CaseStudy1and2_18BCS6033

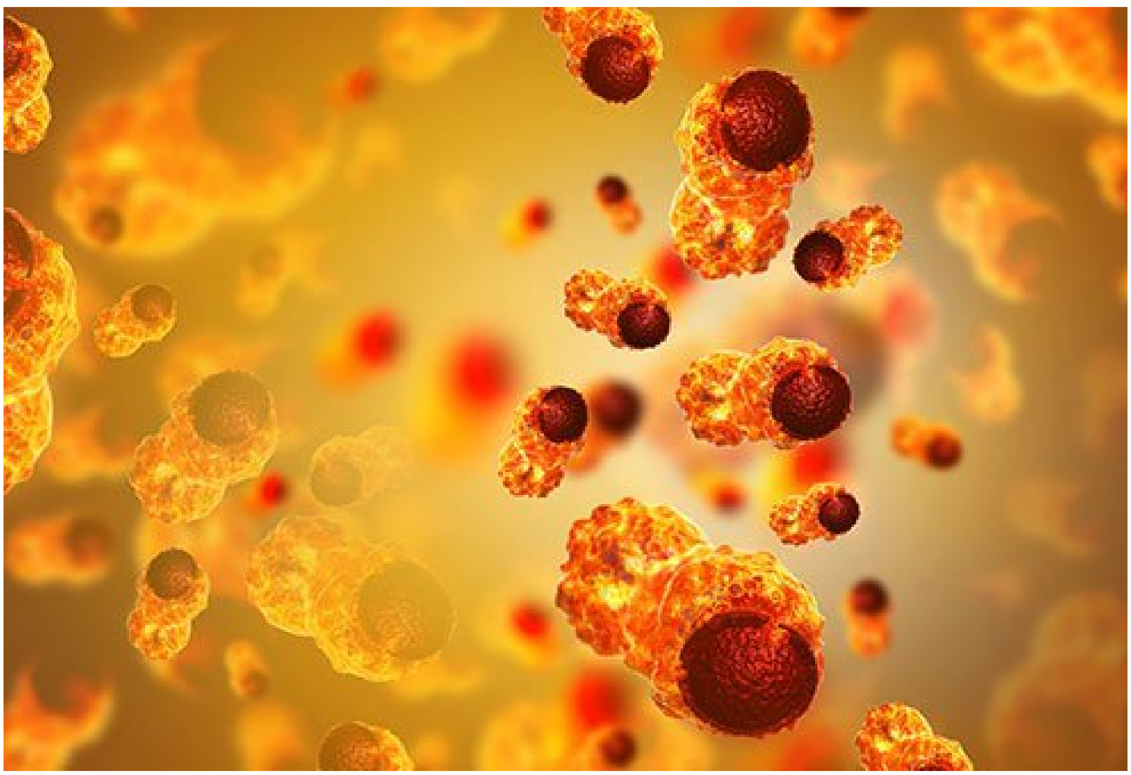
December 10, 2020

1 Case Study on Prediction of Cancer Mortality Rates for US counties.

1.1 Karan Trehan

1.1.1 18BCS6033

1.1.2 18AITAIML1 - Group B



We will be Building a multivariate Ordinary Least Squares regression model to predict "TARGET_deathRate".

[1]:
```
# Supress Warnings
import warnings
warnings.filterwarnings('ignore')
```

1.2 Importing the Required Libraries

[2]:
```
#For Data Handling
import numpy as np
import pandas as pd

#For Data Visualization/Exploratory Analysis
from matplotlib import pyplot as plt
import seaborn as sns

#For Statistical Calculations
import scipy.stats as st

#For Regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_selection import RFE
from sklearn.svm import SVR

%matplotlib inline
sns.set(color_codes=True)
```

1.3 Reading and Understanding the Data

Let's start with the following steps:

1. Importing data using the pandas library
2. Understanding the structure of the data

[3]:
```
df = pd.read_csv('https://query.data.world/s/
↪xlh353wvypzveoxm7h4u4c6hnucftk',encoding='iso-8859-1')
```

[4]:
```
#Checking the number of rows and columns in the Dataset
df.shape
```

[4]:
```
(3047, 34)
```

[5]:
```
#Checking information about the Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3047 entries, 0 to 3046
Data columns (total 34 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   avgAnnCount              3047 non-null   float64
 1   avgDeathsPerYear         3047 non-null   int64
```

```
 2   TARGET_deathRate         3047 non-null   float64
 3   incidenceRate            3047 non-null   float64
 4   medIncome                3047 non-null   int64
 5   popEst2015               3047 non-null   int64
 6   povertyPercent           3047 non-null   float64
 7   studyPerCap              3047 non-null   float64
 8   binnedInc                3047 non-null   object
 9   MedianAge                3047 non-null   float64
 10  MedianAgeMale            3047 non-null   float64
 11  MedianAgeFemale          3047 non-null   float64
 12  Geography                3047 non-null   object
 13  AvgHouseholdSize         3047 non-null   float64
 14  PercentMarried           3047 non-null   float64
 15  PctNoHS18_24             3047 non-null   float64
 16  PctHS18_24               3047 non-null   float64
 17  PctSomeCol18_24          762 non-null    float64
 18  PctBachDeg18_24          3047 non-null   float64
 19  PctHS25_Over             3047 non-null   float64
 20  PctBachDeg25_Over        3047 non-null   float64
 21  PctEmployed16_Over       2895 non-null   float64
 22  PctUnemployed16_Over     3047 non-null   float64
 23  PctPrivateCoverage       3047 non-null   float64
 24  PctPrivateCoverageAlone  2438 non-null   float64
 25  PctEmpPrivCoverage       3047 non-null   float64
 26  PctPublicCoverage        3047 non-null   float64
 27  PctPublicCoverageAlone   3047 non-null   float64
 28  PctWhite                 3047 non-null   float64
 29  PctBlack                 3047 non-null   float64
 30  PctAsian                 3047 non-null   float64
 31  PctOtherRace             3047 non-null   float64
 32  PctMarriedHouseholds     3047 non-null   float64
 33  BirthRate                3047 non-null   float64
dtypes: float64(29), int64(3), object(2)
memory usage: 809.5+ KB
```

There are 34 Columns in the Dataset.

- 3 Columns contain Missing Values
- 2 Columns are Object Type and the rest contain Numerical Values

[6]:
```
#Viewing the Statistical Measures/Details of the Dataset
df.describe()
```

[6]:
```
        avgAnnCount  avgDeathsPerYear  TARGET_deathRate  incidenceRate  \
count   3047.000000       3047.000000       3047.000000    3047.000000
mean     606.338544        185.965868        178.664063     448.268586
std     1416.356223        504.134286         27.751511      54.560733
min        6.000000          3.000000         59.700000     201.300000
```

```
25%       76.000000       28.000000        161.200000     420.300000
50%      171.000000       61.000000        178.100000     453.549422
75%      518.000000      149.000000        195.200000     480.850000
max    38150.000000    14010.000000        362.800000    1206.900000

            medIncome    popEst2015  povertyPercent  studyPerCap    MedianAge  \
count     3047.000000  3.047000e+03     3047.000000  3047.000000  3047.000000
mean     47063.281917  1.026374e+05       16.878175   155.399415    45.272333
std      12040.090836  3.290592e+05        6.409087   529.628366    45.304480
min      22640.000000  8.270000e+02        3.200000     0.000000    22.300000
25%      38882.500000  1.168400e+04       12.150000     0.000000    37.700000
50%      45207.000000  2.664300e+04       15.900000     0.000000    41.000000
75%      52492.000000  6.867100e+04       20.400000    83.650776    44.000000
max     125635.000000  1.017029e+07       47.400000  9762.308998   624.000000

       MedianAgeMale  ...  PctPrivateCoverageAlone  PctEmpPrivCoverage  \
count    3047.000000  ...              2438.000000         3047.000000
mean       39.570725  ...                48.453774           41.196324
std         5.226017  ...                10.083006            9.447687
min        22.400000  ...                15.700000           13.500000
25%        36.350000  ...                41.000000           34.500000
50%        39.600000  ...                48.700000           41.100000
75%        42.500000  ...                55.600000           47.700000
max        64.700000  ...                78.900000           70.700000

       PctPublicCoverage  PctPublicCoverageAlone     PctWhite     PctBlack  \
count        3047.000000             3047.000000  3047.000000  3047.000000
mean           36.252642               19.240072    83.645286     9.107978
std             7.841741                6.113041    16.380025    14.534538
min            11.200000                2.600000    10.199155     0.000000
25%            30.900000               14.850000    77.296180     0.620675
50%            36.300000               18.800000    90.059774     2.247576
75%            41.550000               23.100000    95.451693    10.509732
max            65.100000               46.600000   100.000000    85.947799

          PctAsian  PctOtherRace  PctMarriedHouseholds    BirthRate
count  3047.000000   3047.000000           3047.000000  3047.000000
mean      1.253965      1.983523             51.243872     5.640306
std       2.610276      3.517710              6.572814     1.985816
min       0.000000      0.000000             22.992490     0.000000
25%       0.254199      0.295172             47.763063     4.521419
50%       0.549812      0.826185             51.669941     5.381478
75%       1.221037      2.177960             55.395132     6.493677
max      42.619425     41.930251             78.075397    21.326165

[8 rows x 32 columns]
```

Since there are absurd values in some of the Features, it implies that the dataset contains outliers.

For Example : The Maximum Age in 'MedianAge' Column is 624 which cannot be possible as no-one known to be alive for such a long period of time. We'll treat these outliers later.

1.4 Data Preparation and Pre-processing

[7]:
```
#Checking the 'binnedInc' Column because it has Numeric values stored in string␣
 ↪manner.
df['binnedInc']
```

[7]:
```
0        (61494.5, 125635]
1       (48021.6, 51046.4]
2       (48021.6, 51046.4]
3         (42724.4, 45201]
4       (48021.6, 51046.4]
                ...
3042      (45201, 48021.6]
3043    (48021.6, 51046.4]
3044    (51046.4, 54545.6]
3045    (48021.6, 51046.4]
3046    (40362.7, 42724.4]
Name: binnedInc, Length: 3047, dtype: object
```

[8]:
```
#Grouping the Dataframe based on 'binnedInc'
binnedInc = df.groupby('binnedInc')
binnedInc.first()
```

[8]:
```
                    avgAnnCount  avgDeathsPerYear  TARGET_deathRate  \
binnedInc
(34218.1, 37413.8]         94.0                41             189.7
(37413.8, 40362.7]        250.0                97             175.9
(40362.7, 42724.4]         88.0                36             190.5
(42724.4, 45201]          427.0               202             194.8
(45201, 48021.6]           72.0                32             214.7
(48021.6, 51046.4]        173.0                70             161.3
(51046.4, 54545.6]        428.0               152             176.0
(54545.6, 61494.5]       4025.0              1380             177.8
(61494.5, 125635]        1397.0               469             164.9
[22640, 34218.1]           80.0                40             196.3

                    incidenceRate  medIncome  popEst2015  povertyPercent  \
binnedInc
(34218.1, 37413.8]          445.2      35615       16704            21.5
(37413.8, 40362.7]          461.8      37782       41516            23.2
(40362.7, 42724.4]          459.4      42579       13088            22.3
(42724.4, 45201]            430.4      44243       75882            17.1
(45201, 48021.6]            502.0      46383        9982            17.7
(48021.6, 51046.4]          411.6      48127       43269            18.6
```

```
(51046.4, 54545.6]          505.4      52313       61023            15.6
(54545.6, 61494.5]          510.9      60397      843954            13.1
(61494.5, 125635]           489.8      61898      260131            11.2
[22640, 34218.1]            396.6      33817       14415            22.2

                    studyPerCap  MedianAge  MedianAgeMale  ...  \
binnedInc                                                  ...
(34218.1, 37413.8]     0.000000       41.5           40.9  ...
(37413.8, 40362.7]     0.000000       42.6           42.2  ...
(40362.7, 42724.4]     0.000000       49.3           48.4  ...
(42724.4, 45201]     342.637253       42.8           42.2  ...
(45201, 48021.6]       0.000000       45.2           45.1  ...
(48021.6, 51046.4]    23.111234       33.0           32.2  ...
(51046.4, 54545.6]   180.259902       45.4           43.5  ...
(54545.6, 61494.5]   427.748432       35.8           34.7  ...
(61494.5, 125635]    499.748204       39.3           36.9  ...
[22640, 34218.1]       0.000000       44.5           43.7  ...

                    PctPrivateCoverageAlone PctEmpPrivCoverage  \
binnedInc
(34218.1, 37413.8]                     40.1               36.5
(37413.8, 40362.7]                     35.0               28.3
(40362.7, 42724.4]                     37.8               29.9
(42724.4, 45201]                       40.3               35.0
(45201, 48021.6]                       38.9               37.0
(48021.6, 51046.4]                     53.8               43.6
(51046.4, 54545.6]                     38.8               32.6
(54545.6, 61494.5]                     59.4               44.4
(61494.5, 125635]                      61.6               41.6
[22640, 34218.1]                       41.9               38.9

                    PctPublicCoverage  PctPublicCoverageAlone   PctWhite  \
binnedInc
(34218.1, 37413.8]               44.8                    26.4  96.844181
(37413.8, 40362.7]               46.4                    28.7  75.106455
(40362.7, 42724.4]               48.1                    26.6  91.787477
(42724.4, 45201]                 45.3                    25.0  91.744686
(45201, 48021.6]                 46.9                    24.3  98.234714
(48021.6, 51046.4]               31.1                    15.3  89.228509
(51046.4, 54545.6]               43.2                    20.2  84.882631
(54545.6, 61494.5]               31.4                    16.5  74.729668
(61494.5, 125635]                32.9                    14.0  81.780529
[22640, 34218.1]                 43.4                    25.4  97.912346

                    PctBlack  PctAsian  PctOtherRace  PctMarriedHouseholds  \
binnedInc
(34218.1, 37413.8]  0.836770  0.376547      0.029885             55.288859
```

```
(37413.8, 40362.7]  0.616955  0.866157      8.356721             51.013900
(40362.7, 42724.4]  0.185071  0.208205      0.616903             53.446998
(42724.4, 45201]    0.782626  1.161359      1.362643             51.021514
(45201, 48021.6]    0.473373  0.000000      0.029586             49.746322
(48021.6, 51046.4]  0.969102  2.246233      3.741352             45.372500
(51046.4, 54545.6]  1.653205  1.538057      3.314635             51.220360
(54545.6, 61494.5]  6.710854  6.041472      2.699184             50.063573
(61494.5, 125635]   2.594728  4.821857      1.843479             52.856076
[22640, 34218.1]    0.497719  0.000000      0.000000             53.272695

                    BirthRate
binnedInc
(34218.1, 37413.8]   2.292861
(37413.8, 40362.7]   4.204317
(40362.7, 42724.4]   5.587583
(42724.4, 45201]     4.603841
(45201, 48021.6]     6.267540
(48021.6, 51046.4]   4.333096
(51046.4, 54545.6]   4.964476
(54545.6, 61494.5]   5.533430
(61494.5, 125635]    6.118831
[22640, 34218.1]     5.469020

[10 rows x 33 columns]
```

- We can see that in 'binnedInc', we have the ranges of Income of every county.
- We can also infer that by removing '(' ; '[' ; ']' and ',' and then averaging the two values obtained, the column can be converted to float64 type.

