared:
0.893
Method:                 Least Squares   F-statistic:
3900.
Date:                Fri, 01 Sep 2023   Prob (F-statistic):
0.00
Time:                        09:53:58   Log-Likelihood:
-13688.
No. Observations:                2803   AIC:
2.739e+04
Df Residuals:                    2796   BIC:
2.743e+04
Df Model:                           6
```

```
Covariance Type:            nonrobust
======================================================================
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
---------
Intercept      33.0356      2.701     12.233      0.000      27.740
38.331
All_Poverty     0.0003   5.89e-05      4.351      0.000       0.000
0.000
Med_Income     -0.0005   5.83e-05     -8.285      0.000      -0.001
-0.000
All_With        0.0005   9.38e-06     52.767      0.000       0.000
0.001
All_Without    -0.0004   5.02e-05     -8.832      0.000      -0.001
-0.000
Rising         -5.6806      5.163     -1.100      0.271     -15.805
4.443
Falling         2.4960      2.521      0.990      0.322      -2.446
7.438
======================================================================
========
Omnibus:                     1466.445   Durbin-Watson:
1.582
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
187089.919
Skew:                           1.474   Prob(JB):
0.00
Kurtosis:                      42.915   Cond. No.
1.93e+06
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.93e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

Multicollinearity

```
# Importing VIF to check for multicollinearity
from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

```
X = df[[ 'All_Poverty',  'Med_Income', 'All_With', 'All_Without',
'Rising', 'Falling', 'Avg_Ann_Incidence']]

X.dropna(inplace=True)
```

```
<ipython-input-167-32f8c2353909>:1: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X.dropna(inplace=True)
```

```
X1 = X[['All_Poverty', 'Med_Income', 'All_With', 'All_Without',
'Rising',
       'Falling']]

# Getting the VIFs corresponding to each of the features and then
treating the one with the highest value
pd.DataFrame([[var, variance_inflation_factor(X1.values,
X1.columns.get_loc(var))] for var in X1.columns],
                   index=range(X1.shape[1]), columns=['Variable',
'VIF'])
```

```
      Variable        VIF
0  All_Poverty  25.654166
1   Med_Income   1.271954
2     All_With  15.627624
3  All_Without  16.031002
4       Rising   1.015347
5      Falling   1.201154
```

The model's performance is impacted by considerable multicollinearity, evident from the elevated value of the variable's VIF (Variance Inflation Factor). This situation could potentially result in coefficients that lack statistical significance.

```
X.drop('All_Poverty', axis = 1, inplace=True)
X1.drop('All_Poverty', axis = 1, inplace=True)
```

```
<ipython-input-170-5a3ca0a533f8>:1: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X.drop('All_Poverty', axis = 1, inplace=True)
<ipython-input-170-5a3ca0a533f8>:2: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X1.drop('All_Poverty', axis = 1, inplace=True)
```

```
pd.DataFrame([[var, variance_inflation_factor(X1.values,
X1.columns.get_loc(var))] for var in X1.columns],
                   index=range(X1.shape[1]), columns=['Variable',
'VIF'])
```

```
      Variable       VIF
0   Med_Income  1.265045
1     All_With  9.776656
2  All_Without  8.926300
3       Rising  1.015175
4      Falling  1.199055
```

```
X.drop('All_Without', axis = 1, inplace=True)
X1.drop('All_Without', axis = 1, inplace=True)
```

```
<ipython-input-172-97aa2e8cd206>:1: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X.drop('All_Without', axis = 1, inplace=True)
<ipython-input-172-97aa2e8cd206>:2: SettingWithCopyWarning:
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/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X1.drop('All_Without', axis = 1, inplace=True)
```

```
pd.DataFrame([[var, variance_inflation_factor(X1.values,
X1.columns.get_loc(var))] for var in X1.columns],
                   index=range(X1.shape[1]), columns=['Variable',
'VIF'])
```

```
     Variable       VIF
0  Med_Income  1.207631
1    All_With  1.257105
2      Rising  1.015051
3     Falling  1.193283
```

