Multiple Linear Regression Challenge Your Task: Build a multivariate Ordinary Least Squares regression model to predict "TARGET_deathRate"

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Checking the encoding of the dataset Use package called "chardet"

In [ ]:
```
import chardet
with open(file, 'rb') as rawdata:
    result = chardet.detect(rawdata.read(100000))
result
```

Importing the dataset

In [21]:
```
file = '/Users/anuram/Library/Mobile Documents/com~apple~CloudDocs/Linear Regression /Cancer datset/cancer_reg.csv'
df = pd.read_csv(file, encoding='ISO-8859-1')
```

Data summary and description df has 3047 rows and 34 columns column names of the dataframe is ['avgAnnCount', 'avgDeathsPerYear', 'TARGET_deathRate', 'incidenceRate', 'medIncome', 'popEst2015', 'povertyPercent', 'studyPerCap', 'binnedInc', 'MedianAge', 'MedianAgeMale', 'MedianAgeFemale', 'Geography', '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']

In [22]:
```
df.shape
df.head()
df.columns
df.describe()
```

Out[22]:

	avgAnnCount	avgDeathsPerYear	TARGET_deathRate	incidenceRate	medIncome	popEst2015	povertyPercent	studyPerCap	MedianAge	MedianAgeMale	...	PctPrivateCoverageAlone	PctEm
count	3047.000000	3047.000000	3047.000000	3047.000000	3047.000000	3.047000e+03	3047.000000	3047.000000	3047.000000	3047.000000	...	2438.000000	
mean	606.338544	185.965868	178.664063	448.268586	47063.281917	1.026374e+05	16.878175	155.399415	45.272333	39.570725	...	48.453774	
std	1416.356223	504.134286	27.751511	54.560733	12040.090836	3.290592e+05	6.409087	529.628366	45.304480	5.226017	...	10.083006	
min	6.000000	3.000000	59.700000	201.300000	22640.000000	8.270000e+02	3.200000	0.000000	22.300000	22.400000	...	15.700000	
25%	76.000000	28.000000	161.200000	420.300000	38882.500000	1.168400e+04	12.150000	0.000000	37.700000	36.350000	...	41.000000	
50%	171.000000	61.000000	178.100000	453.549422	45207.000000	2.664300e+04	15.900000	0.000000	41.000000	39.600000	...	48.700000	
75%	518.000000	149.000000	195.200000	480.850000	52492.000000	6.867100e+04	20.400000	83.650776	44.000000	42.500000	...	55.600000	
max	38150.000000	14010.000000	362.800000	1206.900000	125635.000000	1.017029e+07	47.400000	9762.308998	624.000000	64.700000	...	78.900000	