[9]:
```
#Converting the String values in 'binnedInc' to Float type by splitting the␣
↪values and averaging them
df['binnedInc']=df['binnedInc'].str.replace('(','')
df['binnedInc']=df['binnedInc'].str.replace('[','')
df['binnedInc']=df['binnedInc'].str.replace(']','')

a=df['binnedInc'].str.split(',',expand=True).astype(float)
b=(a[0]+a[1])/2
df['binnedInc']=b
df.head()
```

[9]:
```
   avgAnnCount  avgDeathsPerYear  TARGET_deathRate  incidenceRate  medIncome  \
0       1397.0               469             164.9          489.8      61898
1        173.0                70             161.3          411.6      48127
2        102.0                50             174.7          349.7      49348
3        427.0               202             194.8          430.4      44243
4         57.0                26             144.4          350.1      49955

   popEst2015  povertyPercent  studyPerCap  binnedInc  MedianAge  ...  \
```

```
0      260131            11.2  499.748204  93564.75        39.3  ...
1       43269            18.6   23.111234  49534.00        33.0  ...
2       21026            14.6   47.560164  49534.00        45.0  ...
3       75882            17.1  342.637253  43962.70        42.8  ...
4       10321            12.5    0.000000  49534.00        48.3  ...

   PctPrivateCoverageAlone  PctEmpPrivCoverage PctPublicCoverage  \
0                      NaN                41.6              32.9
1                     53.8                43.6              31.1
2                     43.5                34.9              42.1
3                     40.3                35.0              45.3
4                     43.9                35.1              44.0

   PctPublicCoverageAlone   PctWhite  PctBlack  PctAsian  PctOtherRace  \
0                    14.0  81.780529  2.594728  4.821857      1.843479
1                    15.3  89.228509  0.969102  2.246233      3.741352
2                    21.1  90.922190  0.739673  0.465898      2.747358
3                    25.0  91.744686  0.782626  1.161359      1.362643
4                    22.7  94.104024  0.270192  0.665830      0.492135

   PctMarriedHouseholds  BirthRate
0             52.856076   6.118831
1             45.372500   4.333096
2             54.444868   3.729488
3             51.021514   4.603841
4             54.027460   6.796657

[5 rows x 34 columns]
```

[10]:
```
#Checking the 'Geography' Column for duplicate values.
df['Geography'].is_unique
```

[10]:
```
True
```

We can observe that the Geography column has Unique Values and it will be redundant to create Dummy Variables for it (it will increase the complexity of the Model), so we will drop the column.

[11]:
```
df = df.drop(columns = ['Geography'],axis=1)
```

1.5 Univariate Analysis of Columns

[12]:
```
#For plotting the Distribution Plot of all the columns, we temporarily create a␣
↪copy of our main dataframe.
#In the copy dataframe , 'df2' , we impute all the missing values, so that the␣
↪Distribution Plots can be plotted for all columns
df2 = df.copy()
```

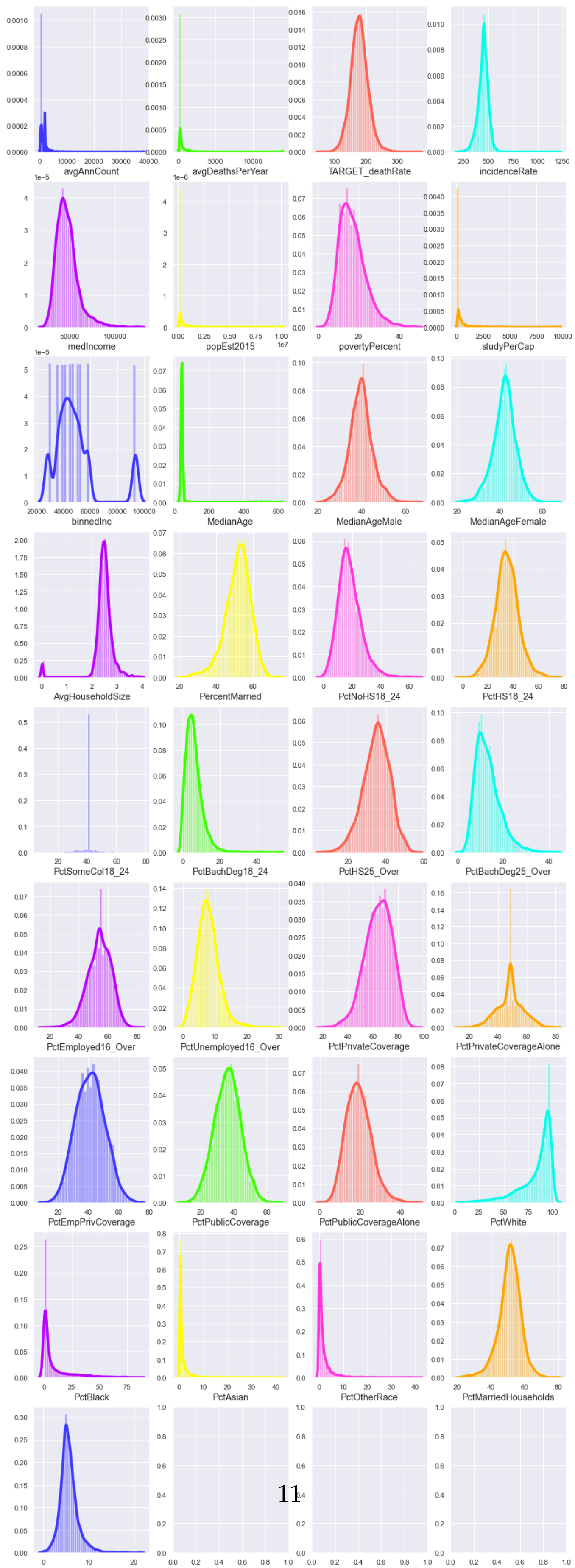
```
df2['PctSomeCol18_24'] = df2['PctSomeCol18_24'].fillna(df2['PctSomeCol18_24'].
 ↪median())
df2['PctEmployed16_Over'] = df2['PctEmployed16_Over'].
 ↪fillna(df2['PctEmployed16_Over'].median())
df2['PctPrivateCoverageAlone'] = df2['PctPrivateCoverageAlone'].
 ↪fillna(df2['PctPrivateCoverageAlone'].median())
```

[13]:
```
#Defining a function 'distributionPlot()' which can be used for plotting the␣
 ↪distribution plots for all columns of the Dataframe
#passed as an argument

def distributionPlot(df,Title):
    # 'n' will store the number of Columns
    n = df.shape[1]

    #We will plot the Graphs in a 8x4 or a 9x4 Subplots
    #Rows will be 8 when the number of columns passed will be 32
    #Rows will be 9 when the number of columns passed will be > 32
    if(n%4==0):
        rows = 8
    else:
        rows = 9
    cols = 4

    #Creating subplots
    fig, axs = plt.subplots(rows,cols, figsize = (15, 50))
    fig.subplots_adjust(top=0.8)

    #Defining the color schemes
    colors = ['#3E37FF', '#3BFF00', '#FF6050', '#00FFEA', '#BA00FF', '#FFFE00',␣
 ↪'#FF36DD','orange'] # to set color

    #Looping through the DataFrame and plotting for each Column
    k=0
    j=0
    for i, var in enumerate(df.columns.values):

        if (j%4==0 and j!=0):
            k+=1
        if (j%8==0 and j!=0):
            j=0

        sns.distplot(df[var],ax=axs[k,␣
 ↪i-int(k*4)],color=colors[j],kde_kws=dict(linewidth=4),hist=True)
        axs[k, i-int(k*4)].set_xlabel(var, fontsize = 'large')
        j+=1
```

```
    #Providing Tiltle for the Plot
    plt.suptitle(Title, fontsize = 'xx-large',y=0.815)
    plt.show()
```

[14]:
```
#Plotting Distribution Plots
distributionPlot(df2,"Distribution Plots")
```

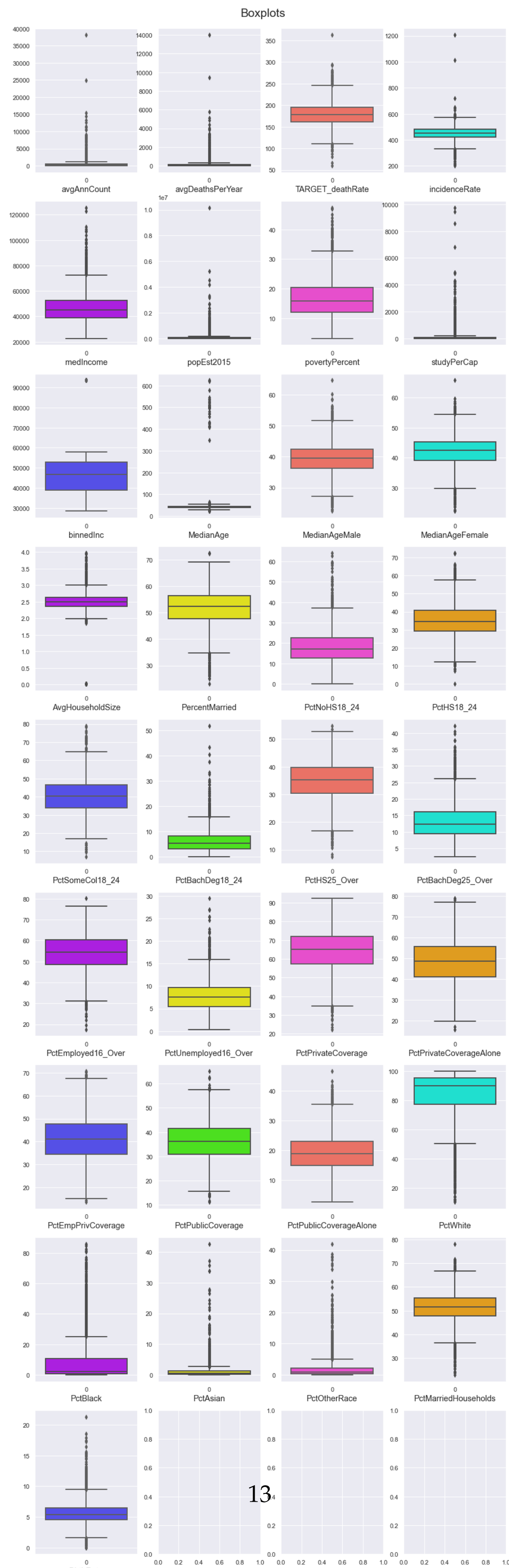
Distribution Plots



We can Infer from the above plots that many Features are both Positively and Negatively Skewed.

[15]:
```
#Defining a function 'boxPlot()' which can be used for plotting the box-plots␣
↪for all columns of the Dataframe passed as
#an argument
def boxPlot(df,Title):
    # 'n' will store the number of Columns
    n = df.shape[1]

    #We will plot the Graphs in a 8x4 or a 9x4 Subplots
    #Rows will be 8 when the number of columns passed will be 32
    #Rows will be 9 when the number of columns passed will be > 32
    if(n%4==0):
        rows = 8
    else:
        rows = 9
    cols = 4

    #Creating subplots
    fig, axs = plt.subplots(rows,cols, figsize = (15, 50))

    #Defining the color schemes
    colors = ['#3E37FF', '#3BFF00', '#FF6050', '#00FFEA', '#BA00FF', '#FFFE00',␣
↪'#FF36DD','orange'] # to set color

    #Looping through the DataFrame and plotting for each Column
    k=0
    j=0
    for i, var in enumerate(df.columns.values):
        if (j%4==0 and j!=0):
            k+=1
        if (j%8==0 and j!=0):
            j=0

        sns.boxplot(data = df[var] , ax = axs[k, i-int(k*4)], color=␣
↪colors[j],linewidth=2)
        axs[k, i-int(k*4)].set_xlabel(var, fontsize = 'large')
        j+=1

    #Providing Tiltle for the Plot
    plt.suptitle(Title, fontsize = 'xx-large',y=0.89)
    plt.show()
```

[16]:
```
#Plotting Boxplots Plots
boxPlot(df,Title = "Boxplots")
```


Boxplots
avgAnnCount
avgDeathsPerYear
TARGET_deathRate
incidenceRate
medIncome
popEst2015
povertyPercent
studyPerCap
binnedInc
MedianAge
MedianAgeMale
MedianAgeFemale
AvgHouseholdSize
PercentMarried
PctNoHS18_24
PctHS18_24
PctSomeCol18_24
PctBachDeg18_24
PctHS25_Over
PctBachDeg25_Over
PctEmployed16_Over
PctUnemployed16_Over
PctPrivateCoverage
PctPrivateCoverageAlone
PctEmpPrivCoverage
PctPublicCoverage
PctPublicCoverageAlone
PctWhite
PctBlack
PctAsian
PctOtherRace
PctMarriedHouseholds
BirthRate