```
# Final model after treating for multicollinearity
model1 = smf.ols(formula='Avg_Ann_Incidence ~Med_Income  + All_With +
Rising + Falling', data=df).fit()
print(model1.summary())
```

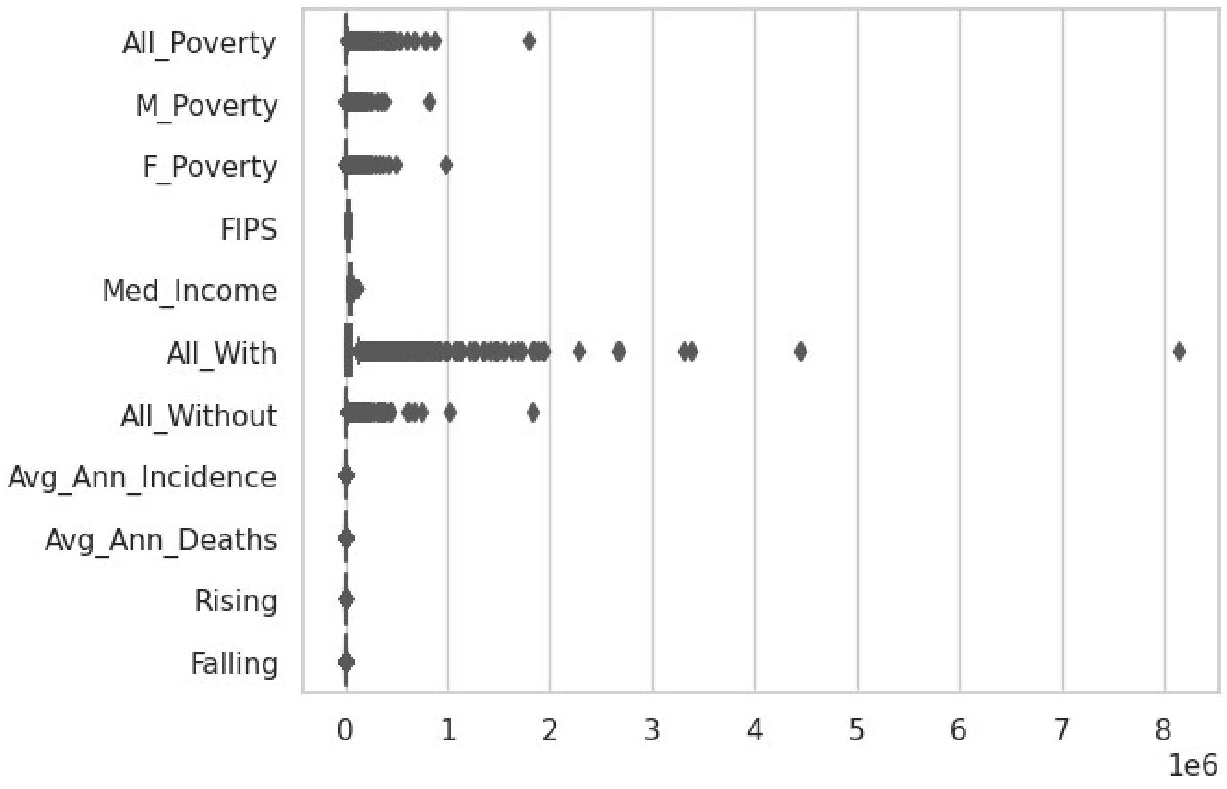
```
                            OLS Regression Results                            
==============================================================================
========
Dep. Variable:      Avg_Ann_Incidence   R-squared:
0.906
Model:                            OLS   Adj. R-squared:
0.906
Method:                 Least Squares   F-statistic:
7020.
Date:                Fri, 01 Sep 2023   Prob (F-statistic):
0.00
Time:                        09:53:58   Log-Likelihood:
-15759.
No. Observations:                2924   AIC:
3.153e+04
Df Residuals:                    2919   BIC:
3.156e+04
Df Model:                           4

Covariance Type:            nonrobust

==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
Intercept     24.4731      3.875      6.315      0.000      16.874
32.072
Med_Income    -0.0002   8.23e-05     -1.898      0.058      -0.000
5.17e-06
All_With       0.0006   3.84e-06    154.332      0.000       0.001
0.001
Rising        -6.8821      8.159     -0.844      0.399     -22.879
9.115
Falling       16.1143      4.122      3.909      0.000       8.032
24.196
==============================================================================
========
Omnibus:                     1260.365   Durbin-Watson:
1.646
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1570186.457
Skew:                           0.370   Prob(JB):
0.00
Kurtosis:                     116.523   Cond. No.
2.42e+06
==============================================================================
========
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.42e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

Detecting Outliers

```
data_mod1 = df.copy()
#using IQR method
dat = data_mod1['Med_Income']
Q1 = dat.quantile(0.25)
Q3 = dat.quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (dat >= Q1 - 3 * IQR) & (dat <= Q3 + 3 *IQR)
data_mod1 = data_mod1.loc[filter]
print(data_mod1.shape)
```

```
(3112, 14)
```

```
# Boxplot for the initial data
ax = sns.boxplot(data=df, orient="h", palette="Set2")
```



```
# Fitting to Average Deaths
model2 = smf.ols(formula='Avg_Ann_Deaths ~ All_Poverty + Med_Income  +
All_With + All_Without + Rising + Falling', data=data_mod1).fit()
print(model2.summary())
```