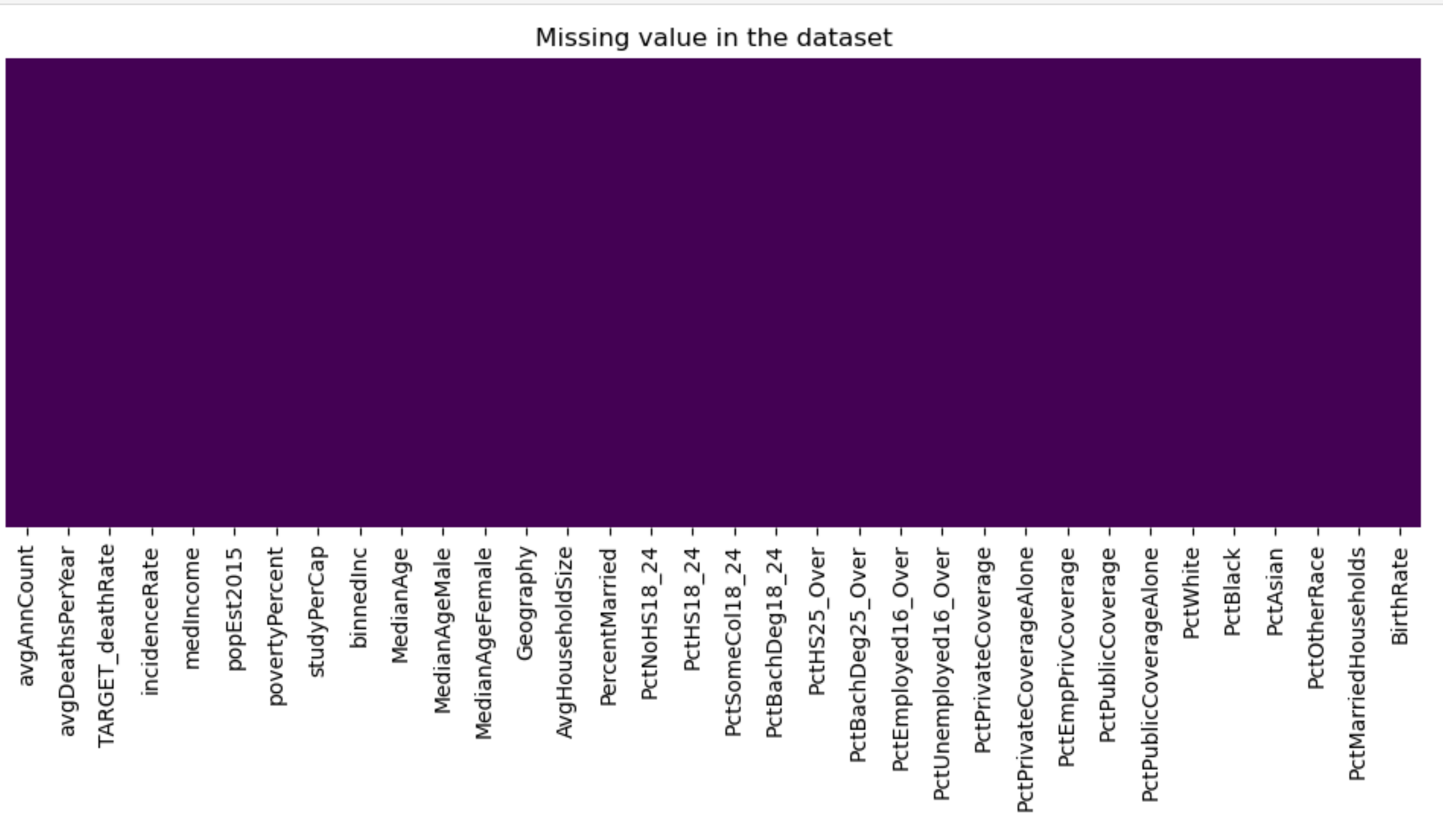
8 rows × 32 columns

Check for Missing values

There are missing values in

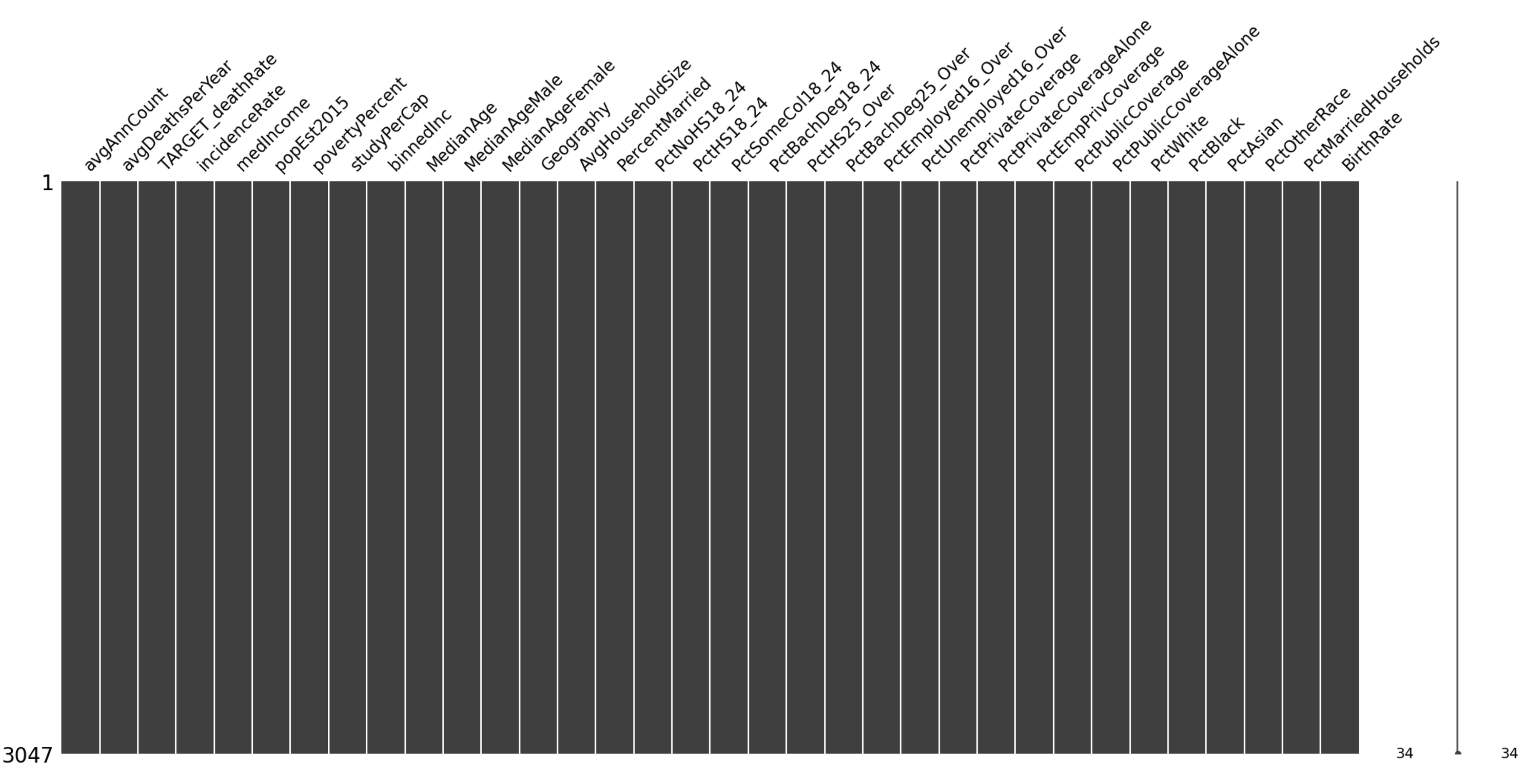
1. PctSomeCol18_24: Percent of county residents ages 18-24 highest education attained: some college
2. PctEmployed16_Over: Percent of county residents ages 16 and over employed
3. PctPrivateCoverageAlone: Percent of county residents with private health coverage alone (no public assistance)