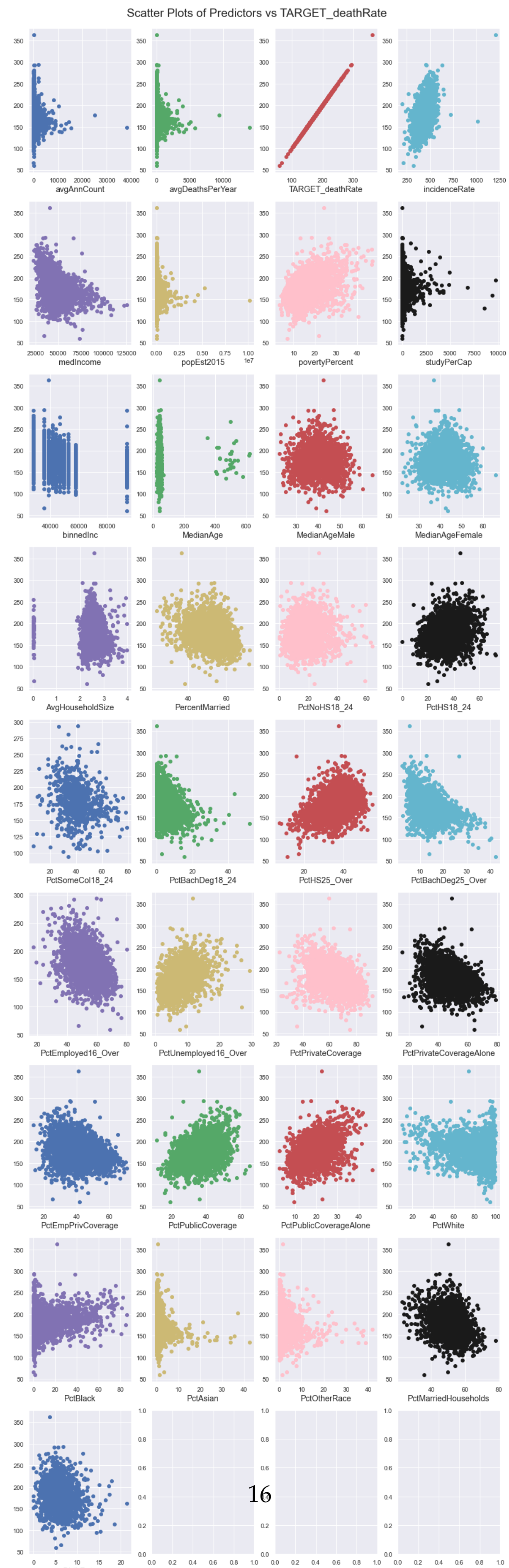
We can Observe that many Features contain Outliers, including the Target Variable

2 Bi-Variate Analysis

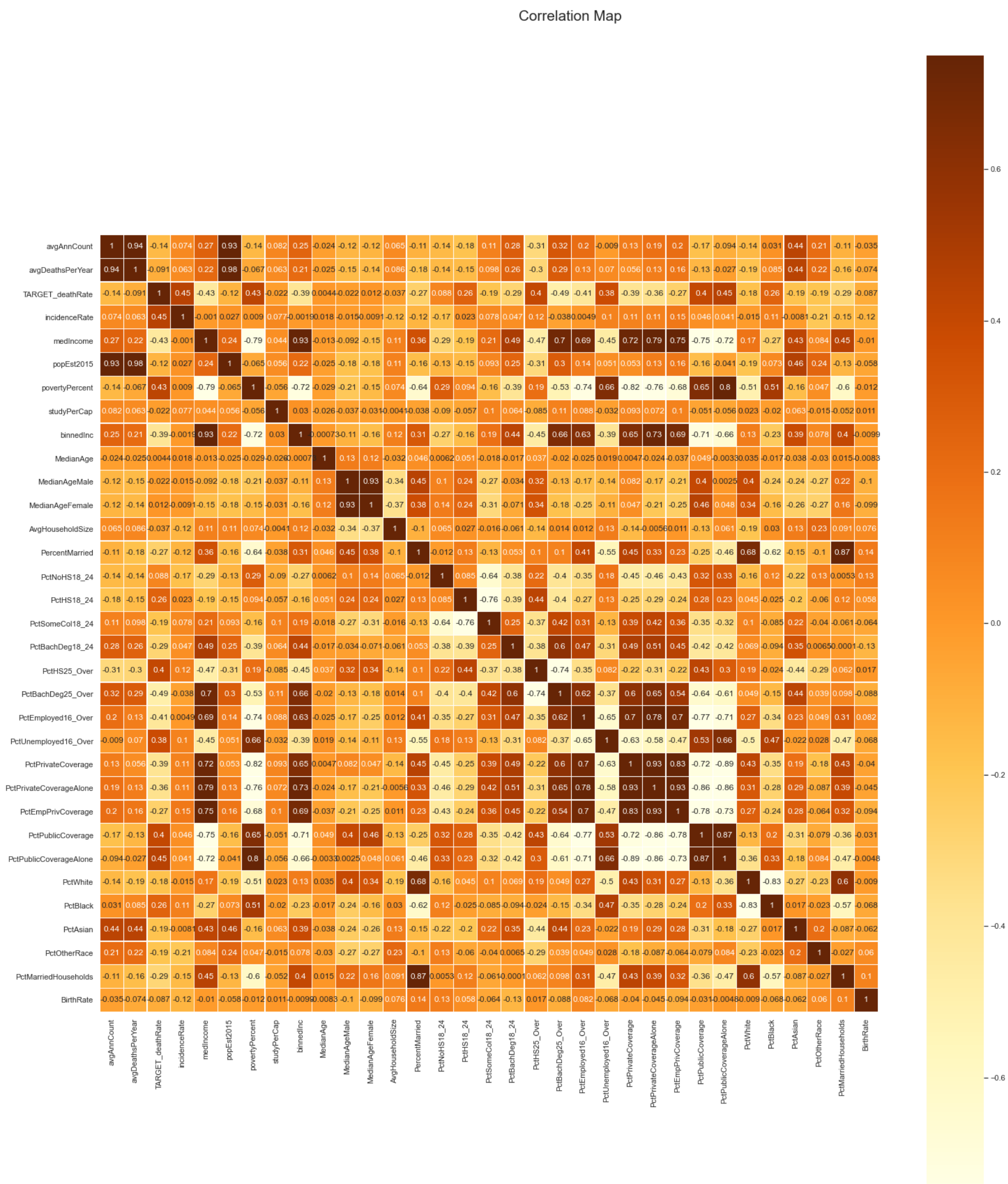
[17]:
```
#Defining a function 'scatterPlot()' which can be used for plotting the␣
↪scatter-plots for all columns v/s 'TARGET_deathRate'
#(dependent Variable) of the Dataframe passed as an argument

def scatterPlot(df,Title):
    # 'n' will store the number of Columns
    n = df.shape[1]

    #We will plot the Graphs in a 8x4 or a 9x4 Subplots
    #Rows will be 8 when the number of columns passed will be 32
    #Rows will be 9 when the number of columns passed will be > 32
    if(n%4==0):
        rows = 8
    else:
        rows = 9
    cols = 4

    #Creating subplots
    fig, axs = plt.subplots(rows,cols, figsize = (15, 50))

    #Defining the color schemes
    colors = ['b', 'g', 'r', 'c', 'm', 'y', 'pink','k'] # to set color

    #Looping through the DataFrame and plotting for each Column
    k=0
    j=0
    for i, var in enumerate(df.columns.values):
        if (j%4==0 and j!=0):
            k+=1
        if (j%8==0 and j!=0):
            j=0

        axs[k, i-int(k*4)].scatter(df[var], df["TARGET_deathRate"], color =␣
↪colors[j])
        axs[k, i-int(k*4)].set_xlabel(var, fontsize = 'large')
        j+=1

    #Providing Tiltle for the Plot
    plt.suptitle(Title, fontsize = 'xx-large',y=0.89)
    plt.show()
```

[18]:

```
#Plotting Scatter Plots v/s 'TARGET_deathRate' (dependent Variable)
scatterPlot(df,"Scatter Plots of Predictors vs TARGET_deathRate")
```


Scatter Plots of Predictors vs TARGET_deathRate
avgAnnCount
avgDeathsPerYear
TARGET_deathRate
incidenceRate
medIncome
popEst2015
povertyPercent
studyPerCap
binnedInc
MedianAge
MedianAgeMale
MedianAgeFemale
AvgHouseholdSize
PercentMarried
PctNoHS18_24
PctHS18_24
PctSomeCol18_24
PctBachDeg18_24
PctHS25_Over
PctBachDeg25_Over
PctEmployed16_Over
PctUnemployed16_Over
PctPrivateCoverage
PctPrivateCoverageAlone
PctEmpPrivCoverage
PctPublicCoverage
PctPublicCoverageAlone
PctWhite
PctBlack
PctAsian
PctOtherRace
PctMarriedHouseholds
BirthRate

```
[19]: #Plotting the Correlation Map to check correlation between the columns and w.r.t.
      ↪ 'TARGET_deathRate'
      plt.figure(figsize=(25,30))
      sns.heatmap(df.corr(),cmap='YlOrBr',linewidths = 0.25, square = True, annot =␣
      ↪True, vmin=-0.75, vmax=0.75)
      plt.suptitle("Correlation Map", fontsize = 'xx-large',y=0.91)
      plt.show()
```



It can be observed that Multiple Predictor Variables have high Correlation implying that Multicollinearity exists in the Dataset.

2.1 Checking for Missing and Duplicated Values

[20]:
```
#Using duplicated() method to check for Duplicate Values
df[df.duplicated()].count()
```

[20]:
```
avgAnnCount                0
avgDeathsPerYear           0
TARGET_deathRate           0
incidenceRate              0
medIncome                  0
popEst2015                 0
povertyPercent             0
studyPerCap                0
binnedInc                  0
MedianAge                  0
MedianAgeMale              0
MedianAgeFemale            0
AvgHouseholdSize           0
PercentMarried             0
PctNoHS18_24               0
PctHS18_24                 0
PctSomeCol18_24            0
PctBachDeg18_24            0
PctHS25_Over               0
PctBachDeg25_Over          0
PctEmployed16_Over         0
PctUnemployed16_Over       0
PctPrivateCoverage         0
PctPrivateCoverageAlone    0
PctEmpPrivCoverage         0
PctPublicCoverage          0
PctPublicCoverageAlone     0
PctWhite                   0
PctBlack                   0
PctAsian                   0
PctOtherRace               0
PctMarriedHouseholds       0
BirthRate                  0
dtype: int64
```

This implies that there are no duplicated rows in the DataFrame.

[21]:
```
#Checking the Percentage of Columns having Missing Values
round(df.isnull().sum()/df.shape[0]*100,2)
```

[21]:
```
avgAnnCount                 0.00
avgDeathsPerYear            0.00
TARGET_deathRate            0.00
incidenceRate               0.00
medIncome                   0.00
popEst2015                  0.00
povertyPercent              0.00
studyPerCap                 0.00
binnedInc                   0.00
MedianAge                   0.00
MedianAgeMale               0.00
MedianAgeFemale             0.00
AvgHouseholdSize            0.00
PercentMarried              0.00
PctNoHS18_24                0.00
PctHS18_24                  0.00
PctSomeCol18_24            74.99
PctBachDeg18_24             0.00
PctHS25_Over                0.00
PctBachDeg25_Over           0.00
PctEmployed16_Over          4.99
PctUnemployed16_Over        0.00
PctPrivateCoverage          0.00
PctPrivateCoverageAlone    19.99
PctEmpPrivCoverage          0.00
PctPublicCoverage           0.00
PctPublicCoverageAlone      0.00
PctWhite                    0.00
PctBlack                    0.00
PctAsian                    0.00
PctOtherRace                0.00
PctMarriedHouseholds        0.00
BirthRate                   0.00
dtype: float64
```

We infer that 'PctSomeCol18_24' has high Missing Value Percentage (74.99%) so Dropping this column.

[22]:
```
#Dropping the Column for the stated above
df = df.drop(columns = ['PctSomeCol18_24'],axis=1)
```

Imputing the Null Values in the remaining two Columns with the values of their respective medians

[23]:
```
#Imputing the Null Values by median of the respective columns
df['PctEmployed16_Over'] = df['PctEmployed16_Over'].
 ↪fillna(df['PctEmployed16_Over'].median())
df['PctPrivateCoverageAlone'] = df['PctPrivateCoverageAlone'].
 ↪fillna(df['PctPrivateCoverageAlone'].median())
```

[24]:
```
#Again Checking the Percentage of Columns having Missing Values in case all the␣
 ↪values have not been imputed.
round(df.isnull().sum()/df.shape[0]*100,2)
```

[24]:
```
avgAnnCount                0.0
avgDeathsPerYear           0.0
TARGET_deathRate           0.0
incidenceRate              0.0
medIncome                  0.0
popEst2015                 0.0
povertyPercent             0.0
studyPerCap                0.0
binnedInc                  0.0
MedianAge                  0.0
MedianAgeMale              0.0
MedianAgeFemale            0.0
AvgHouseholdSize           0.0
PercentMarried             0.0
PctNoHS18_24               0.0
PctHS18_24                 0.0
PctBachDeg18_24            0.0
PctHS25_Over               0.0
PctBachDeg25_Over          0.0
PctEmployed16_Over         0.0
PctUnemployed16_Over       0.0
PctPrivateCoverage         0.0
PctPrivateCoverageAlone    0.0
PctEmpPrivCoverage         0.0
PctPublicCoverage          0.0
PctPublicCoverageAlone     0.0
PctWhite                   0.0
PctBlack                   0.0
PctAsian                   0.0
PctOtherRace               0.0
PctMarriedHouseholds       0.0
BirthRate                  0.0
dtype: float64
```

We can infer that Multiple Features Contain Outliers including the Target Variable , 'TARGET_deathRate', so we need to treat these outliers.