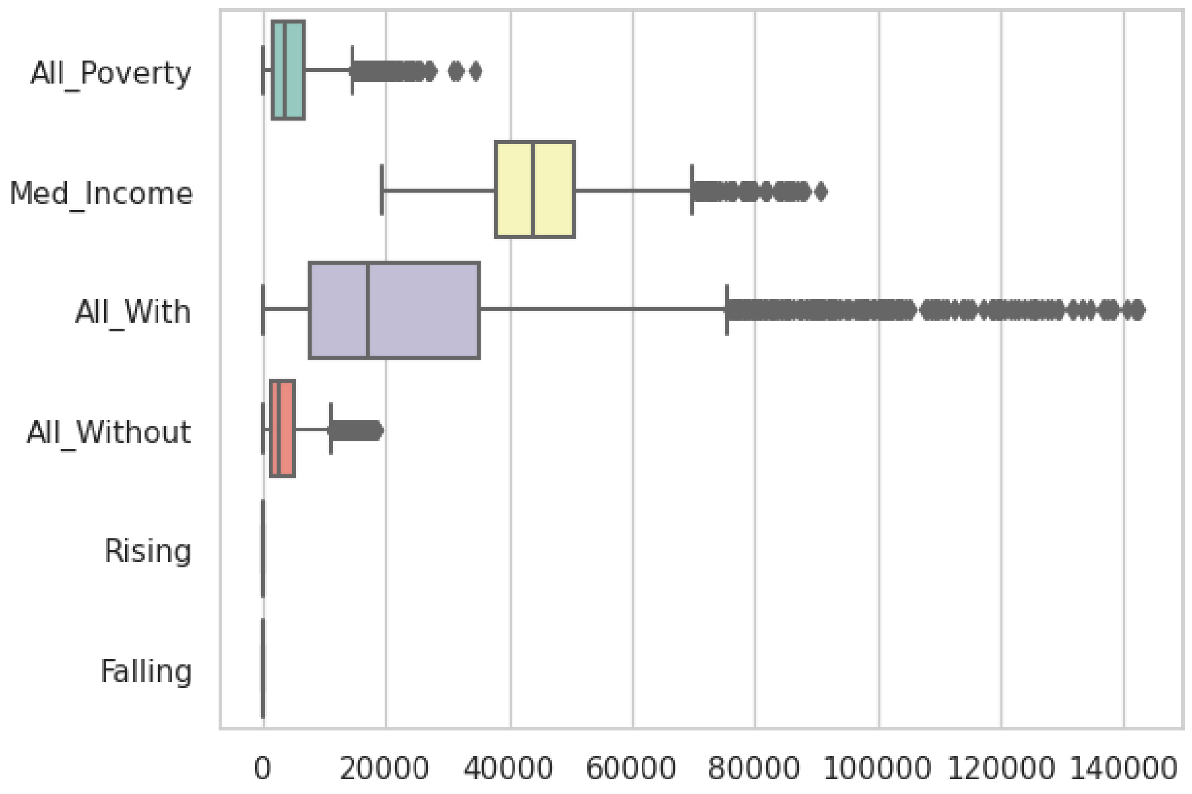
```
                            OLS Regression Results

======================================================================
========
Dep. Variable:         Avg_Ann_Deaths   R-squared:
0.898
Model:                            OLS   Adj. R-squared:
0.898
Method:                 Least Squares   F-statistic:
4089.
Date:                Fri, 01 Sep 2023   Prob (F-statistic):
0.00
Time:                        09:53:58   Log-Likelihood:
-13494.
No. Observations:                2782   AIC:
2.700e+04
Df Residuals:                    2775   BIC:
2.704e+04
Df Model:                           6

Covariance Type:            nonrobust

======================================================================
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
---------
Intercept      32.2407      2.773     11.628      0.000      26.804
37.677
All_Poverty  7.577e-05   5.94e-05      1.276      0.202   -4.07e-05
0.000
Med_Income     -0.0005   6.04e-05     -7.836      0.000      -0.001
-0.000
All_With        0.0005    9.7e-06     54.729      0.000       0.001
0.001
All_Without    -0.0004   4.88e-05     -8.435      0.000      -0.001
-0.000
Rising         -5.6779      4.996     -1.136      0.256     -15.474
4.119
Falling         1.0933      2.481      0.441      0.660      -3.772
5.958
======================================================================
========
Omnibus:                     1576.591   Durbin-Watson:
```

```
1.582
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
187369.073
Skew:                           1.716   Prob(JB):
0.00
Kurtosis:                      43.058   Cond. No.
1.87e+06
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.87e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

```
data_mod1 = df.copy()
for i in ['All_Poverty', 'Med_Income', 'All_With', 'All_Without',
'Rising', 'Falling']:
    #using IQR method
    dat = data_mod1[i]
    Q1 = dat.quantile(0.25)
    Q3 = dat.quantile(0.75)
    IQR = Q3 - Q1    #IQR is interquartile range.