In [31]:
```
plt.figure(figsize=(12,4))
sns.heatmap(df.isnull(),cbar=False,cmap='viridis',yticklabels=False)
plt.title('Missing value in the dataset');
```



In [30]:
```
# Use missing no package to visualize NaN (missing) values
import missingno as msno
msno.matrix(df)
```

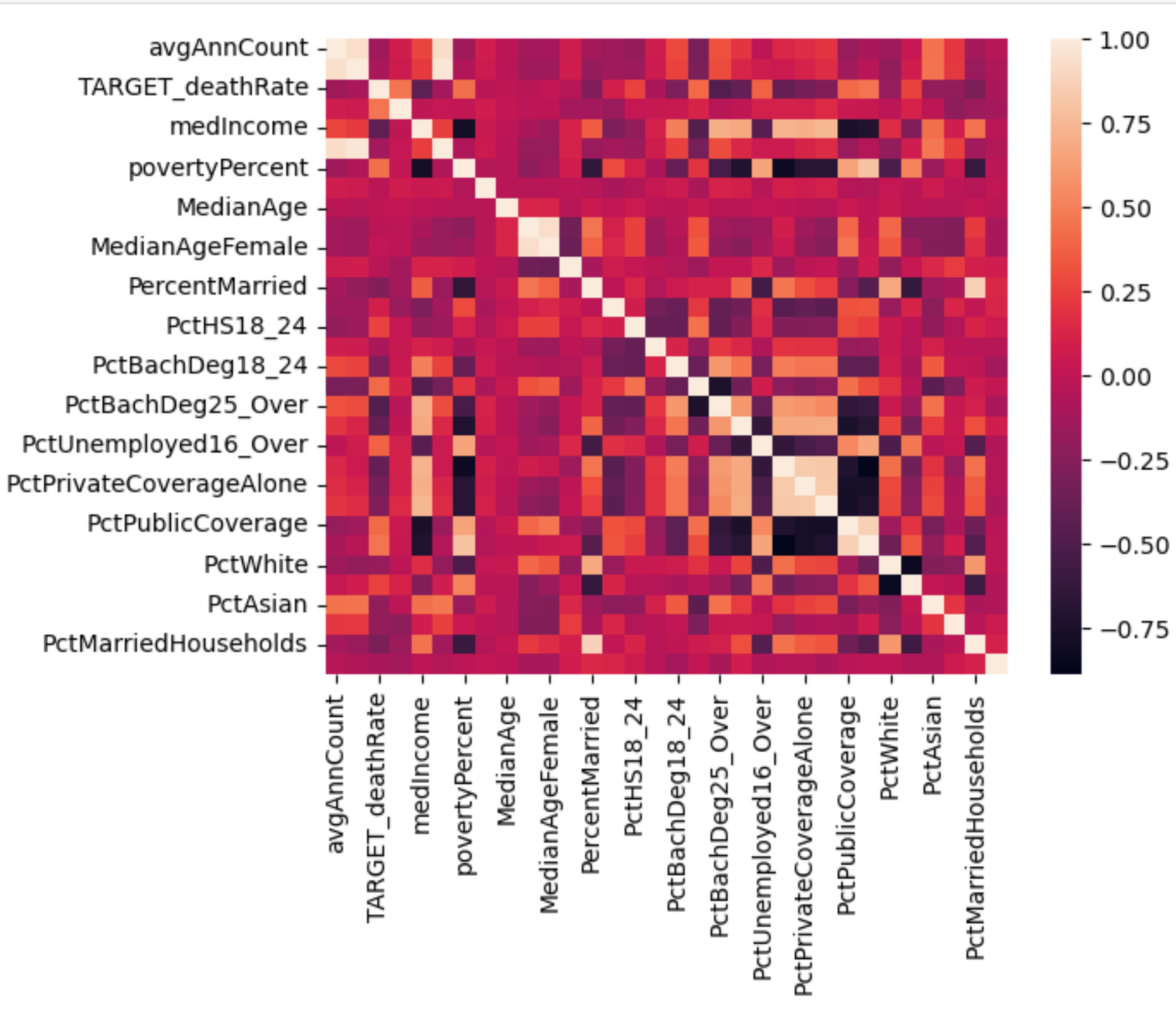
Out[30]: <AxesSubplot:>



Correlation plot There are some correlations between the features and the target

The top three features that have high correlation with the target_deathrate are: incidenceRate 0.449432 PctPublicCoverageAlone 0.449358 povertyPercent 0.429389