2.2 Capping the Outliers

[25]:
```
#Creating a copy of the original Dataframe which will be used in the Notebook␣
↪from now onwards
final_df = df.copy()

#Bringing the values of Outliers between 0.01 and 0.99 Quantile
#This may add a little Bias to the Dataset but it's better than dropping the␣
↪rows containing Outliers as,
#Almost 2/3rd of the rows are dropped if we drop the Rows containing  Outliers.
#Hence Capping is more feasible in this case
for col in final_df.columns:
    print("Capping The",col)

    #The values less than 0.01 Quantile are brought to 0.01 Quantile and
    #The values greater than 0.99 Quantile are brought to 0.99 Quantile.
    if (((final_df[col].dtype)=='float64') | ((final_df[col].dtype)=='int64')):
        percentiles = final_df[col].quantile([0.01,0.99]).values
        final_df[col][final_df[col] <= percentiles[0]] = percentiles[0]
        final_df[col][final_df[col] >= percentiles[1]] = percentiles[1]

    #In case of Categorical Variables
    else:
        final_df[col]=final_df[col]
```

```
Capping The avgAnnCount
Capping The avgDeathsPerYear
Capping The TARGET_deathRate
Capping The incidenceRate
Capping The medIncome
Capping The popEst2015
Capping The povertyPercent
Capping The studyPerCap
Capping The binnedInc
Capping The MedianAge
Capping The MedianAgeMale
Capping The MedianAgeFemale
Capping The AvgHouseholdSize
Capping The PercentMarried
Capping The PctNoHS18_24
Capping The PctHS18_24
Capping The PctBachDeg18_24
Capping The PctHS25_Over
Capping The PctBachDeg25_Over
Capping The PctEmployed16_Over
Capping The PctUnemployed16_Over
Capping The PctPrivateCoverage
Capping The PctPrivateCoverageAlone
```

```
Capping The PctEmpPrivCoverage
Capping The PctPublicCoverage
Capping The PctPublicCoverageAlone
Capping The PctWhite
Capping The PctBlack
Capping The PctAsian
Capping The PctOtherRace
Capping The PctMarriedHouseholds
Capping The BirthRate
```

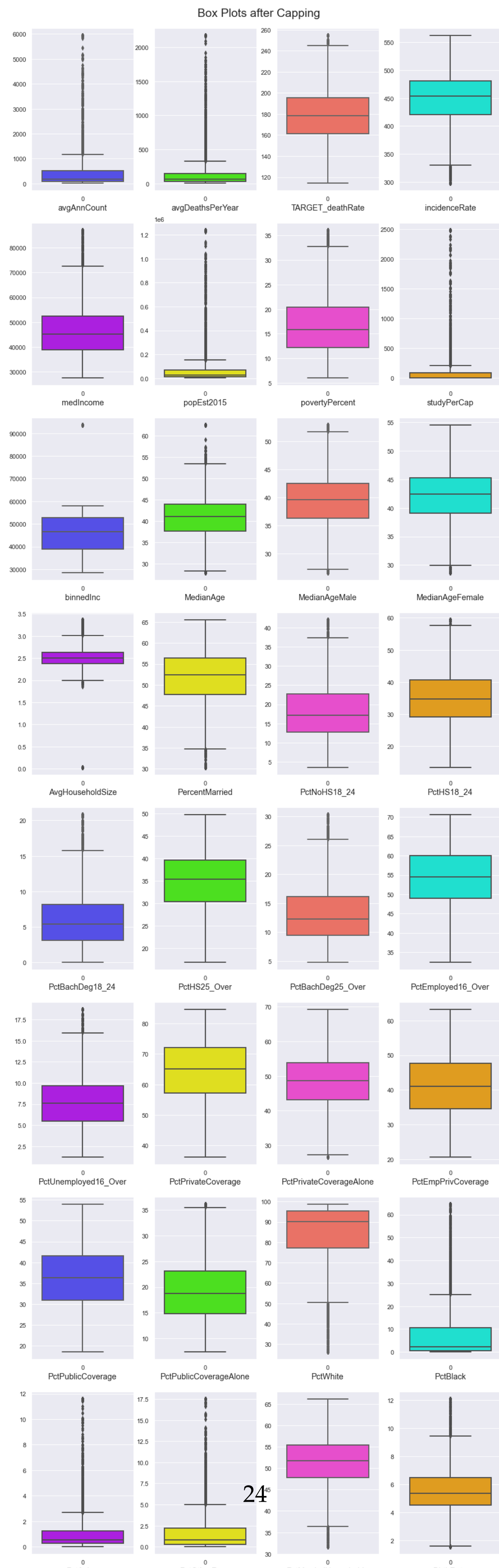
[26]:
```python
#Again checking the Statistics of the Final Dataset after Outlier Treatment
final_df.describe()
```

[26]:
```
         avgAnnCount  avgDeathsPerYear  TARGET_deathRate  incidenceRate  \
count  3047.000000       3047.000000       3047.000000    3047.000000
mean    561.637369        169.728408        178.620553     447.962187
std     961.316735        328.365958         26.830895      50.103072
min      11.000000          4.000000        114.246000     297.614000
25%      76.000000         28.000000        161.200000     420.300000
50%     171.000000         61.000000        178.100000     453.549422
75%     518.000000        149.000000        195.200000     480.850000
max    5932.920000       2169.660000        254.300000     561.670000

          medIncome    popEst2015  povertyPercent  studyPerCap     binnedInc  \
count   3047.000000  3.047000e+03     3047.000000  3047.000000   3047.000000
mean   46964.201464  9.053098e+04       16.844534   138.716908  48878.118280
std    11525.261203  1.904633e+05        6.240246   371.706993  16889.719362
min    27438.880000  1.905040e+03        6.000000     0.000000  28429.050000
25%    38882.500000  1.168400e+04       12.150000     0.000000  38888.250000
50%    45207.000000  2.664300e+04       15.900000     0.000000  46611.300000
75%    52492.000000  6.867100e+04       20.400000    83.650776  52796.000000
max    86982.180000  1.236855e+06       36.016000  2477.718821  93564.750000

         MedianAge  ...  PctPrivateCoverageAlone  PctEmpPrivCoverage  \
count  3047.000000  ...              3047.000000         3047.000000
mean     41.057716  ...                48.495823           41.189785
std       5.529938  ...                 8.850783            9.327220
min      27.900000  ...                26.600000           20.600000
25%      37.700000  ...                43.100000           34.500000
50%      41.000000  ...                48.700000           41.100000
75%      44.000000  ...                53.800000           47.700000
max      62.402000  ...                69.154000           63.154000

       PctPublicCoverage  PctPublicCoverageAlone     PctWhite     PctBlack  \
count        3047.000000             3047.000000  3047.000000  3047.000000
mean           36.246004               19.226148    83.709524     9.020747
std             7.700855                5.991119    16.108120    14.161351
```

```
min          18.546000                  7.400000    26.180228     0.000000
25%          30.900000                 14.850000    77.296180     0.620675
50%          36.300000                 18.800000    90.059774     2.247576
75%          41.550000                 23.100000    95.451693    10.509732
max          53.908000                 36.054000    98.605393    64.460731

          PctAsian  PctOtherRace  PctMarriedHouseholds    BirthRate
count  3047.000000   3047.000000           3047.000000  3047.000000
mean      1.159109      1.900508             51.254326     5.623223
std       1.812262      2.923984              6.375559     1.864892
min       0.000000      0.000000             31.653827     1.547823
25%       0.254199      0.295172             47.763063     4.521419
50%       0.549812      0.826185             51.669941     5.381478
75%       1.221037      2.177960             55.395132     6.493677
max      11.572793     17.499236             66.102467    12.083967

[8 rows x 32 columns]
```

We can see that Much of the Outliers are brought in range, For example : Max MedainAge is now 62.402000. Outlier Treatment has refined the Dataset by a bit

[27]:
```
#Plotting Boxplots Plots after Outlier Treatment to see the Impact
boxPlot(final_df,"Box Plots after Capping")
```


Box Plots after Capping
avgAnnCount
avgDeathsPerYear
TARGET_deathRate
incidenceRate
medIncome
popEst2015
povertyPercent
studyPerCap
binnedInc
MedianAge
MedianAgeMale
MedianAgeFemale
AvgHouseholdSize
PercentMarried
PctNoHS18_24
PctHS18_24
PctBachDeg18_24
PctHS25_Over
PctBachDeg25_Over
PctEmployed16_Over
PctUnemployed16_Over
PctPrivateCoverage
PctPrivateCoverageAlone
PctEmpPrivCoverage
PctPublicCoverage
PctPublicCoverageAlone
PctWhite
PctBlack
PctAsian
PctOtherRace
PctMarriedHouseholds
BirthRate

We can Observe that the Outliers for many variables have been Treated to a very high extent.

2.3 Transforming the Skewed Variables

[28]:
```
#Creating two lists for columns which are Right Skewednand Left Skewed
right_skewed=[]
left_skewed=[]

#Column Names of Columns having Skewness greater than 0.5 are placed in the␣
 ↪right_skewed list and vice versa for left_skewed
for i in  final_df.columns:
    if st.skew(final_df[i])>0.5:
        right_skewed.append(i)
    elif st.skew(final_df[i])<-0.5:
        left_skewed.append(i)

#Printing the Lists
print('Right Skewed :\n ', right_skewed,'\n\nLeft  Skewed :\n ',left_skewed)
```

```
Right Skewed :
  ['avgAnnCount', 'avgDeathsPerYear', 'medIncome', 'popEst2015',
'povertyPercent', 'studyPerCap', 'binnedInc', 'PctNoHS18_24', 'PctBachDeg18_24',
'PctBachDeg25_Over', 'PctUnemployed16_Over', 'PctBlack', 'PctAsian',
'PctOtherRace', 'BirthRate']

Left  Skewed :
  ['AvgHouseholdSize', 'PercentMarried', 'PctWhite']
```

[29]:
```
#Checking Stats of the Skewed Columns
df[right_skewed + left_skewed].describe()
```

[29]:

	avgAnnCount	avgDeathsPerYear	medIncome	popEst2015	\
count	3047.000000	3047.000000	3047.000000	3.047000e+03	
mean	606.338544	185.965868	47063.281917	1.026374e+05	
std	1416.356223	504.134286	12040.090836	3.290592e+05	
min	6.000000	3.000000	22640.000000	8.270000e+02	
25%	76.000000	28.000000	38882.500000	1.168400e+04	
50%	171.000000	61.000000	45207.000000	2.664300e+04	
75%	518.000000	149.000000	52492.000000	6.867100e+04	
max	38150.000000	14010.000000	125635.000000	1.017029e+07	

	povertyPercent	studyPerCap	binnedInc	PctNoHS18_24	\
count	3047.000000	3047.000000	3047.000000	3047.000000	
mean	16.878175	155.399415	48878.118280	18.224450	
std	6.409087	529.628366	16889.719362	8.093064	

```
min          3.200000     0.000000  28429.050000     0.000000
25%         12.150000     0.000000  38888.250000    12.800000
50%         15.900000     0.000000  46611.300000    17.100000
75%         20.400000    83.650776  52796.000000    22.700000
max         47.400000  9762.308998  93564.750000    64.100000

       PctBachDeg18_24  PctBachDeg25_Over  PctUnemployed16_Over     PctBlack  \
count      3047.000000        3047.000000           3047.000000  3047.000000
mean          6.158287          13.282015              7.852412     9.107978
std           4.529059           5.394756              3.452371    14.534538
min           0.000000           2.500000              0.400000     0.000000
25%           3.100000           9.400000              5.500000     0.620675
50%           5.400000          12.300000              7.600000     2.247576
75%           8.200000          16.100000              9.700000    10.509732
max          51.800000          42.200000             29.400000    85.947799

          PctAsian  PctOtherRace    BirthRate  AvgHouseholdSize  \
count  3047.000000   3047.000000  3047.000000       3047.000000
mean      1.253965      1.983523     5.640306          2.479662
std       2.610276      3.517710     1.985816          0.429174
min       0.000000      0.000000     0.000000          0.022100
25%       0.254199      0.295172     4.521419          2.370000
50%       0.549812      0.826185     5.381478          2.500000
75%       1.221037      2.177960     6.493677          2.630000
max      42.619425     41.930251    21.326165          3.970000

       PercentMarried     PctWhite
count     3047.000000  3047.000000
mean        51.773679    83.645286
std          6.896928    16.380025
min         23.100000    10.199155
25%         47.750000    77.296180
50%         52.400000    90.059774
75%         56.400000    95.451693
max         72.500000   100.000000
```

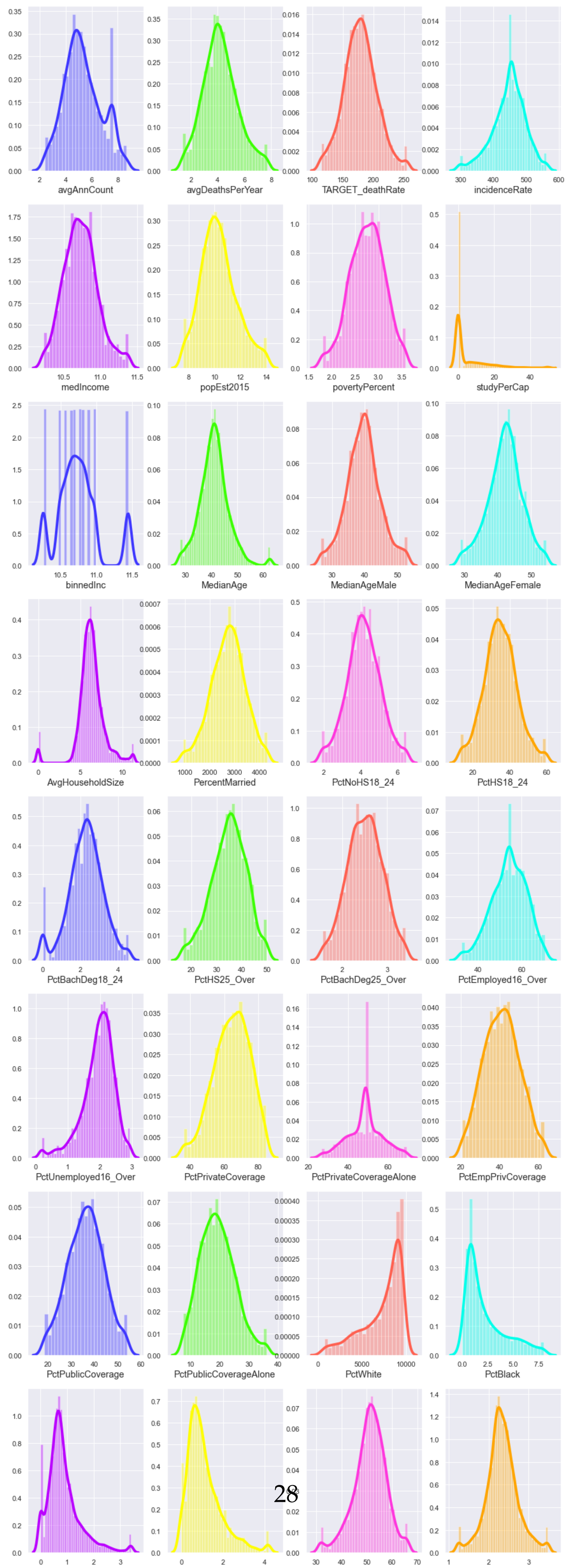
[30]:

```python
#If the columns in the right_skewed list have minimum value 0, then they undergo
 ↪Square-Root Transformation
#Else the columns undergo Logarithmic Transformation
for i in right_skewed:
    if (min(df[i]) == 0):
        final_df[i]= np.sqrt((final_df[i]))
    else:
        final_df[i]= np.log((final_df[i]))

#All the columns in left_skewed list go Squared Transformation
for i in left_skewed:
```

```
        final_df[i]= ((final_df[i])**2)
```

[31]:
```
#Checking the Skewness of All the columns after Skewness Treatment
distributionPlot(final_df,"Distribution Plots after Skewness Treatment")
```

Distribution Plots after Skewness Treatment



The Skewness Treatment has brought almost all the Columns to Normal Distribution.

