Objective :

- The outcome of this model is to build a multivariate Ordinary Least Squares regression model to predict "TARGET_deathRate"
- The statistical software output including R-squared and Root Mean Squared Error (RMSE)

Project Plan :

Analyzing and Importing data:

Importing data and analyzing it to extract insights that supports for decision making process (like obtaining dependent and independent variables).

Data Cleansing:

-Check nulls (If null: Delete the row or replace with mean, median or mode)
-Foreign values
-Check voids
-Wrong format data

Minimize the Dimension of Dataframe:

-Check correlation between columns
-Pearson correlation is used to check correlation between two continuous column.
-Highly correlated columns are obtained and one of them are dropped

Model building:

-Splitting the data into train and test data sets
-Train the model using decision tree classifier
-Fit the data into the model
-Predicting the response of the test dataset
-Evaluating accuracy of the model with assumptions
-Comparison and Evaluation Metrics -R square -Mean Square Error -OLS

Assumptions:

According to the Gauss–Markov theorem, in a linear regression model the ordinary least squares (OLS) estimator gives the best linear unbiased estimator of the coefficients

-The expectation of errors (residuals) is by checking the residual normailty.
-The errors are uncorrelated by multicollinearity and autocorrelation.
-The errors have equal variance — homoscedasticity of errors.

Recording the Experimental Design

-Load dataset and libraries.
-Clean dataset.
-Carry out data analysis.
-Carry out data modeling.
-Summarize findings.

Challenges

While performing model selection/diagnosis, I performed the following steps in an effort to check for the following assumptions:

-Assess the linearity of the model (parameters)

-Assess the normality of residual distribution

Import library

In [1]:
```
import pandas  as pd #Data manipulation
import numpy as np #Data manipulation
import matplotlib.pyplot as plt # Visualization
import seaborn as sns #Visualization
import plotly.express as px #Visualization
import plotly.graph_objs as go #Visualization
import os

## for statistical tests
import scipy
import statsmodels.formula.api as smf
import statsmodels.api as sm

## for machine learning
from sklearn import model_selection, preprocessing, feature_selection, ensembl
e, linear_model, metrics, decomposition

## for explainer
#from lime import lime_tabular


pd.options.plotting.backend = "plotly"
plt.rcParams['figure.figsize'] = [8,5]
plt.rcParams['font.size'] =14
plt.rcParams['font.weight']= 'bold'
plt.style.use('seaborn-whitegrid')
```

Read the file

In [2]:
```
df=pd.read_excel(r'C:\Users\Roja.s\Downloads\capson project\cancer_reg.xlsx')
df
```

Out[2]:

	avgAnnCount	avgDeathsPerYear	TARGET_deathRate	incidenceRate	medIncome	popEst2(
0	1397.000000	469	164.9	489.800000	61898	260
1	173.000000	70	161.3	411.600000	48127	43
2	102.000000	50	174.7	349.700000	49348	21(
3	427.000000	202	194.8	430.400000	44243	75
4	57.000000	26	144.4	350.100000	49955	10
...	...	...	...	...	...	
3042	1962.667684	15	149.6	453.549422	46961	6
3043	1962.667684	43	150.1	453.549422	48609	37
3044	1962.667684	46	153.9	453.549422	51144	34
3045	1962.667684	52	175.0	453.549422	50745	25
3046	1962.667684	48	213.6	453.549422	41193	37(

3047 rows × 34 columns

Data cleansing

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
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
Geography                  0
AvgHouseholdSize           0
PercentMarried             0
PctNoHS18_24               0
PctHS18_24                 0
PctSomeCol18_24         2285
PctBachDeg18_24            0
PctHS25_Over               0
PctBachDeg25_Over          0
PctEmployed16_Over       152
PctUnemployed16_Over       0
PctPrivateCoverage         0
PctPrivateCoverageAlone  609
PctEmpPrivCoverage         0
PctPublicCoverage          0
PctPublicCoverageAlone     0
PctWhite                   0
PctBlack                   0
PctAsian                   0
PctOtherRace               0
PctMarriedHouseholds       0
BirthRate                  0
dtype: int64
```

In [4]:
```
# replacing null values
df.fillna(df.mean(),inplace=True)
```

```
C:\Users\Roja.s\AppData\Local\Temp\ipykernel_5580\2147398004.py:2: FutureWarn
ing: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only
=None') is deprecated; in a future version this will raise TypeError.  Select
only valid columns before calling the reduction.
  df.fillna(df.mean(),inplace=True)
```

In [5]:
```
# getting the info
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          3047 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       3047 non-null   float64
 22  PctUnemployed16_Over     3047 non-null   float64
 23  PctPrivateCoverage       3047 non-null   float64
 24  PctPrivateCoverageAlone  3047 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
```

In [6]:
```
len(df[df.duplicated()])
```

Out[6]: 0

To find the missing value

In [106]:
```
num_vars=df.columns[df.dtypes!='object']