In [86]:
```
# correlation plot
#correlation heat map
corr = df.corr()
sns.heatmap(corr,annot= False);
```

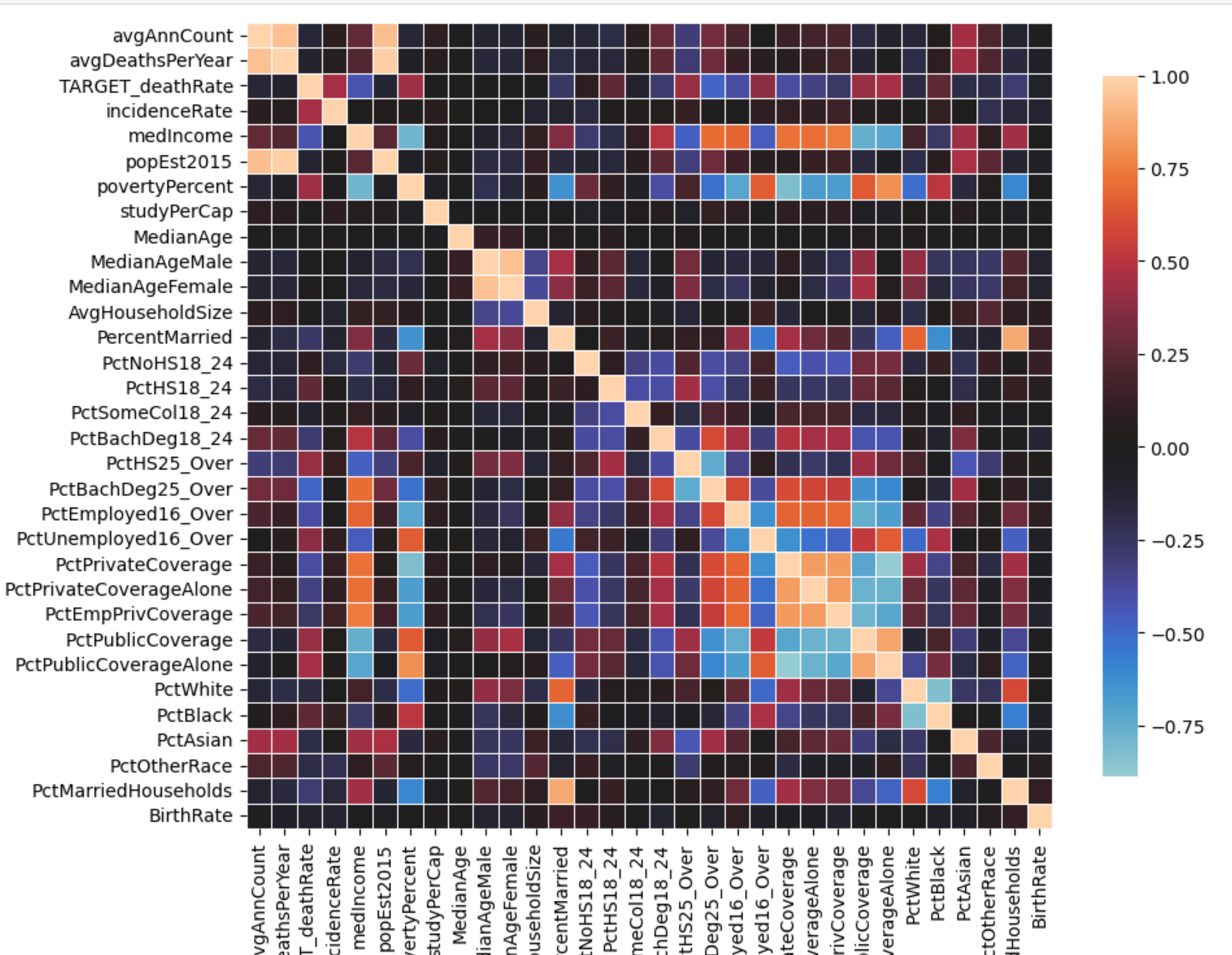


In [37]:
```
#correlation heat map

def correlation_heatmap(df):
    correlations = df.corr()

    fig, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(correlations, vmax=1.0, center=0, fmt='.2f',
                square=True, linewidths=.5, annot=False, cbar_kws={"shrink": .70})
    plt.show();

correlation_heatmap(df)
```