[32]:
```
#Checking the stats to observe any change
final_df.describe()
```

[32]:
```
        avgAnnCount  avgDeathsPerYear  TARGET_deathRate  incidenceRate  \
count   3047.000000       3047.000000       3047.000000    3047.000000
mean       5.319542          4.210056        178.620553     447.962187
std        1.414317          1.295532         26.830895      50.103072
min        2.397895          1.386294        114.246000     297.614000
25%        4.330733          3.332205        161.200000     420.300000
50%        5.141664          4.110874        178.100000     453.549422
75%        6.249975          5.003946        195.200000     480.850000
max        8.688272          7.682326        254.300000     561.670000

          medIncome  popEst2015  povertyPercent  studyPerCap   binnedInc  \
count   3047.000000 3047.000000     3047.000000  3047.000000 3047.000000
mean      10.729368   10.330772        2.755976     5.878143   10.747857
std        0.232604    1.388662        0.373177    10.207769    0.302164
min       10.219716    7.552258        1.791759     0.000000   10.255167
25%       10.568300    9.365974        2.497321     0.000000   10.568447
50%       10.719007   10.190282        2.766319     0.000000   10.749598
75%       10.868416   11.137082        3.015535     9.146077   10.874191
max       11.373459   14.028083        3.583963    49.776690   11.446409

          MedianAge  ...  PctPrivateCoverageAlone  PctEmpPrivCoverage  \
count   3047.000000  ...              3047.000000         3047.000000
mean      41.057716  ...                48.495823           41.189785
std        5.529938  ...                 8.850783            9.327220
min       27.900000  ...                26.600000           20.600000
25%       37.700000  ...                43.100000           34.500000
50%       41.000000  ...                48.700000           41.100000
75%       44.000000  ...                53.800000           47.700000
max       62.402000  ...                69.154000           63.154000

        PctPublicCoverage  PctPublicCoverageAlone     PctWhite     PctBlack  \
count         3047.000000             3047.000000  3047.000000  3047.000000
mean            36.246004               19.226148  7266.670839     2.264601
std              7.700855                5.991119  2308.844505     1.973222
min             18.546000                7.400000   685.404341     0.000000
25%             30.900000               14.850000  5974.700036     0.787829
50%             36.300000               18.800000  8110.762927     1.499192
75%             41.550000               23.100000  9111.025854     3.241870
max             53.908000               36.054000  9723.023570     8.028744
```

```
         PctAsian  PctOtherRace  PctMarriedHouseholds    BirthRate
count 3047.000000   3047.000000           3047.000000  3047.000000
mean     0.876667      1.108045             51.254326     2.338994
std      0.625054      0.820344              6.375559     0.390360
min      0.000000      0.000000             31.653827     1.244115
25%      0.504182      0.543297             47.763063     2.126363
50%      0.741493      0.908947             51.669941     2.319801
75%      1.105006      1.475791             55.395132     2.548269
max      3.401881      4.183209             66.102467     3.476200

[8 rows x 32 columns]
```

2.4 Scaling the Features

[33]:
```python
#Scaling the Features between 0 - 1, for easier and efficient performance by the Model
scaler = MinMaxScaler()
num_vars = final_df.columns
final_df[num_vars] = scaler.fit_transform(final_df[num_vars])
final_df.describe()
```

[33]:
```
       avgAnnCount  avgDeathsPerYear  TARGET_deathRate  incidenceRate  \
count  3047.000000       3047.000000       3047.000000    3047.000000
mean      0.464463          0.448499          0.459641       0.569380
std       0.224838          0.205770          0.191575       0.189744
min       0.000000          0.000000          0.000000       0.000000
25%       0.307269          0.309069          0.335256       0.464621
50%       0.436185          0.432746          0.455924       0.590539
75%       0.612377          0.574592          0.578020       0.693929
max       1.000000          1.000000          1.000000       1.000000

         medIncome   popEst2015  povertyPercent  studyPerCap    binnedInc  \
count  3047.000000  3047.000000     3047.000000  3047.000000  3047.000000
mean      0.441738     0.429059        0.538006     0.118090     0.413594
std       0.201608     0.214438        0.208223     0.205071     0.253654
min       0.000000     0.000000        0.000000     0.000000     0.000000
25%       0.302133     0.280075        0.393684     0.000000     0.262987
50%       0.432758     0.407365        0.543777     0.000000     0.415055
75%       0.562257     0.553570        0.682833     0.183742     0.519646
max       1.000000     1.000000        1.000000     1.000000     1.000000

         MedianAge  ...  PctPrivateCoverageAlone  PctEmpPrivCoverage  \
count  3047.000000  ...              3047.000000         3047.000000
mean      0.381361  ...                 0.514542            0.483851
std       0.160279  ...                 0.207989            0.219186
min       0.000000  ...                 0.000000            0.000000
25%       0.284042  ...                 0.387743            0.326644
```

```
50%        0.379688  ...                  0.519340             0.481741
75%        0.466640  ...                  0.639188             0.636838
max        1.000000  ...                  1.000000             1.000000

       PctPublicCoverage  PctPublicCoverageAlone     PctWhite     PctBlack  \
count        3047.000000             3047.000000  3047.000000  3047.000000
mean            0.500537                0.412722     0.728208     0.282062
std             0.217772                0.209085     0.255470     0.245770
min             0.000000                0.000000     0.000000     0.000000
25%             0.349358                0.259999     0.585253     0.098126
50%             0.502064                0.397850     0.821606     0.186728
75%             0.650529                0.547917     0.932283     0.403783
max             1.000000                1.000000     1.000000     1.000000

          PctAsian  PctOtherRace  PctMarriedHouseholds    BirthRate
count  3047.000000   3047.000000           3047.000000  3047.000000
mean      0.257701      0.264879              0.568977     0.490518
std       0.183738      0.196104              0.185074     0.174886
min       0.000000      0.000000              0.000000     0.000000
25%       0.148207      0.129876              0.467631     0.395257
50%       0.217966      0.217285              0.581042     0.481920
75%       0.324822      0.352789              0.689180     0.584276
max       1.000000      1.000000              1.000000     1.000000

[8 rows x 32 columns]
```

2.5 Splitting the Dataset into Train and Test Datasets

[34]:
```
y = final_df.pop('TARGET_deathRate')
X = final_df
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size = 0.7,␣
 ↪test_size = 0.3, random_state = 100)
```

2.6 Model-1 Using Ordinary Least Square Linear Model (containing all Features)

[35]:
```
# Adding a constant manually because OLS otherwise fits the line through the␣
 ↪origin
X_train_lm = sm.add_constant(X_train[list(X_train.columns)])

# Create a first fitted model
lr = sm.OLS(y_train, X_train_lm).fit()

#Viewing Summary
print(lr.summary())
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:       TARGET_deathRate   R-squared:                       0.757
Model:                            OLS   Adj. R-squared:                  0.754
Method:                 Least Squares   F-statistic:                     211.3
Date:                Thu, 10 Dec 2020   Prob (F-statistic):               0.00
Time:                        22:19:51   Log-Likelihood:                 2036.6
No. Observations:                2132   AIC:                            -4009.
Df Residuals:                    2100   BIC:                            -3828.
Df Model:                          31
Covariance Type:            nonrobust
================================================================================
===========
                         coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
const                  0.7539      0.061     12.413      0.000       0.635
0.873
avgAnnCount           -0.1323      0.016     -8.102      0.000      -0.164
-0.100
avgDeathsPerYear       4.3225      0.095     45.608      0.000       4.137
4.508
incidenceRate          0.1836      0.014     13.580      0.000       0.157
0.210
medIncome              0.0190      0.052      0.364      0.716      -0.084
0.122
popEst2015            -4.3140      0.099    -43.746      0.000      -4.507
-4.121
povertyPercent        -0.0077      0.035     -0.218      0.828      -0.077
0.061
studyPerCap           -0.0129      0.011     -1.126      0.260      -0.035
0.010
binnedInc              0.0707      0.031      2.263      0.024       0.009
0.132
MedianAge             -0.0003      0.030     -0.010      0.992      -0.059
0.059
MedianAgeMale          0.0033      0.036      0.092      0.927      -0.066
0.073
MedianAgeFemale       -0.4866      0.037    -13.038      0.000      -0.560
-0.413
AvgHouseholdSize       0.0456      0.023      2.009      0.045       0.001
0.090
PercentMarried        -0.0401      0.033     -1.228      0.220      -0.104
0.024
PctNoHS18_24           0.0101      0.014      0.734      0.463      -0.017
0.037
PctHS18_24             0.0979      0.013      7.358      0.000       0.072
0.124
PctBachDeg18_24        0.0096      0.014      0.676      0.499      -0.018
```

```
0.037
PctHS25_Over              -0.0483      0.018     -2.616      0.009      -0.085
-0.012
PctBachDeg25_Over         -0.1236      0.022     -5.636      0.000      -0.167
-0.081
PctEmployed16_Over        -0.1013      0.022     -4.550      0.000      -0.145
-0.058
PctUnemployed16_Over       0.1231      0.019      6.369      0.000       0.085
0.161
PctPrivateCoverage        -0.0365      0.041     -0.884      0.377      -0.118
0.044
PctPrivateCoverageAlone    0.0294      0.022      1.358      0.175      -0.013
0.072
PctEmpPrivCoverage        -0.0353      0.026     -1.338      0.181      -0.087
0.016
PctPublicCoverage         -0.4676      0.047     -9.897      0.000      -0.560
-0.375
PctPublicCoverageAlone     0.3234      0.047      6.843      0.000       0.231
0.416
PctWhite                  -0.0445      0.021     -2.168      0.030      -0.085
-0.004
PctBlack                  -0.0090      0.019     -0.472      0.637      -0.046
0.028
PctAsian                   0.0028      0.019      0.150      0.881      -0.034
0.039
PctOtherRace              -0.1048      0.014     -7.663      0.000      -0.132
-0.078
PctMarriedHouseholds       0.0073      0.033      0.218      0.827      -0.058
0.073
BirthRate                 -0.0290      0.013     -2.291      0.022      -0.054
-0.004
==============================================================================
Omnibus:                      354.345   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1774.670
Skew:                           0.695   Prob(JB):                         0.00
Kurtosis:                       7.248   Cond. No.                         197.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

2.6.1 Residual Analysis of Training Data of Model 1

[36]:
```
#Creating a function for Error Terms Distribution Plot
def errorTermsPlot(y,y_hat,color,typ,xlabel):
    # Plot the histogram of the error terms
    fig = plt.figure()
    sns.distplot(((y) - y_hat), bins = 20,color=color,kde_kws=dict(linewidth=4))
    fig.suptitle('Error Terms for ' + typ + ' Data' , fontsize = 15)
    # Plot heading
    plt.xlabel(xlabel, fontsize = 12)
```

[37]:
```
#Predicting y_train based on X_train_lm
y_train_pred = lr.predict(X_train_lm)

#Plotting the Graph
errorTermsPlot(y_train,y_train_pred,'r','Training','Errors')
```

Error Terms are Normally Distributed for the Training Data Prediction

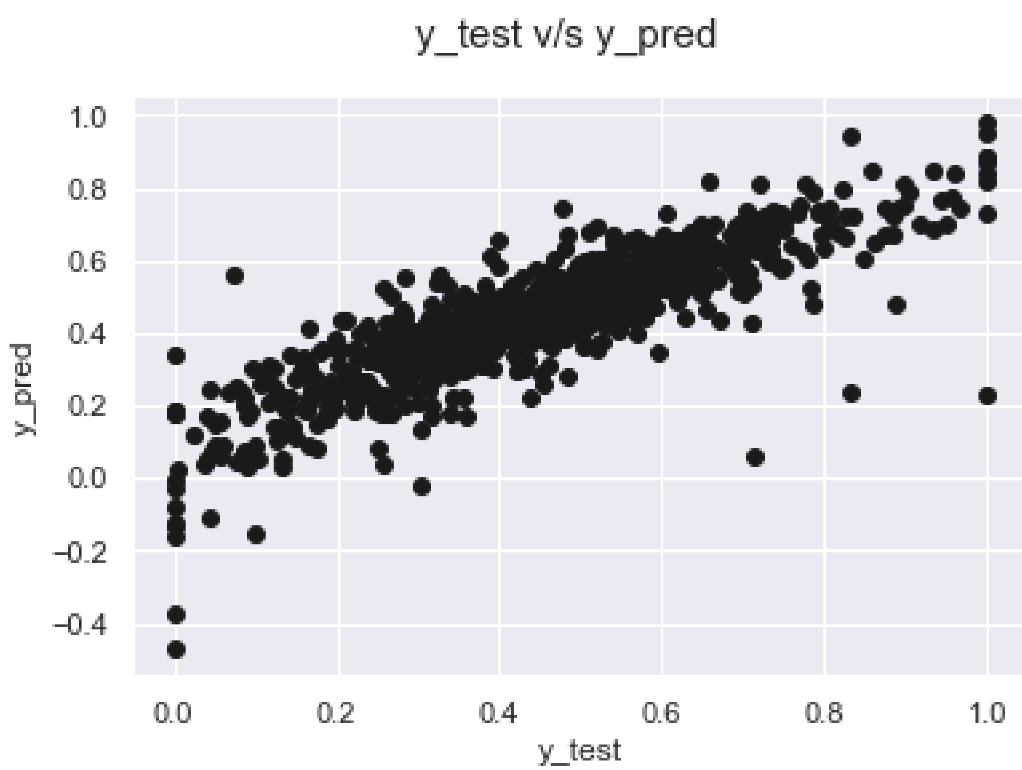
2.6.2 Making Predictions using Test Data of Model-1

[38]:
```
# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test[X_train.columns.values]

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)