    filter = (dat >= Q1 - 3 * IQR) & (dat <= Q3 + 3 *IQR)
    data_mod1 = data_mod1.loc[filter]
    print(data_mod1.shape)
```

```
(2875, 14)
(2855, 14)
(2730, 14)
(2671, 14)
(2631, 14)
(2527, 14)
```

```
# Boxplot for the filtered data
ax = sns.boxplot(data=data_mod1[['All_Poverty', 'Med_Income',
'All_With', 'All_Without', 'Rising', 'Falling']], orient="h", palette=
'Set3')
```



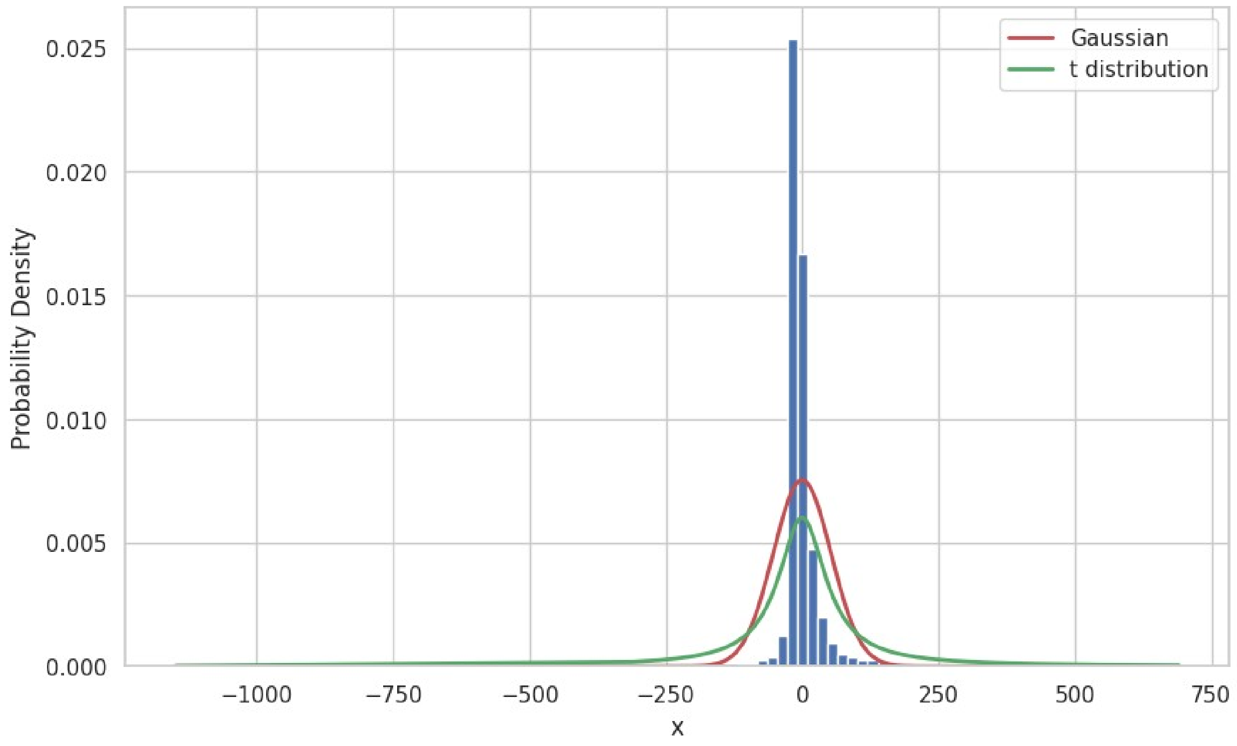
```
fin_df = data_mod1
# Fitting to Average Deaths
model1 = smf.ols(formula='Avg_Ann_Incidence ~Med_Income  + All_With +
Rising + Falling', data=df).fit()
print(model1.summary())

                            OLS Regression Results

======================================================================
========
Dep. Variable:      Avg_Ann_Incidence   R-squared:
0.906
Model:                            OLS   Adj. R-squared:
0.906
Method:                 Least Squares   F-statistic:
7020.
Date:                Fri, 01 Sep 2023   Prob (F-statistic):
0.00
Time:                        10:12:47   Log-Likelihood:
-15759.
No. Observations:                2924   AIC:
3.153e+04
Df Residuals:                    2919   BIC:
3.156e+04
Df Model:                           4
```

```
Covariance Type:            nonrobust
==============================================================================
               coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
Intercept     24.4731      3.875      6.315      0.000      16.874
32.072
Med_Income    -0.0002   8.23e-05     -1.898      0.058      -0.000
5.17e-06
All_With       0.0006   3.84e-06    154.332      0.000       0.001
0.001
Rising        -6.8821      8.159     -0.844      0.399     -22.879
9.115
Falling       16.1143      4.122      3.909      0.000       8.032
24.196
==============================================================================
Omnibus:                     1260.365   Durbin-Watson:
1.646
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
1570186.457
Skew:                           0.370   Prob(JB):
0.00
Kurtosis:                     116.523   Cond. No.
2.42e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 2.42e+06. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