In [42]:
```
corr['TARGET_deathRate'].sort_values(ascending = False)
```

Out[42]:
```
TARGET_deathRate           1.000000
incidenceRate              0.449432
PctPublicCoverageAlone     0.449358
povertyPercent             0.429389
PctHS25_Over               0.404589
PctPublicCoverage          0.404572
PctUnemployed16_Over       0.378412
PctHS18_24                 0.261976
PctBlack                   0.257024
PctNoHS18_24               0.088463
MedianAgeFemale            0.012048
MedianAge                  0.004375
MedianAgeMale             -0.021929
studyPerCap               -0.022285
AvgHouseholdSize          -0.036905
BirthRate                 -0.087407
avgDeathsPerYear          -0.090715
PctSomeCol18_24           -0.094765
popEst2015                -0.120073
avgAnnCount               -0.143532
PctWhite                  -0.177400
PctAsian                  -0.186331
PctOtherRace              -0.189894
PercentMarried            -0.266820
PctEmpPrivCoverage        -0.267399
PctBachDeg18_24           -0.287817
PctMarriedHouseholds      -0.293325
PctPrivateCoverageAlone   -0.326067
PctPrivateCoverage        -0.386066
PctEmployed16_Over        -0.397487
medIncome                 -0.428615
PctBachDeg25_Over         -0.485477
Name: TARGET_deathRate, dtype: float64
```

Deal with Missing data

1. DROPPING
2. IMPUTATE (Missing values in continuous data can be solved by imputing with mean ,median ,mode or with multiple imputation)

Using mean values to fill the missing values in PctSomeCol18_24,PctEmployed16_Over,PctPrivateCoverageAlone

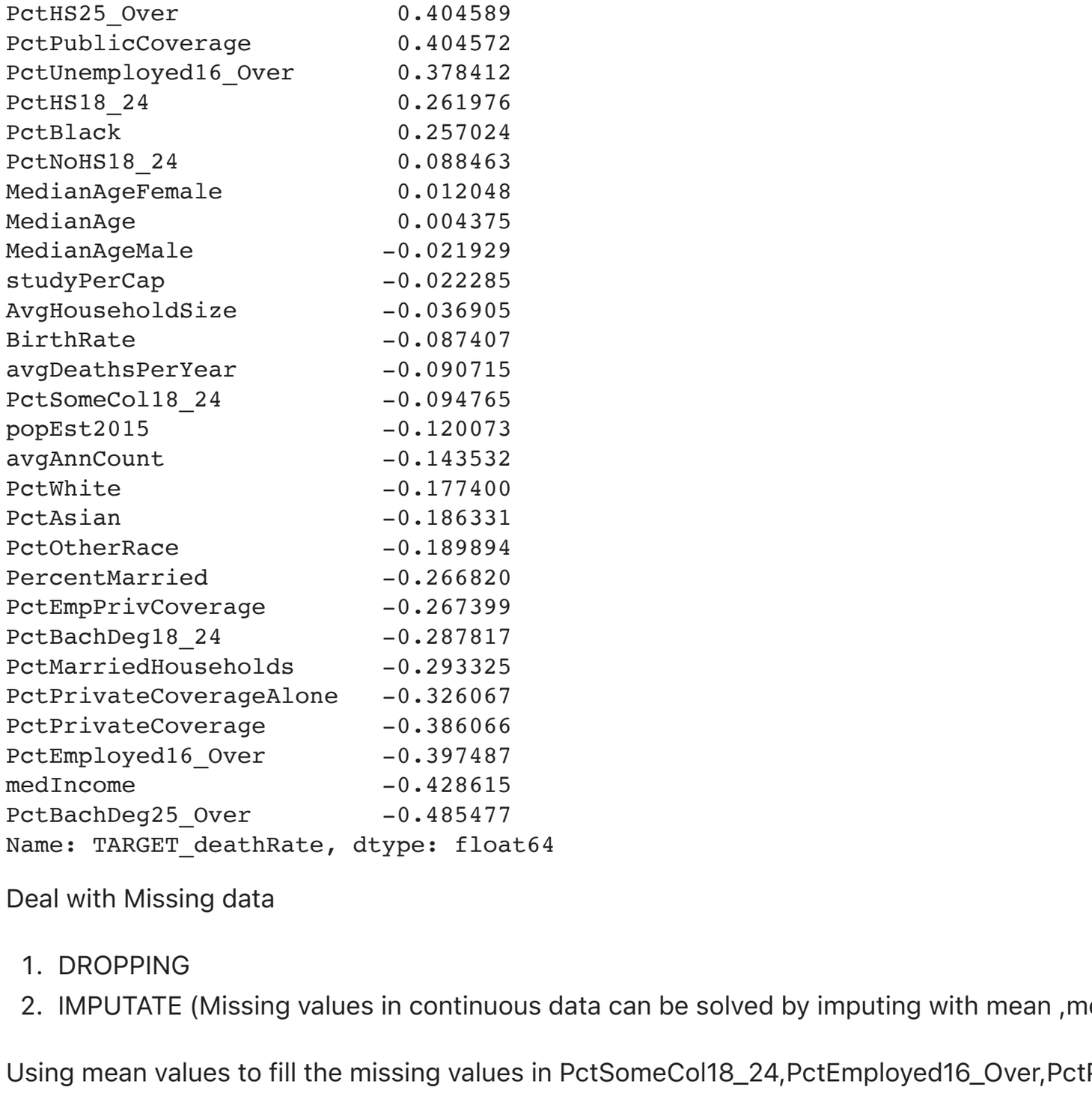
In [29]:
```
df['PctSomeCol18_24'].fillna(df['PctSomeCol18_24'].mean(),inplace=True)
df['PctEmployed16_Over'].fillna(df['PctEmployed16_Over'].mean(),inplace=True)
df['PctPrivateCoverageAlone'].fillna(df['PctPrivateCoverageAlone'].mean(),inplace=True)
```