# Making predictions
y_pred = lr.predict(X_test_new)
```

2.6.3 Evaluation of Model-1

[39]:
```
#Creating a function for y_test vs y_pred Plot
def yTest_vs_yPredPlot(y,y_hat,color):
    # Plotting y_test and y_pred to understand the spread.
    fig = plt.figure()
    plt.scatter(y,y_hat,color=color)
    fig.suptitle('y_test v/s y_pred', fontsize=15)              # Plot heading
    plt.xlabel('y_test', fontsize=12)                           # X-label
    plt.ylabel('y_pred', fontsize=12)
```

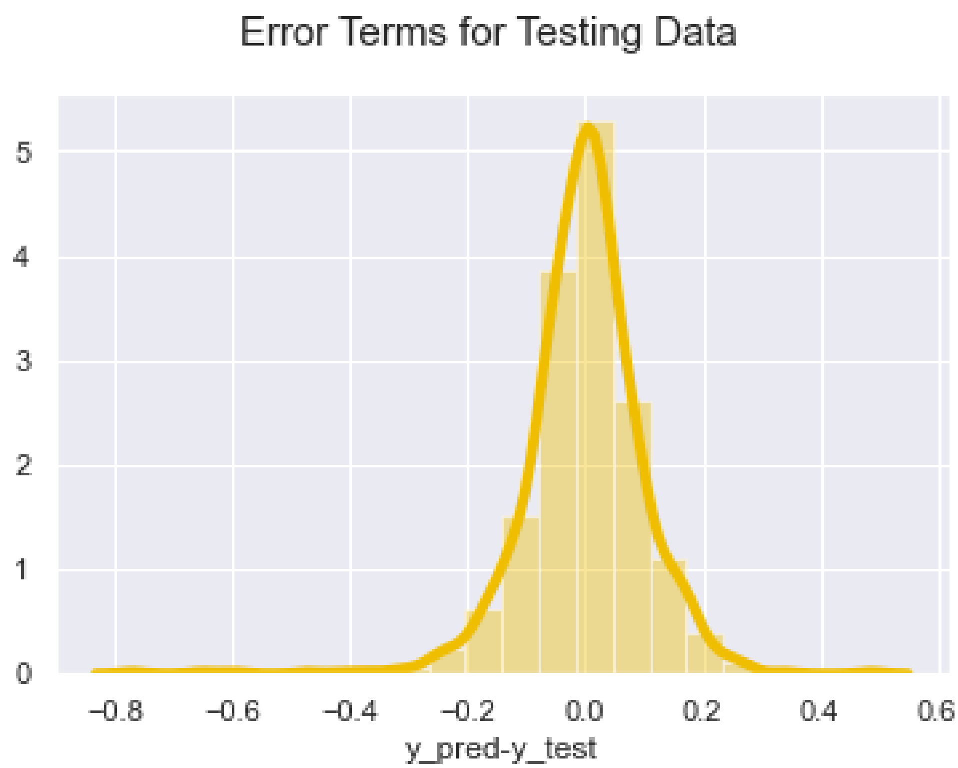
[40]:
```
#Plotting y_test vs y_pred for Model 1
yTest_vs_yPredPlot(y_test,y_pred,'k')
```



Since the Data is not scattered and somewhat signifies Linear Relation, The Model is not good.

2.6.4 Residual Analysis of Testing Data of Model 1

[41]:
```
#Plotting the Graph for Test Errors
errorTermsPlot(y_pred,y_test,'#EFBE01','Testing','y_pred-y_test')
```



2.7 Model-2 , Checking for High VIF values and Insignificant Features and performing Backward Elimination in OLS Model

[42]:
```
#Defining a function which will calculate the VIF values and store them in a␣
↪DataFrame
#High VIF Means High Multicollinearity
def calculateVIF(X_train_lm):
    vif = pd.DataFrame()
    vif['Features'] = X_train_lm.columns
    vif['VIF'] = [variance_inflation_factor(X_train_lm.values, i) for i in␣
↪range(X_train_lm.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
```

```
    vif = vif.sort_values(by = "VIF", ascending = False).reset_index()
    vif = vif.drop(columns = ['index'],axis = 1)
    return vif
```

[43]:
```
#Calculating the VIF Values for Model-1
vif = calculateVIF(X_train_lm)
vif
```

[43]:

	Features	VIF
0	const	893.87
1	popEst2015	107.35
2	avgDeathsPerYear	91.53
3	medIncome	26.42
4	PctPublicCoverage	25.37
5	PctPublicCoverageAlone	23.59
6	PctPrivateCoverage	19.50
7	binnedInc	14.84
8	MedianAgeFemale	13.54
9	povertyPercent	12.86
10	MedianAgeMale	11.78
11	PercentMarried	10.51
12	PctMarriedHouseholds	9.12
13	PctEmpPrivCoverage	7.99
14	PctWhite	6.75
15	MedianAge	5.69
16	PctBlack	5.36
17	PctEmployed16_Over	5.16
18	PctBachDeg25_Over	5.12
19	PctPrivateCoverageAlone	4.90
20	PctHS25_Over	3.63
21	avgAnnCount	3.30
22	PctUnemployed16_Over	2.89
23	PctAsian	2.79
24	AvgHouseholdSize	2.28
25	PctBachDeg18_24	1.94
26	PctNoHS18_24	1.81
27	PctOtherRace	1.80
28	PctHS18_24	1.63
29	incidenceRate	1.54
30	studyPerCap	1.26
31	BirthRate	1.21

[44]:
```
#Using the function defined above and dropping the Columns which have VIF value
↪>= 10
X_train_vif = X_train_lm.copy()

while True:
```

```
    #Calculating the VIF Values
    vif = calculateVIF(X_train_vif)

    #Dropping the Columns with VIF >= 10
    #vif.iloc[0,0] is 'const' , so we consider the columns from vif.iloc[1,0]
    if (vif.iloc[1,1] >= 10):
        print('Eliminating : ',vif.iloc[1,0])
        X_train_vif.drop(columns=[vif.iloc[1,0]], axis=1,inplace=True)

    else:
        break

#Removing 'const'
X_train_vif.drop(columns=[vif.iloc[0,0]],axis =1,inplace=True)
```

```
Eliminating :  popEst2015
Eliminating :  medIncome
Eliminating :  PctPublicCoverage
Eliminating :  PctPrivateCoverage
Eliminating :  MedianAgeFemale
```

[45]:

```
#Defining a function to drop the Variables which are insignificant (have␣
↪pvalue>0.05)
def dropPvalues(X_train):

    while True:
        # Add a constant
        X_train_lm = sm.add_constant(X_train[X_train.columns])

        # Create a first fitted model
        lr = sm.OLS(y_train, X_train_lm).fit()

        #Extracting pvalues from the model
        l = lr.pvalues
        l=l.sort_values(ascending=False)

        #Dropping the Column if pvalue>0.05
        if (l[0] > 0.05):
            print('Eliminating : ',l.index[0])
            X_train.drop(columns=[l.index[0]], axis=1,inplace=True)

        else:
            break
    print(lr.summary())
    return lr , X_train_lm
```

[46]:
```
X_train1 = X_train[X_train_vif.columns.values].copy()

#Dropping the Columns with high P values and then using the obtained Columns to␣
↪generate Model-2
lr1 , X_train_lm1 = dropPvalues(X_train1)
```

```
Eliminating :  PctEmpPrivCoverage
Eliminating :  PctPrivateCoverageAlone
Eliminating :  povertyPercent
Eliminating :  PctNoHS18_24
Eliminating :  AvgHouseholdSize
Eliminating :  PctEmployed16_Over
Eliminating :  studyPerCap
Eliminating :  PctBlack
Eliminating :  PctBachDeg18_24
Eliminating :  BirthRate
Eliminating :  MedianAgeMale
                            OLS Regression Results
==============================================================================
Dep. Variable:       TARGET_deathRate   R-squared:                       0.528
Model:                            OLS   Adj. R-squared:                  0.525
Method:                 Least Squares   F-statistic:                     158.0
Date:                Thu, 10 Dec 2020   Prob (F-statistic):               0.00
Time:                        22:19:54   Log-Likelihood:                 1328.5
No. Observations:                2132   AIC:                            -2625.
Df Residuals:                    2116   BIC:                            -2534.
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
==========
                            coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----------
const                     0.3820      0.037     10.260      0.000       0.309
0.455
avgAnnCount              -0.1979      0.022     -8.973      0.000      -0.241
-0.155
avgDeathsPerYear          0.2688      0.027      9.808      0.000       0.215
0.322
incidenceRate             0.3782      0.017     22.054      0.000       0.345
0.412
binnedInc                -0.0510      0.021     -2.446      0.015      -0.092
-0.010
MedianAge                -0.1413      0.022     -6.474      0.000      -0.184
-0.099
PercentMarried            0.1292      0.036      3.593      0.000       0.059
0.200
```

```
PctHS18_24                0.0786      0.017      4.554      0.000       0.045
0.113
PctHS25_Over              0.0627      0.023      2.676      0.008       0.017
0.109
PctBachDeg25_Over        -0.2231      0.028     -7.992      0.000      -0.278
-0.168
PctUnemployed16_Over      0.0744      0.025      3.029      0.002       0.026
0.123
PctPublicCoverageAlone    0.0563      0.025      2.270      0.023       0.008
0.105
PctWhite                 -0.0610      0.016     -3.825      0.000      -0.092
-0.030
PctAsian                 -0.0794      0.025     -3.235      0.001      -0.127
-0.031
PctOtherRace             -0.1119      0.018     -6.316      0.000      -0.147
-0.077
PctMarriedHouseholds     -0.1653      0.037     -4.519      0.000      -0.237
-0.094
==============================================================================
Omnibus:                       96.468   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              280.424
Skew:                           0.160   Prob(JB):                     1.28e-61
Kurtosis:                       4.748   Cond. No.                         39.7
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

2.7.1 Residual Analysis of Training Data of Model-2

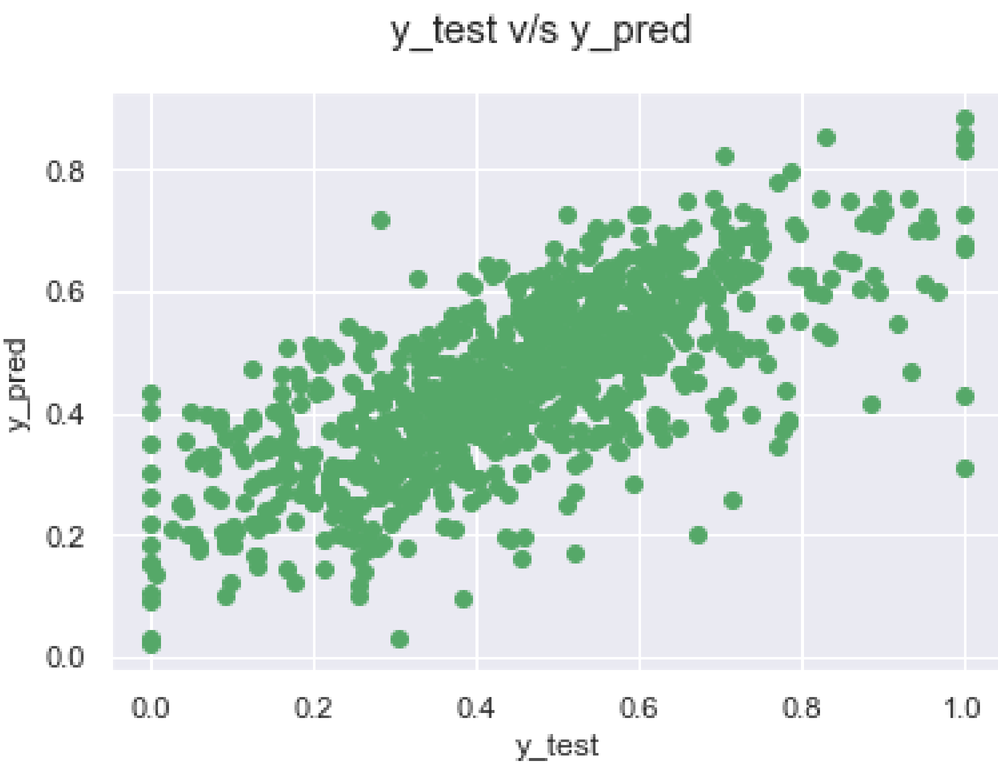
[47]:
```python
y_train_pred1 = lr1.predict(X_train_lm1)

errorTermsPlot(y_train,y_train_pred1,'y','Training','Errors')
```

2.7.2 Making Predictions using Test Data of Model-2

[48]:
```
# Creating X_test_new dataframe by dropping variables from X_test
X_test_new1 = X_test[X_train1.columns.values]

# Adding a constant variable
X_test_new1 = sm.add_constant(X_test_new1)

# Making predictions
y_pred1 = lr1.predict(X_test_new1)
```

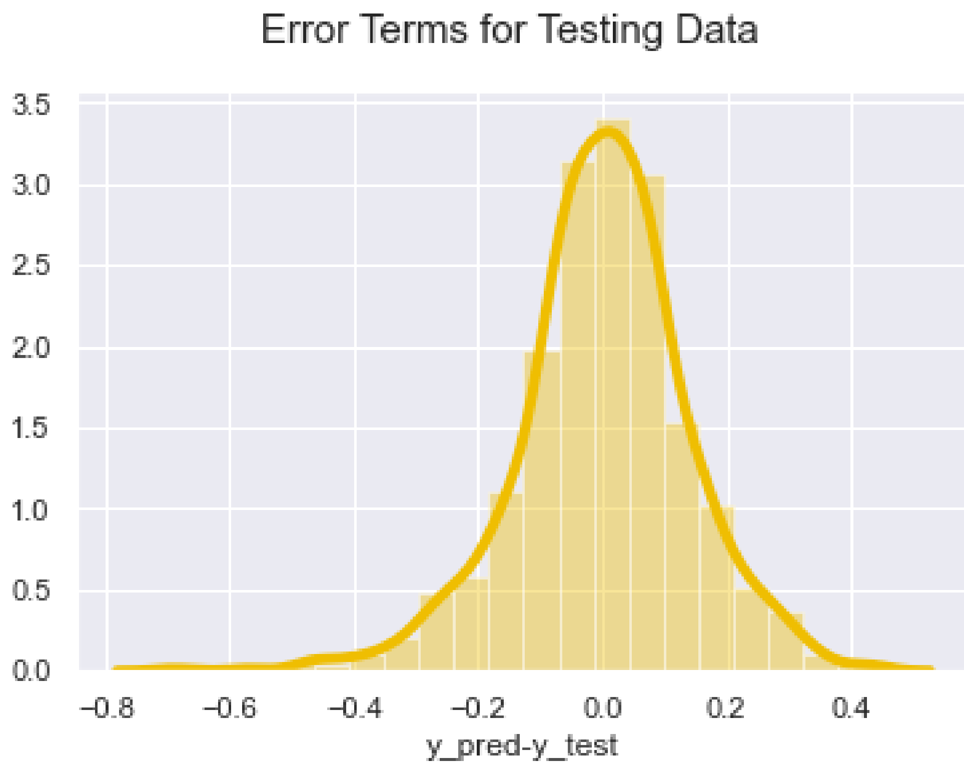
2.7.3 Evaluation of Model-2

[49]:
```
yTest_vs_yPredPlot(y_test,y_pred1,'g')
```



2.7.4 Residual Analysis of Testing Data of Model-2

[50]:
```
errorTermsPlot(y_pred1,y_test,'#EFBE01','Testing','y_pred-y_test')
```



2.8 Model-3 Using Recursive Feature Elimination for Feature Selection and Model Building

[51]:
```
#Fitting the Training Data to RFE model
estimator = SVR(kernel="linear")
rfe = RFE(estimator, step=1)
rfe = rfe.fit(X_train, y_train)
```