df[num_vars].isnull().sum().sort_values(ascending = False)/len(df)
```

Out[106]:
```
PctSomeCol18_24            0.749918
PctPrivateCoverageAlone    0.199869
PctEmployed16_Over         0.049885
avgAnnCount                0.000000
PctHS25_Over               0.000000
PctBlack                   0.000000
PctWhite                   0.000000
PctPublicCoverageAlone     0.000000
PctPublicCoverage          0.000000
PctEmpPrivCoverage         0.000000
PctPrivateCoverage         0.000000
PctUnemployed16_Over       0.000000
PctBachDeg25_Over          0.000000
PctBachDeg18_24            0.000000
TARGET_deathRate           0.000000
PctHS18_24                 0.000000
PctNoHS18_24               0.000000
PercentMarried             0.000000
MedianAgeMale              0.000000
MedianAge                  0.000000
studyPerCap                0.000000
povertyPercent             0.000000
medIncome                  0.000000
incidenceRate              0.000000
PctMarriedHouseholds       0.000000
dtype: float64
```

We observe that :

-Columns (PctEmployed16_Over,PctPrivateCoverageAlone) has missing values which can be droped by observation made above.
-Column (PctSomeCol18_24) has more than 75% missing values and hence we drop that column.
-Column (Geography) is of object datatype and since it had unique values we can't encode it and have to drop it as well.
-Column (binnedInc) is of series type but since we alreaady have a median income column we dont need this column.

correlation

In [7]:
```
correlation=df.corr().abs()
correlation
```

Out[7]:

	avgAnnCount	avgDeathsPerYear	TARGET_deathRate	incidenceRate	m
avgAnnCount	1.000000	0.939408	0.143532	0.073553	
avgDeathsPerYear	0.939408	1.000000	0.090715	0.062690	
TARGET_deathRate	0.143532	0.090715	1.000000	0.449432	
incidenceRate	0.073553	0.062690	0.449432	1.000000	
medIncome	0.269145	0.223207	0.428615	0.001036	
popEst2015	0.926894	0.977634	0.120073	0.026912	
povertyPercent	0.135694	0.066918	0.429389	0.009046	
studyPerCap	0.082071	0.063488	0.022285	0.077283	
MedianAge	0.024098	0.024599	0.004375	0.018089	
MedianAgeMale	0.124969	0.148487	0.021929	0.014733	
MedianAgeFemale	0.122844	0.144069	0.012048	0.009106	
AvgHouseholdSize	0.064788	0.086161	0.036905	0.118400	
PercentMarried	0.106108	0.181029	0.266820	0.119524	
PctNoHS18_24	0.143327	0.136794	0.088463	0.170762	
PctHS18_24	0.182054	0.151418	0.261976	0.022644	
PctSomeCol18_24	0.070159	0.063322	0.094765	0.038442	
PctBachDeg18_24	0.284176	0.259761	0.287817	0.046835	
PctHS25_Over	0.311375	0.295929	0.404589	0.121725	
PctBachDeg25_Over	0.321021	0.293210	0.485477	0.038177	
PctEmployed16_Over	0.197981	0.125015	0.397487	0.004622	
PctUnemployed16_Over	0.009016	0.069701	0.378412	0.099979	
PctPrivateCoverage	0.132244	0.056183	0.386066	0.105174	
PctPrivateCoverageAlone	0.166674	0.112221	0.326067	0.098846	
PctEmpPrivCoverage	0.202349	0.160124	0.267399	0.149825	
PctPublicCoverage	0.173548	0.131687	0.404572	0.046109	
PctPublicCoverageAlone	0.093699	0.027338	0.449358	0.040812	
PctWhite	0.136501	0.187159	0.177400	0.014510	
PctBlack	0.031376	0.084607	0.257024	0.113489	
PctAsian	0.435071	0.443074	0.186331	0.008123	
PctOtherRace	0.209184	0.215149	0.189894	0.208748	
PctMarriedHouseholds	0.106221	0.160266	0.293325	0.152176	
BirthRate	0.034508	0.074420	0.087407	0.118181	

32 rows × 32 columns

In [8]:
```
#Highly correlated columns
hcorr=set()
x=0.90
for i in range (len(correlation.columns)):
    for j in range(i):
        if(correlation.iloc[i,j]>x):
            colname=correlation.columns[i]
            hcorr.add(colname)
print(hcorr)
```