Distribution of the target death rate

- Looks like a right skewed distribution

In [79]:
```
ax = sns.histplot(df['TARGET_deathRate'],bins=50,color='r', kde=False)
ax.set_title('Distribution of death rate')
```

Out[79]: <matplotlib.legend.Legend at 0x7f7988ce3400>



Splitting data into train and test data

In [87]:
```
from sklearn.model_selection import train_test_split
X = df[['incidenceRate','PctPublicCoverageAlone','povertyPercent']] # Independet variables
y = df['TARGET_deathRate'] # dependent variable

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=23)
```

In [114…
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
fit = lm.fit(X_train,y_train)
fit.summary
R_square_train= fit.score(X_train,y_train)
R_square_train
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/l5/3xc64rz12wd5s3qlhgsnk0_h0000gn/T/ipykernel_18686/516146826.py in <module>
      2 lm = LinearRegression()
      3 fit = lm.fit(X_train,y_train)
----> 4 fit.summary
      5 R_square_train= fit.score(X_train,y_train)
      6 R_square_train

AttributeError: 'LinearRegression' object has no attribute 'summary'
```

The coefficients of the multiple linear model: Y = theta_0 + theta_1*incidenceRate + theta_2*PctPublicCoverageAlone+ theta_3*povertyPercent

where theta_0 = 37.74 theta_1 = 0.227 theta_2 = 1.299 theta_3 = 0.819

In [103…
```
sk_theta = [fit.intercept_]+list(fit.coef_)
sk_theta
```

Out[103]: [37.74007811128746, 0.22746974883722773, 1.299049487155849, 0.819555452322212]

In [104…
```
parameter = ['theta_'+str(i) for i in range(1,X_train.shape[1]+1)]
parameter = ['theta_0'] + list(parameter)
columns = ['intersect:x_0=1'] + list(X.columns.values)
parameter_df = pd.DataFrame({'Parameter':parameter,'Columns':columns,'theta':sk_theta})
parameter_df
```

Out[104]:

	Parameter	Columns	theta
0	theta_0	intersect:x_0=1	37.740078
1	theta_1	incidenceRate	0.227470
2	theta_2	PctPublicCoverageAlone	1.299049