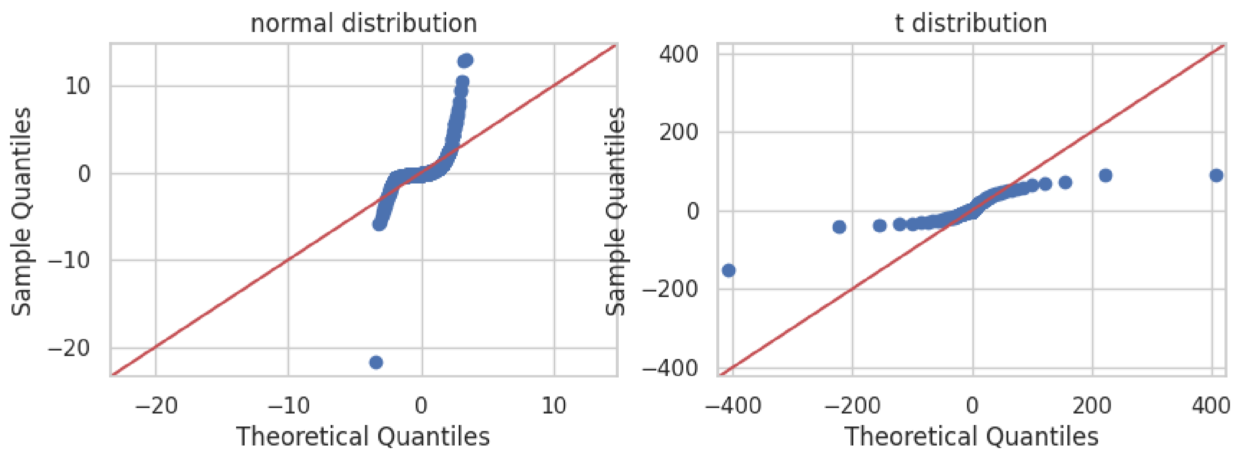
Normality of Errors

```
# histogram superimposed by normal curve
plt.figure(figsize=(10,6))
import scipy.stats as stats
mu = np.mean(model1.resid)
sigma = np.std(model1.resid)
pdf = stats.norm.pdf(sorted(model1.resid), mu, sigma)
pdf2 = stats.t.pdf(sorted(model1.resid), df = 1, loc = mu,
scale=sigma,)
```

```
plt.xlabel('x')
plt.ylabel('Probability Density')
plt.hist(model1.resid, bins=100, density= True)
plt.plot(sorted(model1.resid), pdf, color='r', linewidth=2, label =
'Gaussian')
plt.plot(sorted(model1.resid), pdf2, color='g', linewidth=2, label =
't distribution')
plt.legend()
plt.show()
```



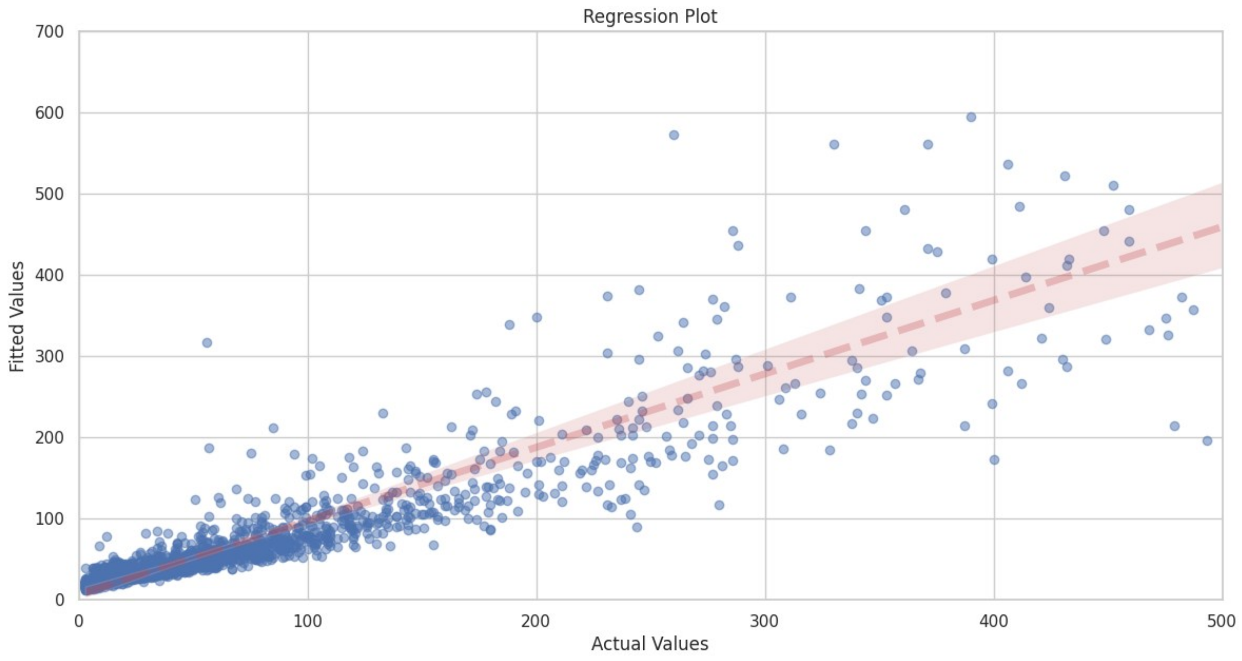
```
# QQplot
fig, [ax1, ax2] = plt.subplots(1,2, figsize=(10,3))
sm.qqplot(model1.resid, stats.norm, fit=True, line='45', ax=ax1)
ax1.set_title("normal distribution")
sm.qqplot(model1.resid, stats.t, fit=True, line='45', ax = ax2)
ax2.set_title("t distribution")

plt.show()
```



As we can see that the QQ plot for the normal distribution is not close to the straight line at the ends and it deviates to the top at the right and to the bottom at the left, we can say that the tails for the residuals are heavier than a normal distribution, hence on following that I tried using a t distribution, which gives a much better QQ plot and hence we can confirm that the residuals come from the t distribution !! Fatter tails suggest we have more number of outliers !

Heteroscedasticity

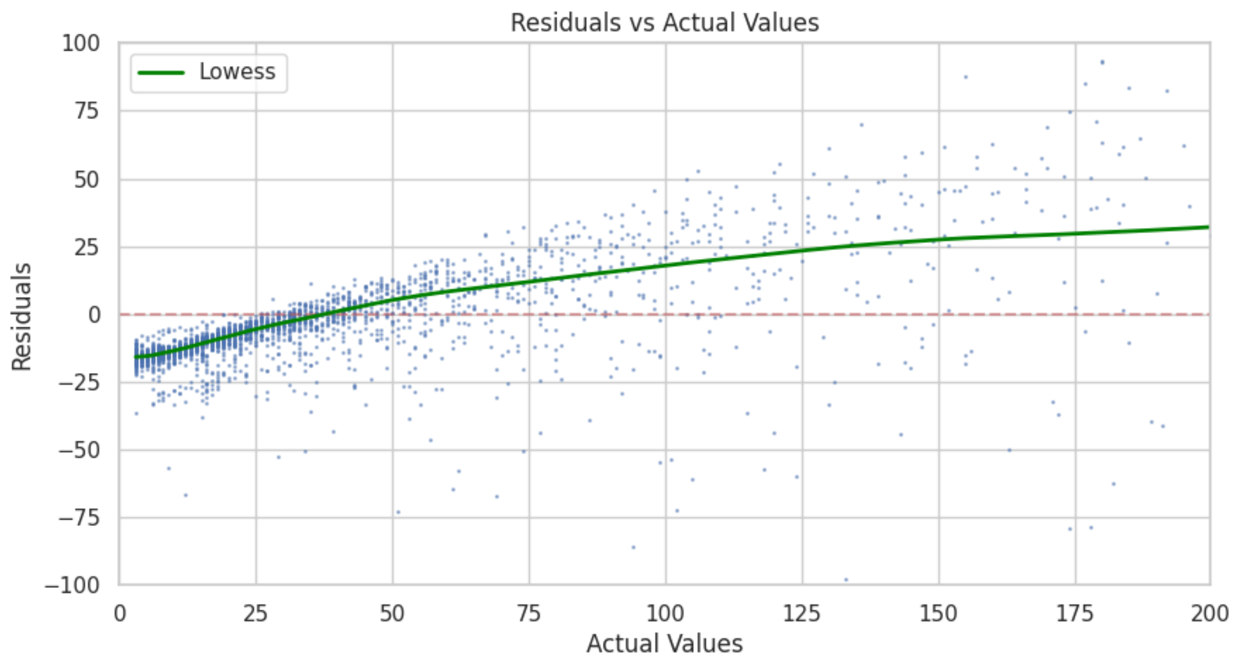
```
plt.figure(figsize=(14,7))
sns.regplot(x = X['Avg_Ann_Incidence'], y = model1.fittedvalues,
line_kws={'color':'r', 'alpha':0.3,
                                                   'linestyle':'--',
'linewidth':5},
            scatter_kws={'alpha':0.5})
plt.xlim(0,500)
plt.ylim(0,700)
plt.title('Regression Plot')
plt.xlabel('Actual Values')
plt.ylabel('Fitted Values')
plt.show()
print("Pearson R: ", stats.pearsonr(model1.fittedvalues,
X['Avg_Ann_Incidence']))
```



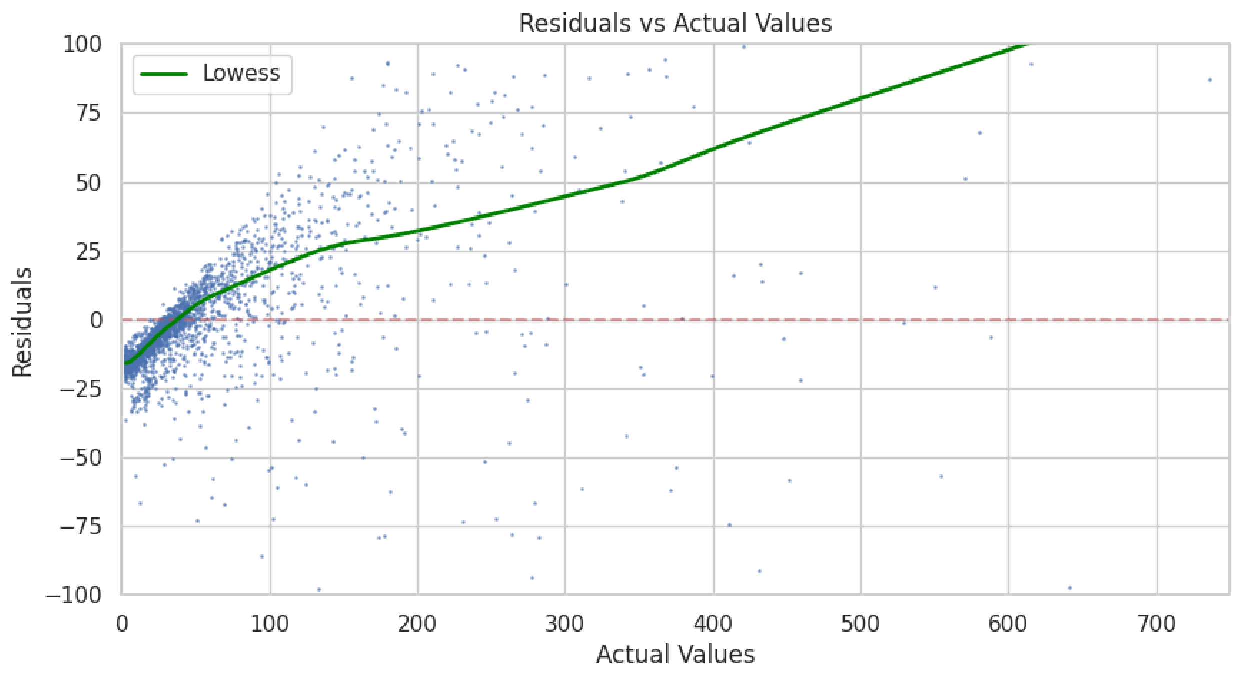
```
Pearson R:  PearsonRResult(statistic=0.9517556839499244, pvalue=0.0)
```

We can see that the point do lie close to our line and follow the trend which is also confirmed by the high Pearson Coefficient for Correlation !

```
y = X['Avg_Ann_Incidence']

# plot actual values versus residuals
from statsmodels.nonparametric.smoothers_lowess import lowess
ys = lowess(model1.resid.values, y, frac=0.2)
ys = pd.DataFrame(ys, index=range(len(ys)), columns=['a', 'b'])
ys = ys.sort_values(by='a')

fig, ax = plt.subplots(figsize=(10,5))
ax.set_xlim([0,200])
ax.set_ylim([-100,100])
plt.title('Residuals vs Actual Values')
plt.scatter(y, model1.resid, alpha=0.5, s=1)
plt.axhline(y=0, color='r', linestyle="--", alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Residuals")

plt.plot(ys.a, ys.b, c='green', linewidth=2, label="Lowess")
plt.legend()
plt.show()
print("Pearson R:", stats.pearsonr(y, model1.resid))
```



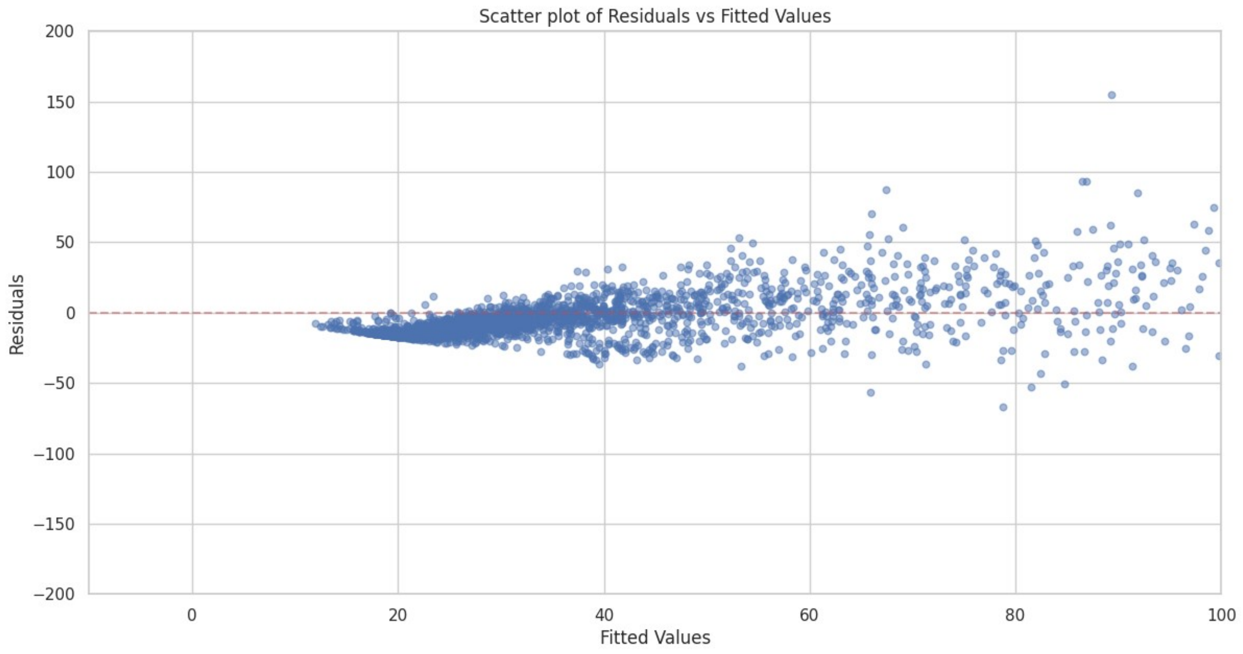
```
Pearson R: PearsonRResult(statistic=0.3068568364381449,
pvalue=8.553812422161798e-65)

#Same plot from a higher view
from statsmodels.nonparametric.smoothers_lowess import lowess
ys = lowess(model1.resid.values, y, frac=0.2)
ys = pd.DataFrame(ys, index=range(len(ys)), columns=['a', 'b'])
ys = ys.sort_values(by='a')

fig, ax = plt.subplots(figsize=(10,5))
ax.set_xlim([0,750])
ax.set_ylim([-100,100])
plt.title('Residuals vs Actual Values')
plt.scatter(y, model1.resid, alpha=0.5, s=1)
plt.axhline(y=0, color='r', linestyle="--", alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Residuals")

plt.plot(ys.a, ys.b, c='green', linewidth=2, label="Lowess")
plt.legend()
plt.show()
print("Pearson R:", stats.pearsonr(y, model1.resid))
```



```
Pearson R: PearsonRResult(statistic=0.3068568364381449,
pvalue=8.553812422161798e-65)
```

From the above curve we can see that the Lowess curve is below the y=0 line for lower values which mean our model is overpredicting these values and then goes to the upper side for most of the residuals are about the y=0 line which means our model is underpredicting these values !

```python
result = model1

# plot actual values versus residuals
plt.figure(figsize=(14,7))
plt.title('Scatter plot of Residuals vs Fitted Values')
plt.scatter(y=result.resid, x=result.fittedvalues, alpha=0.5, s=22)
plt.axhline(y=0, color='r', linestyle="--", alpha=0.5)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.xlim(-10, 100)
plt.ylim(-200, 200)
plt.show()
```



We observe a distinctive cone-shaped pattern in the residual plot, which is a characteristic indication of heteroscedasticity. This phenomenon implies that as the fitted values increase, the variance of the residuals also increases. Consequently, our model is significantly affected by pronounced heteroscedasticity.

Heteroscedasticity refers to a systematic alteration in the dispersion of residuals across the spectrum of observed values. The challenge arises because ordinary least squares (OLS) regression assumes a consistent variance of residuals (homoscedasticity) across the data. Heteroscedasticity, alternatively spelled heteroskedasticity, is more common in datasets that exhibit considerable variation between the smallest and largest observed values. While several factors can contribute to the existence of heteroscedasticity, a prevalent explanation is that the variability of errors changes in proportion to a certain factor. This factor might even be a variable present in the model.

Although heteroscedasticity itself does not introduce bias in the coefficient estimations, it does reduce their precision. Decreased precision elevates the likelihood of coefficient estimates deviating further from the true population values. Notably, heteroscedasticity tends to yield p-values that appear smaller than they actually should be. This discrepancy emerges because heteroscedasticity inflates the variance of coefficient estimates, yet the OLS procedure remains oblivious to this inflation. Consequently, OLS calculates t-values and F-values using an underestimated level of variance. This issue can lead to the erroneous conclusion that a particular model term holds statistical significance when, in reality, it might not be statistically significant.