```
{'avgDeathsPerYear', 'MedianAgeFemale', 'popEst2015'}
```

Droping the highly correlated and missing values

In [9]:
```
df.drop('avgDeathsPerYear',axis=1,inplace=True)
df.drop('popEst2015',axis=1,inplace=True)
df.drop('MedianAgeFemale',axis=1,inplace=True)
df.drop('Geography',axis=1,inplace=True)
df.drop('binnedInc',axis=1,inplace=True)
```

In [10]:
```
df.drop('AvgHouseholdSize',axis=1,inplace=True)
```

In [11]:
```
df.head()
```

Out[11]:

	avgAnnCount	TARGET_deathRate	incidenceRate	medIncome	povertyPercent	studyPerCap
0	1397.0	164.9	489.8	61898	11.2	499.748204
1	173.0	161.3	411.6	48127	18.6	23.111234
2	102.0	174.7	349.7	49348	14.6	47.560164
3	427.0	194.8	430.4	44243	17.1	342.637253
4	57.0	144.4	350.1	49955	12.5	0.000000

5 rows × 28 columns

In [12]:
```
df.drop('BirthRate',axis=1,inplace=True)
df.drop('PctAsian',axis=1,inplace=True)
df.drop('PctOtherRace',axis=1,inplace=True)
```

In [13]:
```
df.head()
```

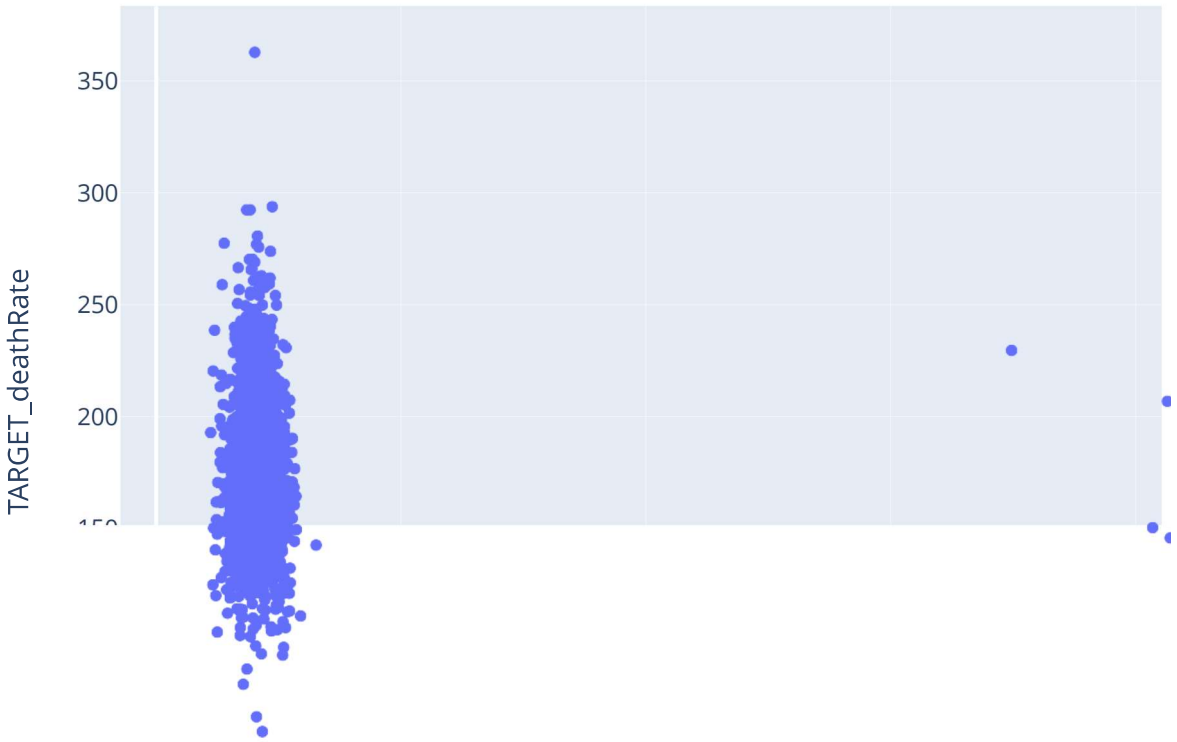
Out[13]:

	avgAnnCount	TARGET_deathRate	incidenceRate	medIncome	povertyPercent	studyPerCap
0	1397.0	164.9	489.8	61898	11.2	499.748204
1	173.0	161.3	411.6	48127	18.6	23.111234
2	102.0	174.7	349.7	49348	14.6	47.560164
3	427.0	194.8	430.4	44243	17.1	342.637253
4	57.0	144.4	350.1	49955	12.5	0.000000

5 rows × 25 columns

To Observe Outliers

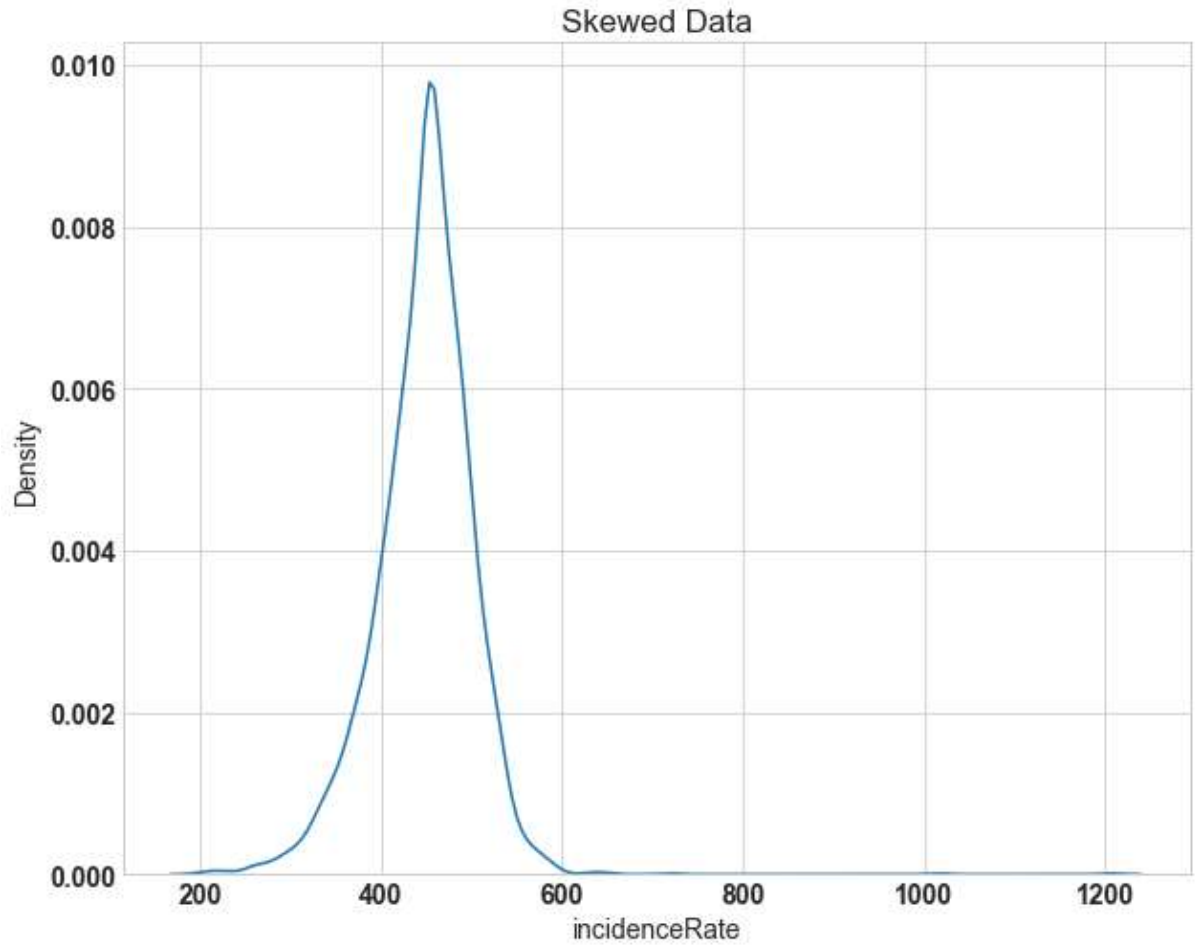
In [31]:
```
px.scatter(df,x='MedianAge',y='TARGET_deathRate')
```



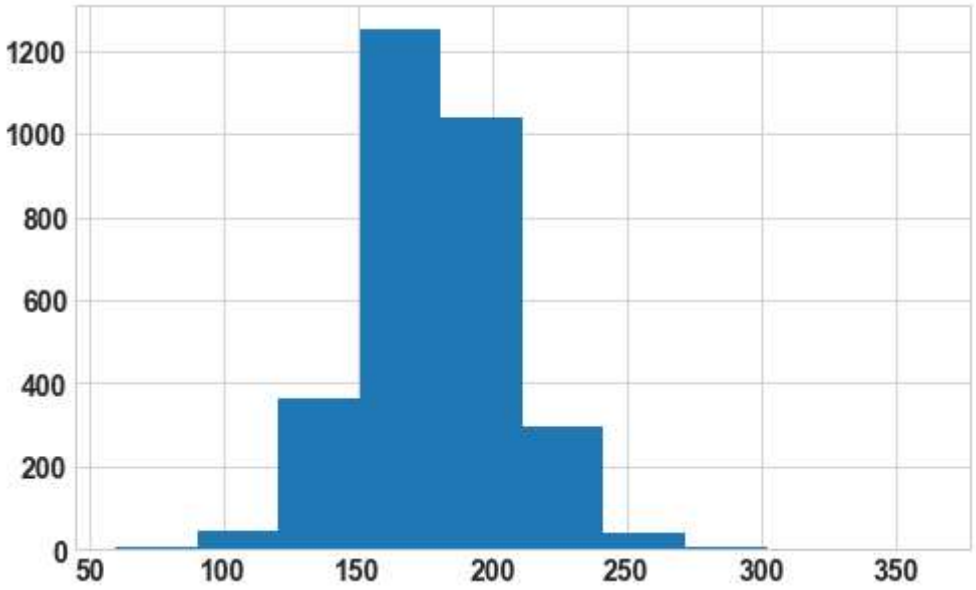
In [ ]:
```
similar way of finding outlier for other variables
# x1 = df.plot.scatter(x='incidenceRate',y='TARGET_deathRate',c='DarkBlue')
# x1 = df.plot.scatter(x='medIncome',y='TARGET_deathRate',c='DarkBlue')
# x1 = df.plot.scatter(x='povertyPercent',y='TARGET_deathRate',c='DarkBlue')
# x1 = df.plot.scatter(x='PercentMarried',y='TARGET_deathRate',c='DarkBlue')
# x1 = df.plot.scatter(x='PctHS25_Over',y='TARGET_deathRate',c='DarkBlue')
# x1 = df.plot.scatter(x='PctEmployed16_Over',y='TARGET_deathRate',c='DarkBlu
e')
# x1 = df.plot.scatter(x='PctUnemployed16_Over',y='TARGET_deathRate',c='DarkBl
ue')
# x1 = df.plot.scatter(x='minIncome',y='TARGET_deathRate',c='DarkBlue')
```

In [60]:
```
fig, (ax1) = plt.subplots(ncols=1, figsize=(10, 8))
ax1.set_title('Skewed Data')
#sns.kdeplot(df['avgAnnCount'], ax=ax1)
#sns.kdeplot(df['avgDeathsPerYear'], ax=ax1)
sns.kdeplot(df['incidenceRate'], ax=ax1)
#sns.kdeplot(df['popEst2015'], ax=ax1)
#sns.kdeplot(df['PctWhite'], ax=ax1)
```

Out[60]:
```
<AxesSubplot:title={'center':'Skewed Data'}, xlabel='incidenceRate', ylabel
='Density'>
```



In [32]:
```
plt.hist(df['TARGET_deathRate'])
plt.show()
```



In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3047 entries, 0 to 3046
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   avgAnnCount              3047 non-null   float64
 1   TARGET_deathRate         3047 non-null   float64
 2   incidenceRate            3047 non-null   float64
 3   medIncome                3047 non-null   int64  
 4   povertyPercent           3047 non-null   float64
 5   studyPerCap              3047 non-null   float64
 6   MedianAge                3047 non-null   float64
 7   MedianAgeMale            3047 non-null   float64
 8   PercentMarried           3047 non-null   float64
 9   PctNoHS18_24             3047 non-null   float64
 10  PctHS18_24               3047 non-null   float64
 11  PctSomeCol18_24          3047 non-null   float64
 12  PctBachDeg18_24          3047 non-null   float64
 13  PctHS25_Over             3047 non-null   float64
 14  PctBachDeg25_Over        3047 non-null   float64
 15  PctEmployed16_Over       3047 non-null   float64
 16  PctUnemployed16_Over     3047 non-null   float64
 17  PctPrivateCoverage       3047 non-null   float64
 18  PctPrivateCoverageAlone  3047 non-null   float64
 19  PctEmpPrivCoverage       3047 non-null   float64
 20  PctPublicCoverage        3047 non-null   float64
 21  PctPublicCoverageAlone   3047 non-null   float64
 22  PctWhite                 3047 non-null   float64
 23  PctBlack                 3047 non-null   float64
 24  PctMarriedHouseholds     3047 non-null   float64
dtypes: float64(24), int64(1)
memory usage: 595.2 KB
```