3	theta_3	povertyPercent	0.819555

Evaluation of the model

- Huge difference between actual and predicted values. (Maybe these features can't explain the output clearly
- R^2 of the model on test data is 37%. Maybe fit another model?
- RMSE is quite high. More

In [110…
```
# predicting output from the test data
y_pred = fit.predict(X_test)
```

In [109…
```
# data frame comparing the results from predicted y and actual y in test set
predict_df = pd.DataFrame({'Actual' : y_test, 'Predicted': y_pred})
predict_df['Diff'] = predict_df['Actual'] - predict_df['Predicted']
predict_df.reset_index(drop = True, inplace = True)
```

Out[109]:

	Actual	Predicted	Diff
0	217.6	202.010057	15.589943
1	205.3	162.525826	42.774174
2	172.7	175.394497	-2.694497
3	158.2	178.758279	-20.558279
4	193.5	210.153606	-16.653606
...	...	...	...
910	153.9	205.044213	-51.144213
911	173.1	176.518702	-3.418702
912	195.3	174.932282	20.367718
913	155.4	159.788084	-4.388084
914	193.2	180.554598	12.645402

915 rows × 3 columns

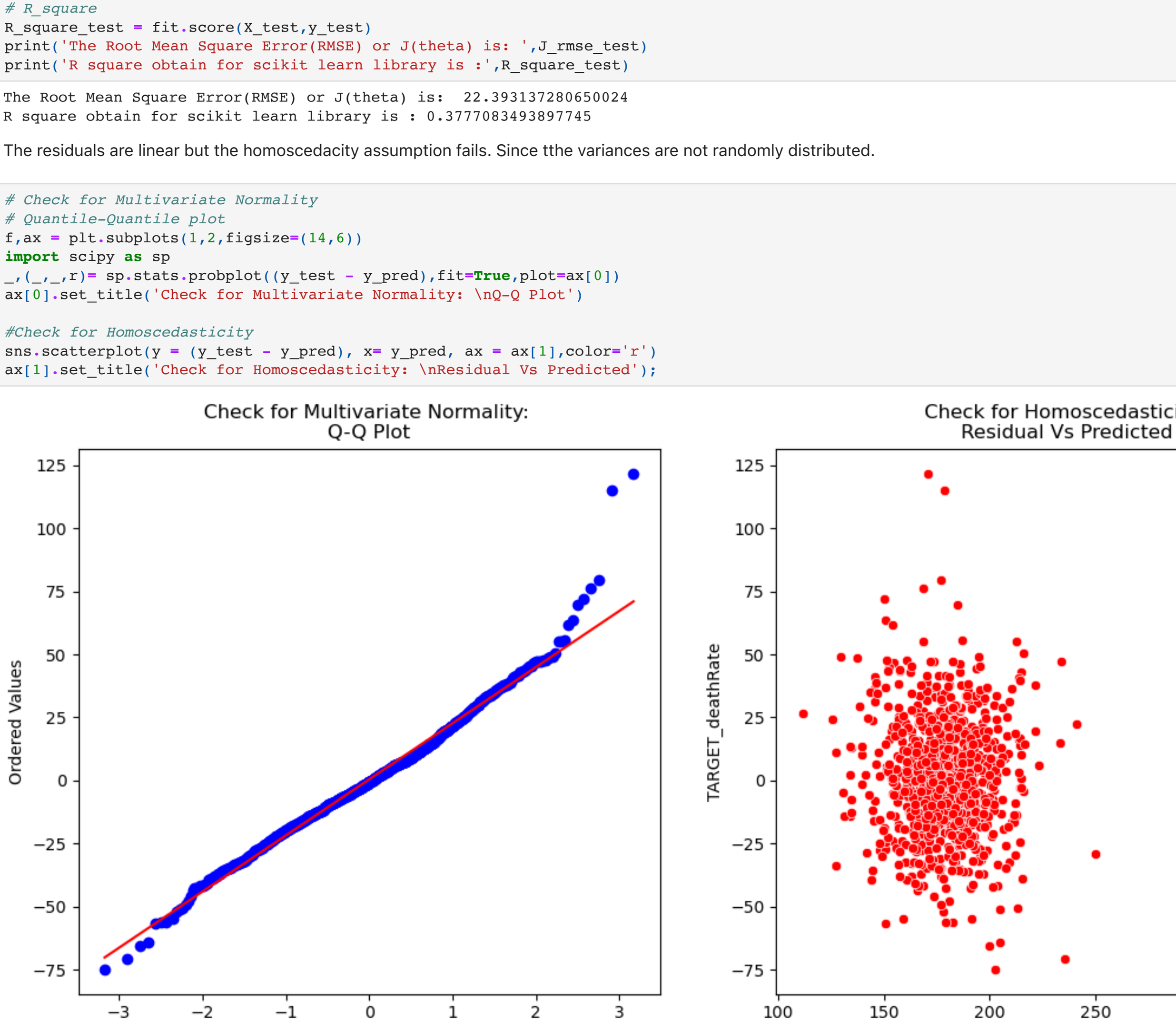
In [116…
```
#Evaluvation: MSE
from sklearn.metrics import mean_squared_error
J_mse_test = mean_squared_error(y_pred, y_test)
J_rmse_test = np.sqrt(J_mse_test)
# R_square
R_square_test = fit.score(X_test,y_test)
print('The Root Mean Square Error(RMSE) or J(theta) is: ',J_rmse_test)
print('R square obtain for scikit learn library is :',R_square_test)
```

```
The Root Mean Square Error(RMSE) or J(theta) is:  22.393137280650024
R square obtain for scikit learn library is : 0.3777083493897745
```

The residuals are linear but the homoscedacity assumption fails. Since tthe variances are not randomly distributed.

In [121…
```
# Check for Multivariate Normality
# Quantile-Quantile plot
f,ax = plt.subplots(1,2,figsize=(14,6))
import scipy as sp
_,(_,_,r)= sp.stats.probplot((y_test - y_pred),fit=True,plot=ax[0])
ax[0].set_title('Check for Multivariate Normality: \nQ-Q Plot')