[52]:
```
#Obtaining the Column Names, Whether they should be included in the Model or not␣
↪and Their Ranking
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

[52]:
```
[('avgAnnCount', True, 1),
 ('avgDeathsPerYear', True, 1),
 ('incidenceRate', True, 1),
 ('medIncome', True, 1),
 ('popEst2015', True, 1),
 ('povertyPercent', False, 10),
 ('studyPerCap', False, 5),
 ('binnedInc', False, 15),
 ('MedianAge', False, 8),
 ('MedianAgeMale', False, 11),
```

```
 ('MedianAgeFemale', True, 1),
 ('AvgHouseholdSize', False, 13),
 ('PercentMarried', False, 9),
 ('PctNoHS18_24', False, 16),
 ('PctHS18_24', True, 1),
 ('PctBachDeg18_24', False, 17),
 ('PctHS25_Over', False, 7),
 ('PctBachDeg25_Over', True, 1),
 ('PctEmployed16_Over', True, 1),
 ('PctUnemployed16_Over', True, 1),
 ('PctPrivateCoverage', True, 1),
 ('PctPrivateCoverageAlone', False, 2),
 ('PctEmpPrivCoverage', False, 14),
 ('PctPublicCoverage', True, 1),
 ('PctPublicCoverageAlone', True, 1),
 ('PctWhite', False, 3),
 ('PctBlack', False, 12),
 ('PctAsian', False, 6),
 ('PctOtherRace', True, 1),
 ('PctMarriedHouseholds', True, 1),
 ('BirthRate', False, 4)]
```

[53]:
```
#List of Supported Columns
col = X_train.columns[rfe.support_]
col
```

[53]:
```
Index(['avgAnnCount', 'avgDeathsPerYear', 'incidenceRate', 'medIncome',
       'popEst2015', 'MedianAgeFemale', 'PctHS18_24', 'PctBachDeg25_Over',
       'PctEmployed16_Over', 'PctUnemployed16_Over', 'PctPrivateCoverage',
       'PctPublicCoverage', 'PctPublicCoverageAlone', 'PctOtherRace',
       'PctMarriedHouseholds'],
      dtype='object')
```

[54]:
```
# Adding a constant manually because OLS otherwise fits the line through the␣
 ↪origin
X_train_rfe = sm.add_constant(X_train[col])
# Create a first fitted model
lr2 = sm.OLS(y_train, X_train_rfe).fit()
print(lr2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        TARGET_deathRate   R-squared:                       0.752
Model:                             OLS   Adj. R-squared:                  0.750
Method:                  Least Squares   F-statistic:                     427.5
Date:                 Thu, 10 Dec 2020   Prob (F-statistic):               0.00
Time:                         22:19:59   Log-Likelihood:                 2013.2
No. Observations:                 2132   AIC:                            -3994.
```

```
Df Residuals:                    2116   BIC:                            -3904.
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
==========
                           coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----------
const                    0.7358      0.032     23.232      0.000       0.674
0.798
avgAnnCount             -0.1346      0.016     -8.307      0.000      -0.166
-0.103
avgDeathsPerYear         4.1971      0.092     45.768      0.000       4.017
4.377
incidenceRate            0.1867      0.013     13.846      0.000       0.160
0.213
medIncome                0.1422      0.024      5.845      0.000       0.094
0.190
popEst2015              -4.1905      0.095    -44.154      0.000      -4.377
-4.004
MedianAgeFemale         -0.4880      0.022    -22.232      0.000      -0.531
-0.445
PctHS18_24               0.0866      0.012      7.084      0.000       0.063
0.111
PctBachDeg25_Over       -0.0832      0.017     -4.894      0.000      -0.116
-0.050
PctEmployed16_Over      -0.1408      0.019     -7.487      0.000      -0.178
-0.104
PctUnemployed16_Over     0.1351      0.019      7.151      0.000       0.098
0.172
PctPrivateCoverage      -0.0923      0.027     -3.391      0.001      -0.146
-0.039
PctPublicCoverage       -0.4929      0.040    -12.393      0.000      -0.571
-0.415
PctPublicCoverageAlone   0.3217      0.042      7.732      0.000       0.240
0.403
PctOtherRace            -0.0862      0.013     -6.799      0.000      -0.111
-0.061
PctMarriedHouseholds    -0.0463      0.015     -2.997      0.003      -0.077
-0.016
==============================================================================
Omnibus:                      387.777   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1990.133
Skew:                           0.764   Prob(JB):                         0.00
Kurtosis:                       7.480   Cond. No.                         142.
==============================================================================
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

2.8.1 Residual Analysis of Training Data of RFE Model, Model-3

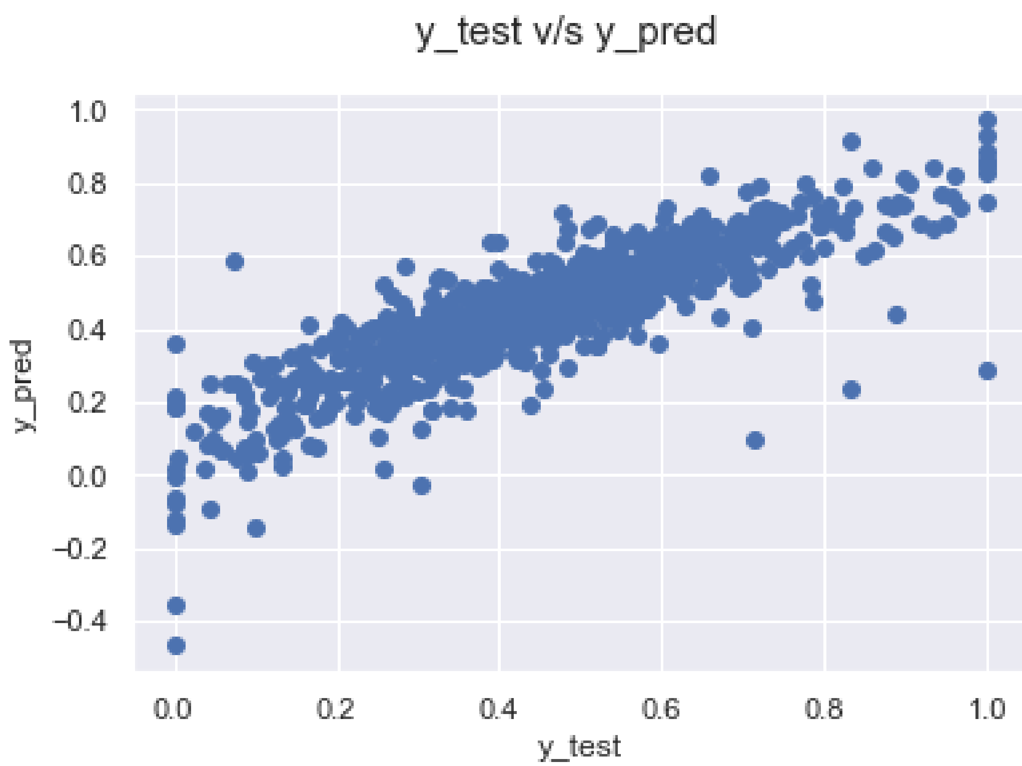
[55]:
```
y_train_pred2 = lr2.predict(X_train_rfe)

errorTermsPlot(y_train,y_train_pred2,'Orange','Training','Errors')
```

2.8.2 Making Predictions using Test Data of Model-3, RFE Model

[56]:
```
# Creating X_test_new dataframe by dropping variables from X_test
X_test_new2 = X_test[col]

# Adding a constant variable
X_test_new2 = sm.add_constant(X_test_new2)

# Making predictions
y_pred2 = lr2.predict(X_test_new2)
```

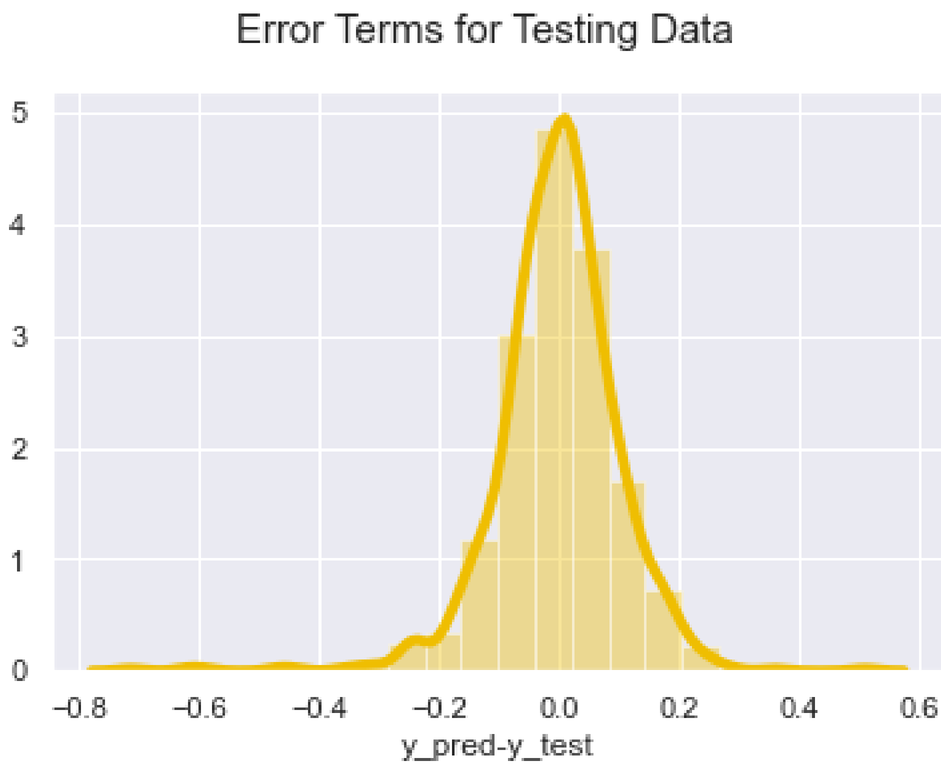
2.8.3 Evaluation of Model-3, RFE Model

[57]:
```
# Plotting y_test and y_pred to understand the spread.
yTest_vs_yPredPlot(y_test,y_pred2,'b')
```



2.8.4 Residual Analysis of Testing Data of Model-3, RFE Model

[58]:
```
errorTermsPlot(y_pred2,y_test,'#EFBE01','Testing','y_pred-y_test')
```



2.9 Model-4 Removing High VIF and High P valued Columns from Model-3, RFE Model

Checking for VIF values in Model-3, RFE Model

[59]:
```
vif = calculateVIF(X_train[col])
vif
```

[59]:

	Features	VIF
0	popEst2015	473.16
1	avgDeathsPerYear	466.46
2	PctPublicCoverage	113.42
3	PctPublicCoverageAlone	76.87
4	PctPrivateCoverage	47.85
5	MedianAgeFemale	34.87
6	medIncome	31.50
7	PctUnemployed16_Over	29.06
8	PctEmployed16_Over	26.34
9	PctBachDeg25_Over	19.80
10	PctMarriedHouseholds	18.44
11	avgAnnCount	16.41
12	incidenceRate	15.40
13	PctHS18_24	9.07

```
14             PctOtherRace    3.87
```

Dropping Columns with VIF > 10

[60]:
```
#Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor

X_train_vif3 = X_train_rfe.copy()

while True:

    # Create a dataframe that will contain the names of all the feature␣
 ↪variables and their respective VIFs
    vif = calculateVIF(X_train_vif3)

    if (vif.iloc[1,1] >= 10):
        X_train_vif3.drop(columns=[vif.iloc[1,0]], axis=1,inplace=True)

    else:
        break

X_train_vif3.drop(columns=[vif.iloc[0,0]],axis =1,inplace=True)
```

[61]:
```
# Building a new model with new Features
X_train3 = X_train[X_train_vif3.columns.values].copy()

lr3 , X_train_lm3 = dropPvalues(X_train3)
```

```
Eliminating :  PctEmployed16_Over
Eliminating :  PctPublicCoverage
Eliminating :  PctUnemployed16_Over
                            OLS Regression Results
==============================================================================
Dep. Variable:       TARGET_deathRate   R-squared:                       0.523
Model:                            OLS   Adj. R-squared:                  0.520
Method:                 Least Squares   F-statistic:                     232.2
Date:                Thu, 10 Dec 2020   Prob (F-statistic):               0.00
Time:                        22:20:00   Log-Likelihood:                 1315.5
No. Observations:                2132   AIC:                            -2609.
Df Residuals:                    2121   BIC:                            -2547.
Df Model:                          10
Covariance Type:            nonrobust
================================================================================
========
                           coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--------
```

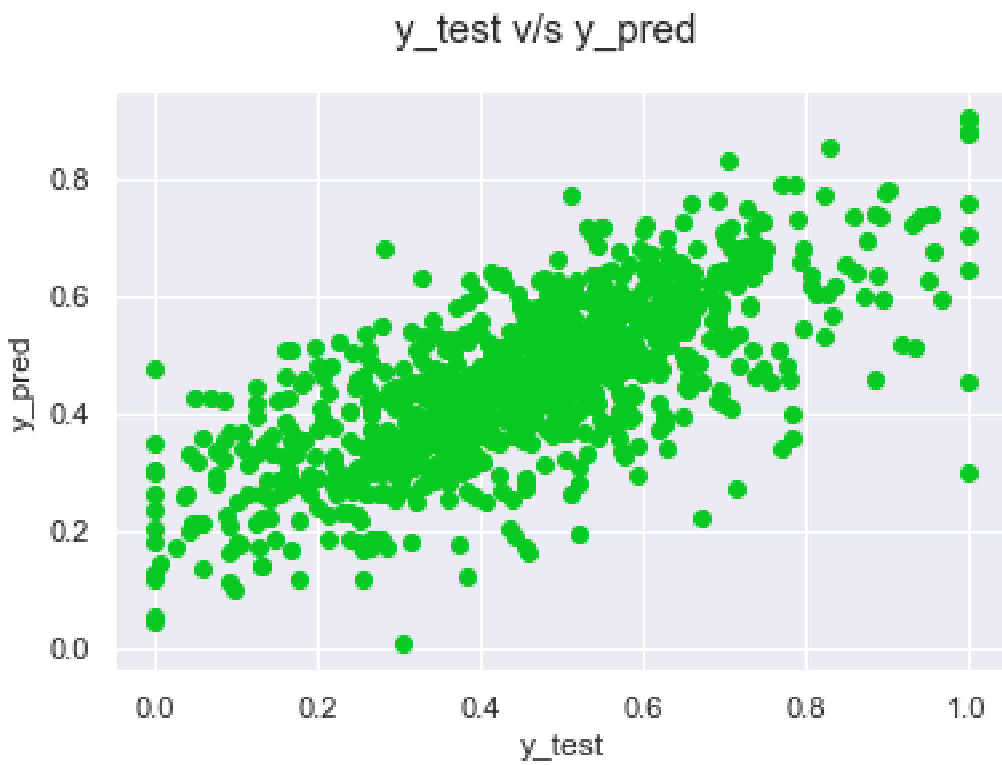
```
const                    0.5241      0.022     24.163      0.000       0.482
0.567
avgAnnCount             -0.1858      0.022     -8.408      0.000      -0.229
-0.143
avgDeathsPerYear         0.2485      0.024     10.312      0.000       0.201
0.296
incidenceRate            0.3963      0.017     22.987      0.000       0.363
0.430
medIncome               -0.0890      0.030     -3.001      0.003      -0.147
-0.031
MedianAgeFemale         -0.0897      0.016     -5.517      0.000      -0.122
-0.058
PctHS18_24               0.0972      0.017      5.746      0.000       0.064
0.130
PctBachDeg25_Over       -0.2674      0.022    -12.066      0.000      -0.311
-0.224
PctPrivateCoverage      -0.0962      0.025     -3.880      0.000      -0.145
-0.048
PctOtherRace            -0.1406      0.017     -8.109      0.000      -0.175
-0.107
PctMarriedHouseholds    -0.0771      0.021     -3.643      0.000      -0.119
-0.036
==============================================================================
Omnibus:                       92.000   Durbin-Watson:                   1.970
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              279.299
Skew:                           0.103   Prob(JB):                     2.24e-61
Kurtosis:                       4.761   Cond. No.                         25.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

2.9.1 Residual Analysis of Training Data of Model-4

[62]:
```python
y_train_pred3 = lr3.predict(X_train_lm3)

errorTermsPlot(y_train,y_train_pred3,'orange','Training','Errors')
```

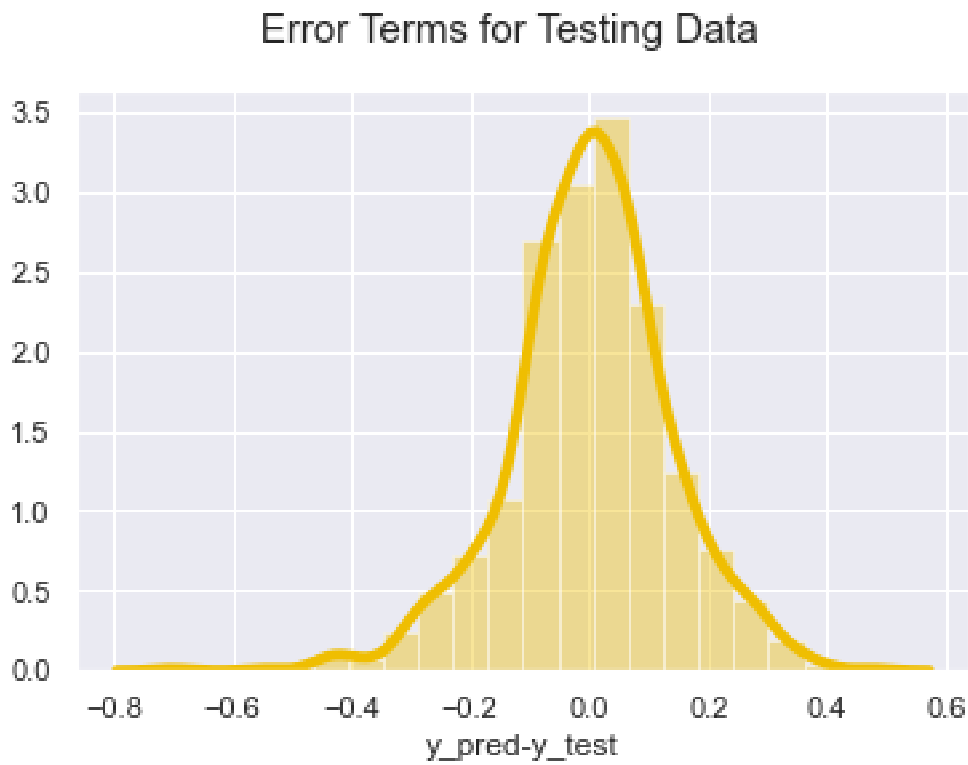
[63]:
```
# Creating X_test_new dataframe by dropping variables from X_test
X_test_new3 = X_test[X_train3.columns.values]

# Adding a constant variable
X_test_new3 = sm.add_constant(X_test_new3)

# Making predictions
y_pred3 = lr3.predict(X_test_new3)
```

[64]:
```
yTest_vs_yPredPlot(y_test,y_pred3,'#08C921')
```



2.9.2 Residual Analysis of Testing Data of Model-4

[65]:
```
errorTermsPlot(y_pred3,y_test,'#EFBE01','Testing','y_pred-y_test')
```



2.10 Selecting the Best Model by checking Error Metrics of all the Data Models

[66]:
```
#Defining the Funtion 'errorMetrics' which will Calculate various␣
↪paramets(Squared error, Mean Squared error,
#Root Mean Squared error and R-Squared Value)
def errorMetrics(y_pred,y):
    error = y_pred - y

    SE = np.square(error) # squared errors
    MSE = np.mean(SE) # mean squared errors
    RMSE = np.sqrt(MSE) # Root Mean Squared Error, RMSE
    Rsquared = 1.0 - (np.var(error) / np.var(y))
    print('Squared Error',round(sum(SE),3) ,
          '\nMean Squared Error' , round(MSE,3),
          '\nRoot Mean Squared Error' , round(RMSE,3),
          '\nR Squared' , round(Rsquared,3),'\n\n')
    return Rsquared
```

[67]:
```
#Passing the pred_variables to the function
pred_variables=[[y_train_pred,y_train],
                [y_pred,y_test],
                [y_train_pred1,y_train],
```

```
                 [y_pred1,y_test],
                 [y_train_pred2,y_train],
                 [y_pred2,y_test],
                 [y_train_pred3,y_train],
                 [y_pred3,y_test]]

#Creating a Dictionary from which we will create a Dataframe later on
d = {'Train R2':[],'Test R2':[]}
j=1

for i in pred_variables:
    print('*'*40)
    if(j%2==0):
        print('---------Test Error of Model',int(j/2),'---------')
    else:
        print('---------Train Error of Model',int(j-(j/2)+1),'--------')

    #Calculating R2 value and appending them to Test R2 and Train R2 respectively
    r = errorMetrics(i[0],i[1])
    if(j%2==0):
        d['Test R2'].append(r)
    else:
        d['Train R2'].append(r)
    j+=1
print('*'*40)

#Creating a DataFrame from the above dictionary
models = pd.DataFrame(d,index=['OLS-all columns','OLS with (VIF<10 & p><0.05)' ,␣
↪'RFE+OLS+VIF' ,'RFE+OLS with (VIF<10 & p<0.05)'])
models
```

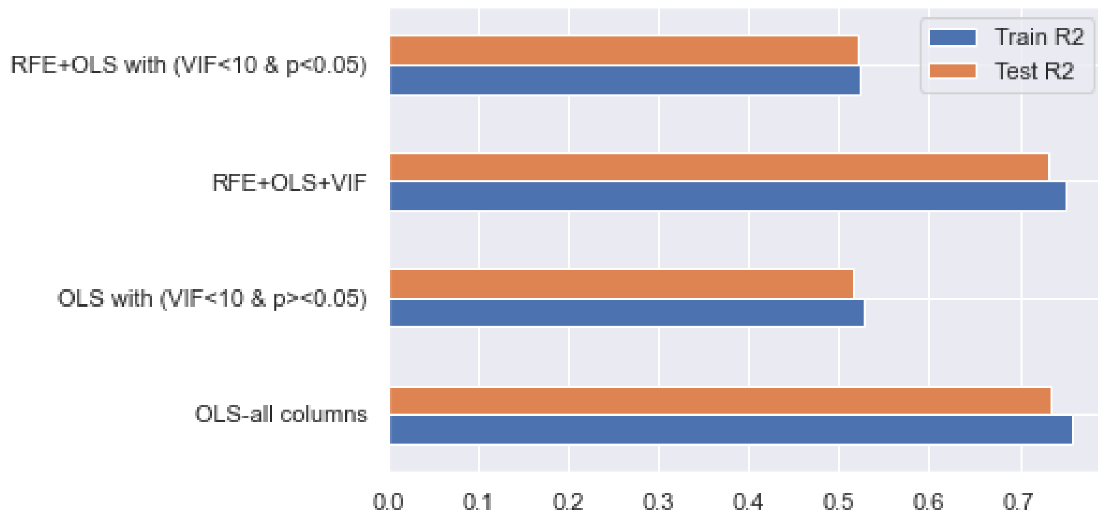
```
****************************************
---------Train Error of Model 1 --------
Squared Error 18.476
Mean Squared Error 0.009
Root Mean Squared Error 0.093
R Squared 0.757


****************************************
---------Test Error of Model 1 ---------
Squared Error 9.433
Mean Squared Error 0.01
Root Mean Squared Error 0.102
R Squared 0.735


****************************************
```

```
---------Train Error of Model 2 --------
Squared Error 35.898
Mean Squared Error 0.017
Root Mean Squared Error 0.13
R Squared 0.528


***************************************
---------Test Error of Model 2 ---------
Squared Error 17.267
Mean Squared Error 0.019
Root Mean Squared Error 0.137
R Squared 0.515


***************************************
---------Train Error of Model 3 --------
Squared Error 18.885
Mean Squared Error 0.009
Root Mean Squared Error 0.094
R Squared 0.752


***************************************
---------Test Error of Model 3 ---------
Squared Error 9.507
Mean Squared Error 0.01
Root Mean Squared Error 0.102
R Squared 0.733


***************************************
---------Train Error of Model 4 --------
Squared Error 36.338
Mean Squared Error 0.017
Root Mean Squared Error 0.131
R Squared 0.523


***************************************
---------Test Error of Model 4 ---------
Squared Error 17.112
Mean Squared Error 0.019
Root Mean Squared Error 0.137
R Squared 0.52


***************************************
```

[67]:
```
                                    Train R2   Test R2
OLS-all columns                     0.757257  0.735211
OLS with (VIF<10 & p><0.05)         0.528354  0.515121
RFE+OLS+VIF                         0.751875  0.733073
RFE+OLS with (VIF<10 & p<0.05)      0.522574  0.519512
```

[68]:
```python
#Plotting the Bar Graph for all the Variables in the DataFrame 'models'
models.plot(kind='barh')
```

[68]: <matplotlib.axes._subplots.AxesSubplot at 0x20b57184790>



We can finally observe that Model-1 and Model-3 have high R2 values for both Test and Train Data but we will not choose them because:

1. Model 1 contains all the Columns and has high Multicollinearity

2. Model 3 contains half the Columns but still has high Multicollinearity.

3. Also there is a possibility that both of these have been overfitted as the y_test vs y_pred graph follows a Linear Relationship and is not Spread all over as it shoul have.

We can choose any one of Model-2 and Model-4 because they both have a better spread in the y_test vs y_pred graph and they both don't have Multicollinearity or any Insignificant Features

[69]:
```python
#Creating Equation for Model-2
l = np.around(np.array(lr1.params.values),3)
s=''
for i in zip(lr1.params.index,l):
    s += str(i[0]) + '    *     ' + str(i[1]) + '    +    '
```

```
print(s)
```

```
const    *    0.382    +    avgAnnCount    *    -0.198    +
avgDeathsPerYear    *    0.269    +    incidenceRate    *    0.378    +
binnedInc    *    -0.051    +    MedianAge    *    -0.141    +
PercentMarried    *    0.129    +    PctHS18_24    *    0.079    +
PctHS25_Over    *    0.063    +    PctBachDeg25_Over    *    -0.223    +
PctUnemployed16_Over    *    0.074    +    PctPublicCoverageAlone    *
0.056    +    PctWhite    *    -0.061    +    PctAsian    *    -0.079    +
PctOtherRace    *    -0.112    +    PctMarriedHouseholds    *    -0.165    +
```

2.10.1 For Model-2 , the equation is:

TARGET_deathRate = const * 0.382 + avgAnnCount * -0.198 + avgDeathsPerYear * 0.269 + incidenceRate * 0.378 + binnedInc * -0.051 + MedianAge * -0.141 + PercentMarried * 0.129 + PctHS18_24 * 0.079 + PctHS25_Over * 0.063 + PctBachDeg25_Over * -0.223 + PctUnemployed16_Over * 0.074 + PctPublicCoverageAlone * 0.056 + PctWhite * -0.061 + PctAsian * -0.079 + PctOtherRace * -0.112 + PctMarriedHouseholds * -0.165

[70]:
```
#Creating Equation for Model-2
l = np.around(np.array(lr3.params.values),3)
s=''
for i in zip(lr3.params.index,l):
    s += str(i[0]) + '    *    ' + str(i[1]) + '    +    '
print(s)
```

```
const    *    0.524    +    avgAnnCount    *    -0.186    +
avgDeathsPerYear    *    0.248    +    incidenceRate    *    0.396    +
medIncome    *    -0.089    +    MedianAgeFemale    *    -0.09    +
PctHS18_24    *    0.097    +    PctBachDeg25_Over    *    -0.267    +
PctPrivateCoverage    *    -0.096    +    PctOtherRace    *    -0.141    +
PctMarriedHouseholds    *    -0.077    +
```

2.10.2 For Model-4, the equation is:

TARGET_deathRate = const * 0.524 + avgAnnCount * -0.186 + avgDeathsPerYear * 0.248 + incidenceRate * 0.396 + medIncome * -0.089 + MedianAgeFemale * -0.09 + PctHS18_24 * 0.097 + PctBachDeg25_Over * -0.267 + PctPrivateCoverage * -0.096 + PctOtherRace * -0.141 + PctMarriedHouseholds * -0.077

I'll personally choose Model 4 because it has the fewest yet efficient predictor variables.

Thank You!