Linear Regression

In [15]:
```
## Sepaerating the input data and target data
from sklearn.model_selection import train_test_split
X = df.drop('TARGET_deathRate',axis=1) # Independet variable
y = df['TARGET_deathRate'] # dependent variable

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_s
tate=23)
```

In [16]:
```
# Scikit Learn module
from sklearn.linear_model import LinearRegression

# Building the model
lin_reg = LinearRegression()

# Training the model with the datset
lin_reg.fit(X_train,y_train)
```

Out[16]:
```
LinearRegression()
```

Comparison and Evaluation Metrics

R square
Mean Square Error

In [33]:
```
# sklearn regression module
y_pred_sk = lin_reg.predict(X_test)

#Evaluvation: MSE
from sklearn.metrics import mean_squared_error
J_mse_sk = mean_squared_error(y_pred_sk, y_test)

# R_square
R_square_sk = lin_reg.score(X_test,y_test)
print('The Mean Square Error(MSE) or J(theta) is: ',J_mse_sk)
print('R square obtain for scikit learn library is :',R_square_sk)
```

```
The Mean Square Error(MSE) or J(theta) is:  441.1526547414508
R square obtain for scikit learn library is : 0.4575028105224187
```

- For further calculating the accuracy of this prediction a mathematical tool is used, which is R-squared Regression Analysis The value of R-squared is between 0 and 1. The Rsquare value obtained for us is 0.44 which is less than 1
- And if the coefficient of R-squared is 1 (or 100%) means that prediction of the dependent variable has been perfect and accurate.

Model Building

In [32]:
```
import statsmodels.api as sm

X_constant = sm.add_constant(X_train)
lin_reg_sm= sm.OLS(y_train,X_constant).fit()
lin_reg_sm.summary()
```

Out[32]:

OLS Regression Results

Dep. Variable:	TARGET_deathRate	**R-squared:**	0.519
Model:	OLS	**Adj. R-squared:**	0.514
Method:	Least Squares	**F-statistic:**	108.5
Date:	Sun, 12 Jun 2022	**Prob (F-statistic):**	0.00
Time:	20:11:19	**Log-Likelihood:**	-10646.
No. Observations:	2437	**AIC:**	2.134e+04
Df Residuals:	2412	**BIC:**	2.149e+04
Df Model:	24		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	134.2562	16.655	8.061	0.000	101.597	166.915
avgAnnCount	-0.0010	0.000	-3.309	0.001	-0.002	-0.000
incidenceRate	0.2036	0.008	25.137	0.000	0.188	0.219
medIncome	5.141e-05	8.66e-05	0.594	0.553	-0.000	0.000
povertyPercent	0.3942	0.176	2.244	0.025	0.050	0.739
studyPerCap	-0.0002	0.001	-0.267	0.789	-0.002	0.001
MedianAge	-0.0111	0.009	-1.265	0.206	-0.028	0.006
MedianAgeMale	-0.3005	0.156	-1.925	0.054	-0.607	0.006
PercentMarried	1.1083	0.178	6.220	0.000	0.759	1.458
PctNoHS18_24	-0.2303	0.065	-3.534	0.000	-0.358	-0.103
PctHS18_24	0.2646	0.059	4.496	0.000	0.149	0.380
PctSomeCol18_24	-0.0090	0.081	-0.111	0.911	-0.167	0.149
PctBachDeg18_24	-0.0386	0.118	-0.328	0.743	-0.270	0.193
PctHS25_Over	0.5913	0.103	5.730	0.000	0.389	0.794
PctBachDeg25_Over	-0.9198	0.163	-5.657	0.000	-1.239	-0.601
PctEmployed16_Over	-0.5790	0.108	-5.367	0.000	-0.790	-0.367
PctUnemployed16_Over	0.3094	0.181	1.708	0.088	-0.046	0.665
PctPrivateCoverage	-0.4288	0.151	-2.839	0.005	-0.725	-0.133
PctPrivateCoverageAlone	0.0497	0.094	0.530	0.596	-0.134	0.233
PctEmpPrivCoverage	0.4130	0.113	3.659	0.000	0.192	0.634
PctPublicCoverage	-0.4616	0.243	-1.903	0.057	-0.937	0.014
PctPublicCoverageAlone	0.6048	0.308	1.965	0.050	0.001	1.208
PctWhite	-0.0568	0.057	-0.998	0.318	-0.168	0.055
PctBlack	-0.0257	0.055	-0.469	0.639	-0.133	0.082
PctMarriedHouseholds	-1.2876	0.169	-7.604	0.000	-1.620	-0.956

Omnibus:	101.870	**Durbin-Watson:**	1.956
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	322.732
Skew:	0.008	**Prob(JB):**	8.31e-71
Kurtosis:	4.783	**Cond. No.**	2.09e+06

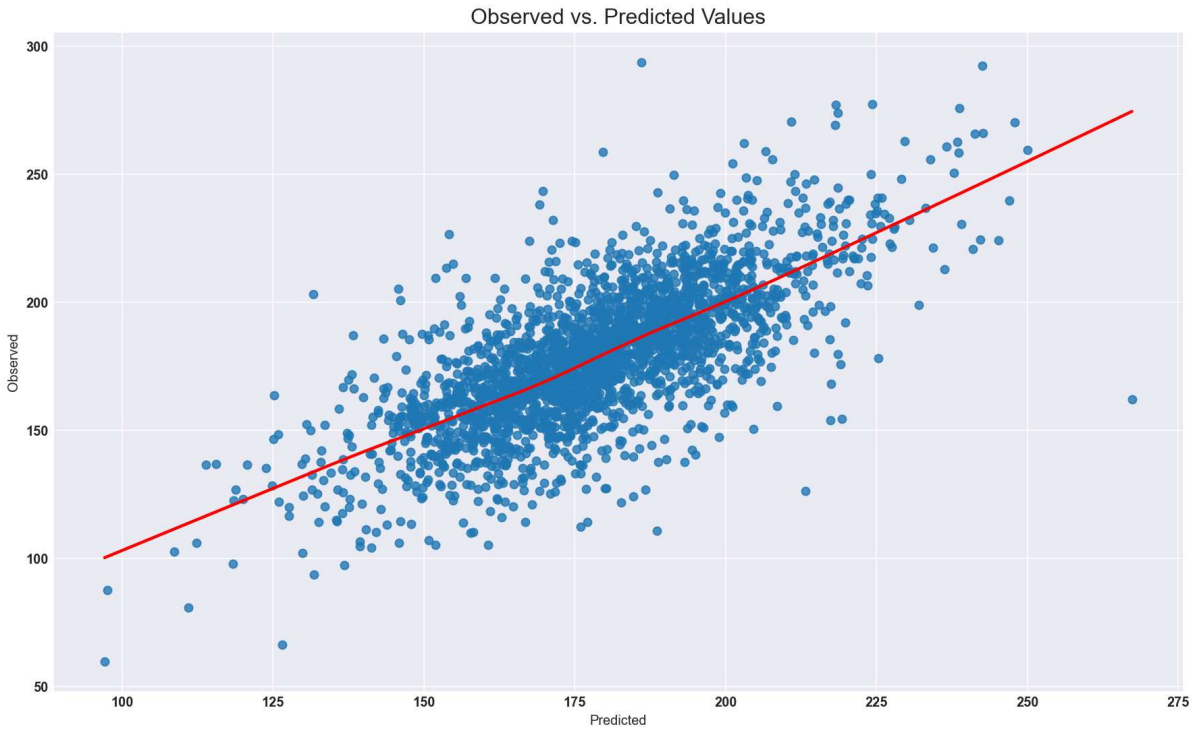
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.09e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Assumptions:

According to the Gauss–Markov theorem, in a linear regression model the ordinary least squares (OLS) estimator gives the best linear unbiased estimator of the coefficients

-The expectation of errors (residuals) is by checking the residual normailty.
-The errors are uncorrelated by multicollinearity and autocorrelation.
-The errors have equal variance — homoscedasticity of errors.

In [19]:
```
#checking for linearity:

%matplotlib inline
%config InlineBackend.figure_format ='retina'
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms
sns.set_style('darkgrid')
sns.mpl.rcParams['figure.figsize'] = (15.0, 9.0)

def linearity_test(model, y):
    '''
    Function for visually inspecting the assumption of linearity in a linear r
egression model.
    It plots observed vs. predicted values and residuals vs. predicted values.

    Args:
    * model - fitted OLS model from statsmodels
    * y - observed values
    '''
    fitted_vals = model.predict()
    resids = model.resid

    fig, ax = plt.subplots(1,1)

    sns.regplot(x=fitted_vals, y=y, lowess=True, ax=ax, line_kws={'color': 're
d'})
    ax.set_title('Observed vs. Predicted Values', fontsize=16)
    ax.set(xlabel='Predicted', ylabel='Observed')

    '''sns.regplot(x=fitted_vals, y=resids, lowess=True, ax=ax[1], line_kws=
{'color': 'red'})
    ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
    ax[1].set(xlabel='Predicted', ylabel='Residuals)'''

linearity_test(lin_reg_sm, y_train)
```

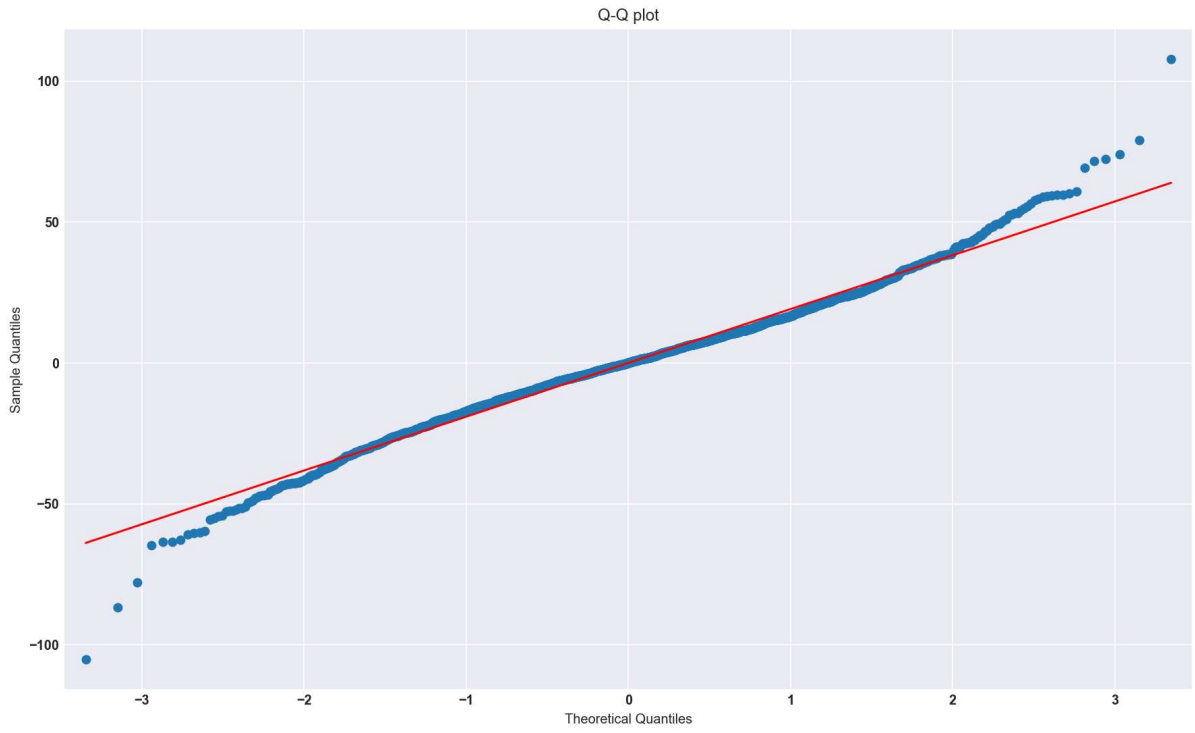


Inference from checking the linearity

We successfully found the line of best fit and fitted it into the data points using the least square regression method in machine learning.The inspection of the plots shows that the linearity assumption is satisfied.The least square error shows high accuracy, it implies that the dataset is linear in nature.The most common metric for evaluating linear regression model performance is by root mean squared error which is obtained as 0.457.

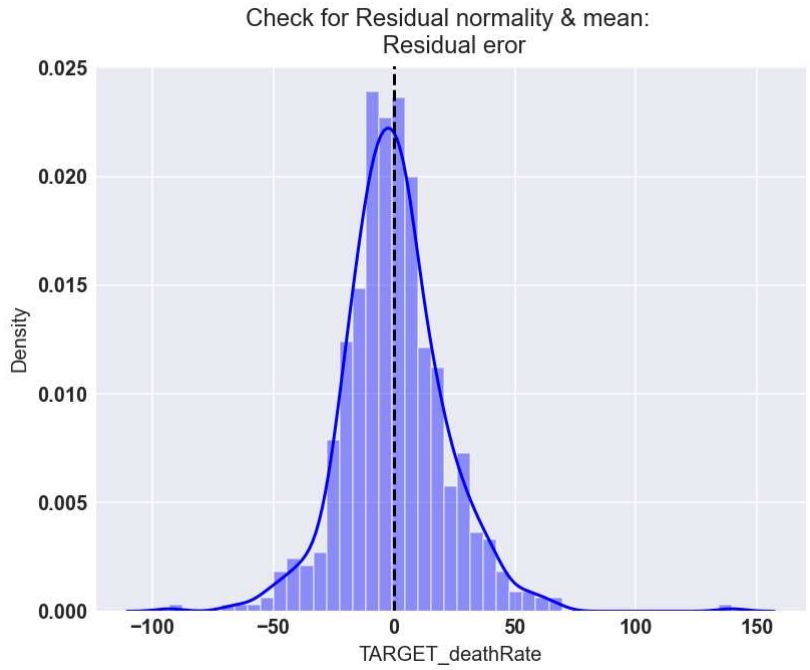
In [20]:
```
from scipy import stats

def normality_of_residuals_test(model):
    '''
    Function for drawing the normal QQ-plot of the residuals and running 4 sta
tistical tests to
    investigate the normality of residuals.

    Arg:
    * model - fitted OLS models from statsmodels
    '''
    sm.ProbPlot(model.resid).qqplot(line='s');
    plt.title('Q-Q plot');

    jb = stats.jarque_bera(model.resid)
    print(f'Jarque-Bera test ---- statistic: {jb[0]:.4f}, p-value: {jb[1]}')

    print('If the returned AD statistic is larger than the critical value, the
n for the 5% significance level, the null hypothesis that the data come from t
he Normal distribution should be rejected. ')

normality_of_residuals_test(lin_reg_sm)
```

```
Jarque-Bera test ---- statistic: 322.7315, p-value: 0.0
If the returned AD statistic is larger than the critical value, then for the
5% significance level, the null hypothesis that the data come from the Normal
distribution should be rejected.
```



In [21]:
```
# Check for Residual  mean
f = plt.figure(figsize=(14,5))
ax = f.add_subplot(122)
sns.distplot((y_test - y_pred_sk),ax=ax,color='b')
ax.axvline((y_test - y_pred_sk).mean(),color='k',linestyle='--')

ax.set_title('Check for Residual normality & mean: \n Residual eror');
print(f'Expectation (mean) of residuals is :', {lin_reg_sm.resid.mean()})
```

```
C:\Users\Roja.s\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level f
unction with similar flexibility) or `histplot` (an axes-level function for h
istograms).
  warnings.warn(msg, FutureWarning)

Expectation (mean) of residuals is : {-2.1979299791768318e-13}
```

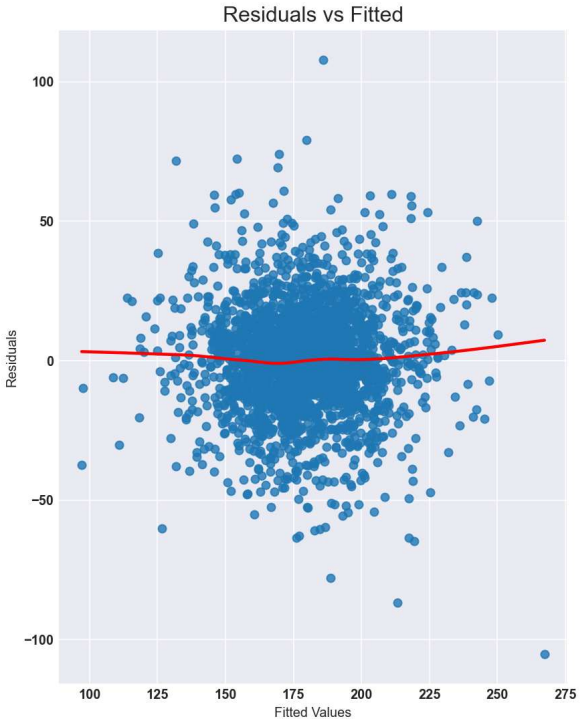


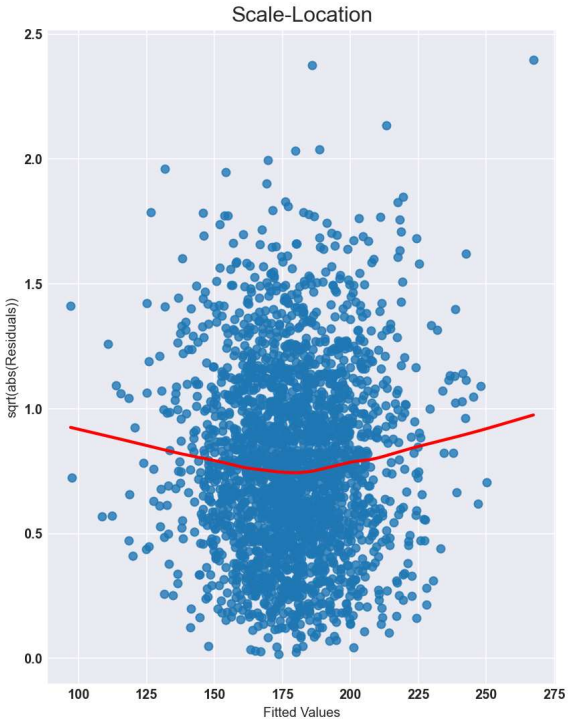
Inference from residual mean:

The residual mean is zero and residual error plot right skewed

In [22]:
```
# homoscedasticity

%matplotlib inline
%config InlineBackend.figure_format ='retina'
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms
sns.set_style('darkgrid')
sns.mpl.rcParams['figure.figsize'] = (15.0, 9.0)

def homoscedasticity_test(model):
    '''
    Function for testing the homoscedasticity of residuals in a linear regress
ion model.
    It plots residuals and standardized residuals vs. fitted values and runs B
reusch-Pagan and Goldfeld-Quandt tests.

    Args:
    * model - fitted OLS model from statsmodels
    '''
    fitted_vals = model.predict()
    resids = model.resid
    resids_standardized = model.get_influence().resid_studentized_internal

    fig, ax = plt.subplots(1,2)

    sns.regplot(x=fitted_vals, y=resids, lowess=True, ax=ax[0], line_kws={'col
or': 'red'})
    ax[0].set_title('Residuals vs Fitted', fontsize=16)
    ax[0].set(xlabel='Fitted Values', ylabel='Residuals')

    sns.regplot(x=fitted_vals, y=np.sqrt(np.abs(resids_standardized)), lowess=
True, ax=ax[1], line_kws={'color': 'red'})
    ax[1].set_title('Scale-Location', fontsize=16)
    ax[1].set(xlabel='Fitted Values', ylabel='sqrt(abs(Residuals))')

    bp_test = pd.DataFrame(sms.het_breuschpagan(resids, model.model.exog),
                           columns=['value'],
                           index=['Lagrange multiplier statistic', 'p-value',
'f-value', 'f p-value'])


    print('\n Breusch-Pagan test ----')
    print(bp_test)

    print('\n Residuals plots ----')

homoscedasticity_test(lin_reg_sm)
```

```
 Breusch-Pagan test ----
                                    value
Lagrange multiplier statistic  1.935670e+02
p-value                        1.827577e-28
f-value                        8.671303e+00
f p-value                      8.288102e-30

 Residuals plots ----
```





Inference from homoscedasticity:

Homoscedasticity should have equal variance of residuals, In the above graph by using the statistical tests: Breusch-Pagan Test the Q-Q plot shows as value log value greater than 1.5 trends to increase.The plot is exhibit heteroscedastic.

The results indicate that the assumption is not satisfied and we should reject the hypothesis of homoscedasticity.

Detecting Multicollinearity using VIF

In [23]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = [variance_inflation_factor(X_constant.values, i) for i in range(X_consta
nt.shape[1])]
pd.DataFrame({'vif': vif[1:]}, index=X.columns).T
```

Out[23]:

	avgAnnCount	incidenceRate	medIncome	povertyPercent	studyPerCap	MedianAge	Median
vif	1.230191	1.230739	7.1551	8.361797	1.044012	1.030116	

1 rows × 24 columns

In [24]:
```
def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shap
e[1])]

    return(vif)
```

Fixing Multicollinearity

Dropping one of the correlated features will help in bringing down the multicollinearity between correlated features

VIF (Variable Inflation Factors),VIF exceeding 5 or 10 indicates high multicollinearity between this independent variable and the others

In [27]:
```
X = df.drop(['PctPrivateCoverageAlone','PctPrivateCoverage','PctPublicCoverage
Alone','PctPublicCoverage'],axis=1)
calc_vif(X)
```

Out[27]:

	variables	VIF
0	avgAnnCount	1.448599
1	TARGET_deathRate	83.286304
2	incidenceRate	100.933031
3	medIncome	111.354109
4	povertyPercent	32.099342
5	studyPerCap	1.127925
6	MedianAge	2.043180
7	MedianAgeMale	134.644389
8	PercentMarried	567.600255
9	PctNoHS18_24	9.642794
10	PctHS18_24	27.880753
11	PctSomeCol18_24	68.267230
12	PctBachDeg18_24	5.585619
13	PctHS25_Over	85.637121
14	PctBachDeg25_Over	33.148504
15	PctEmployed16_Over	144.521749
16	PctUnemployed16_Over	13.845285
17	PctEmpPrivCoverage	87.980025
18	PctWhite	142.490279
19	PctBlack	5.276049
20	PctMarriedHouseholds	455.692054

In [26]:
```
VIF = 1/(1- R_square_sk)
VIF
```

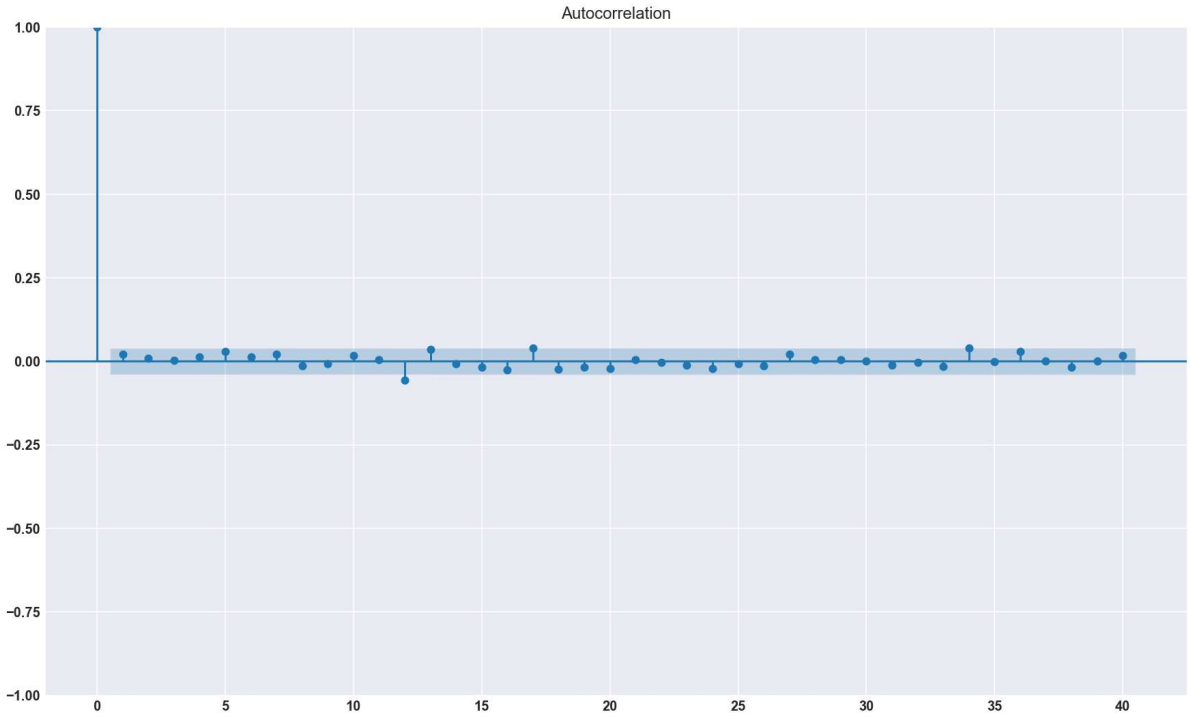
Out[26]: 1.8433275220522134

VIF is not exceeding 5 or 10 which indicates it has low multicollinearity between this independent variables.

Autocorrelation

In [61]:
```
import statsmodels.tsa.api as smt

acf = smt.graphics.plot_acf(lin_reg_sm.resid, lags=40 , alpha=0.05)
acf.show()
```

```
C:\Users\Roja.s\AppData\Local\Temp\ipykernel_20360\1022557584.py:4: UserWarni
ng: Matplotlib is currently using module://matplotlib_inline.backend_inline,
which is a non-GUI backend, so cannot show the figure.
  acf.show()
```



In [62]:
```
lin_reg_sm.summary() # to check DQ statistic
```

Out[62]:

OLS Regression Results

Dep. Variable:	TARGET_deathRate	**R-squared:**	0.519
Model:	OLS	**Adj. R-squared:**	0.514
Method:	Least Squares	**F-statistic:**	108.5
Date:	Sun, 12 Jun 2022	**Prob (F-statistic):**	0.00
Time:	09:41:12	**Log-Likelihood:**	-10646.
No. Observations:	2437	**AIC:**	2.134e+04
Df Residuals:	2412	**BIC:**	2.149e+04
Df Model:	24		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	134.2562	16.655	8.061	0.000	101.597	166.915
avgAnnCount	-0.0010	0.000	-3.309	0.001	-0.002	-0.000
incidenceRate	0.2036	0.008	25.137	0.000	0.188	0.219
medIncome	5.141e-05	8.66e-05	0.594	0.553	-0.000	0.000
povertyPercent	0.3942	0.176	2.244	0.025	0.050	0.739
studyPerCap	-0.0002	0.001	-0.267	0.789	-0.002	0.001
MedianAge	-0.0111	0.009	-1.265	0.206	-0.028	0.006
MedianAgeMale	-0.3005	0.156	-1.925	0.054	-0.607	0.006
PercentMarried	1.1083	0.178	6.220	0.000	0.759	1.458
PctNoHS18_24	-0.2303	0.065	-3.534	0.000	-0.358	-0.103
PctHS18_24	0.2646	0.059	4.496	0.000	0.149	0.380
PctSomeCol18_24	-0.0090	0.081	-0.111	0.911	-0.167	0.149
PctBachDeg18_24	-0.0386	0.118	-0.328	0.743	-0.270	0.193
PctHS25_Over	0.5913	0.103	5.730	0.000	0.389	0.794
PctBachDeg25_Over	-0.9198	0.163	-5.657	0.000	-1.239	-0.601
PctEmployed16_Over	-0.5790	0.108	-5.367	0.000	-0.790	-0.367
PctUnemployed16_Over	0.3094	0.181	1.708	0.088	-0.046	0.665
PctPrivateCoverage	-0.4288	0.151	-2.839	0.005	-0.725	-0.133
PctPrivateCoverageAlone	0.0497	0.094	0.530	0.596	-0.134	0.233
PctEmpPrivCoverage	0.4130	0.113	3.659	0.000	0.192	0.634
PctPublicCoverage	-0.4616	0.243	-1.903	0.057	-0.937	0.014
PctPublicCoverageAlone	0.6048	0.308	1.965	0.050	0.001	1.208
PctWhite	-0.0568	0.057	-0.998	0.318	-0.168	0.055
PctBlack	-0.0257	0.055	-0.469	0.639	-0.133	0.082
PctMarriedHouseholds	-1.2876	0.169	-7.604	0.000	-1.620	-0.956

Omnibus:	101.870	**Durbin-Watson:**	1.956
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	322.732
Skew:	0.008	**Prob(JB):**	8.31e-71
Kurtosis:	4.783	**Cond. No.**	2.09e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.09e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Summary:

From our analysis and modeling, it is clear that the modeling algorithm is good fit for our data.

The model assumption linear regression as follows

In our model the actual vs predicted is plotted and the curve is linear in nature
The residual mean is zero and residual error plot right skewed
*Q-Q plot shows as value log value greater than 1.5 trends to increase.The plot is exhibit heteroscedastic*Variance inflation factor value is less than 5, so no multicollearity.
*The approach which is used for autocorrelation check is the Durbin-Watson test in which the test statistic always has a value between 0 and 4 but in our experimentaion we obtained the value of 2 there is no autocorrelation.