#Check for Homoscedasticity
sns.scatterplot(y = (y_test - y_pred), x= y_pred, ax = ax[1],color='r')
ax[1].set_title('Check for Homoscedasticity: \nResidual Vs Predicted');
```

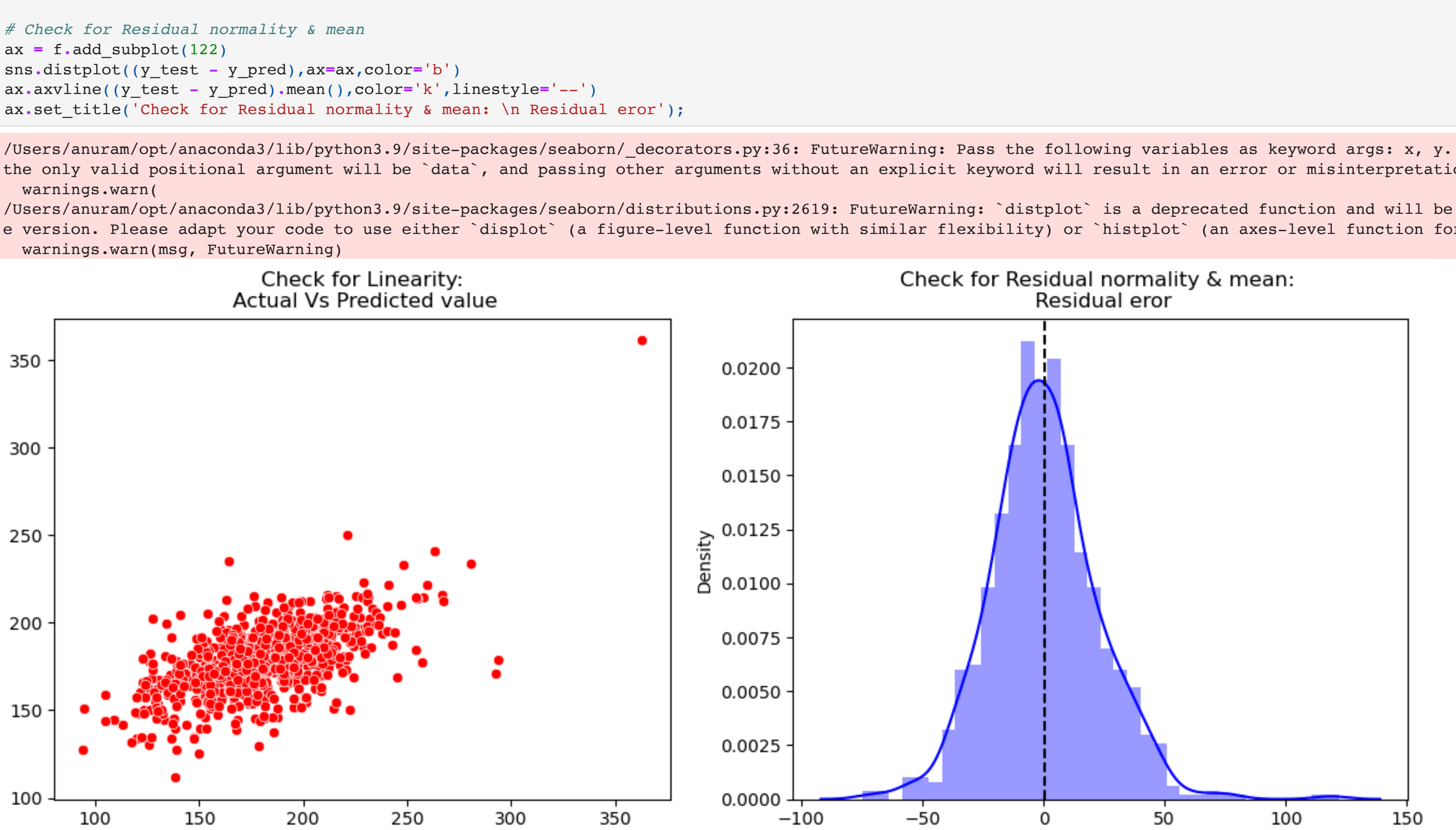


The data is not linear, it has a cluster of points and possibly an outlier point in the data around (350).

In [122…
```
# Check for Linearity
f = plt.figure(figsize=(14,5))
ax = f.add_subplot(121)
sns.scatterplot(y_test,y_pred,ax=ax,color='r')
ax.set_title('Check for Linearity:\n Actual Vs Predicted value')

# Check for Residual normality & mean
ax = f.add_subplot(122)
sns.distplot((y_test - y_pred),ax=ax,color='b')
ax.axvline((y_test - y_pred).mean(),color='k',linestyle='--')
ax.set_title('Check for Residual normality & mean: \n Residual eror');
```

```
/Users/anuram/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
/Users/anuram/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [123…
```
# Check for Multicollinearity
#Variance Inflation Factor
VIF = 1/(1- R_square_test)
VIF
```

Out[123]: 1.606963550000052

In [ ]: