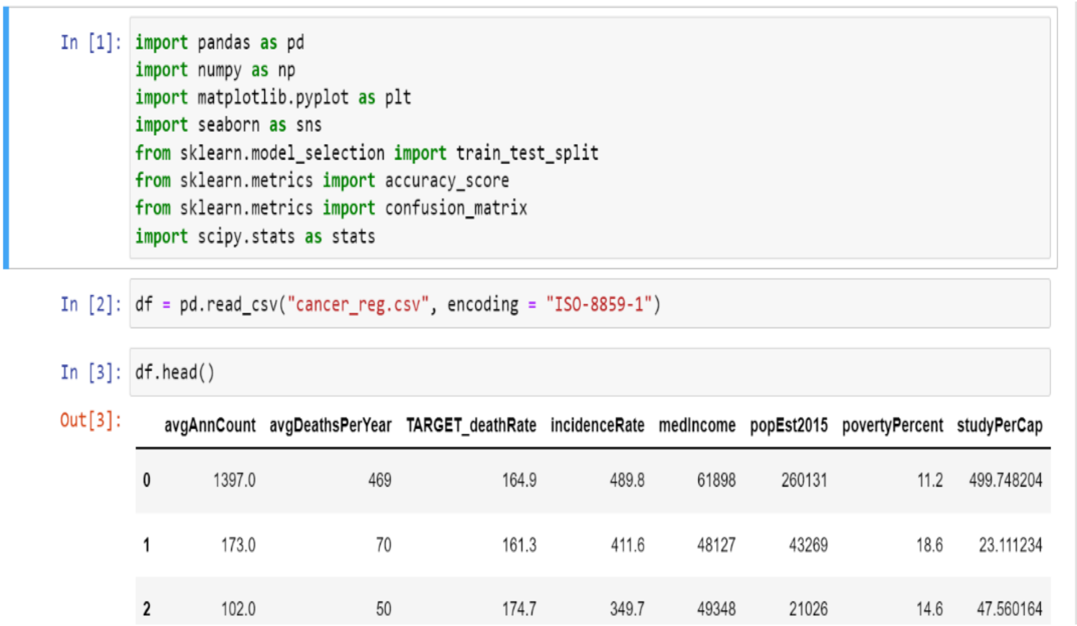
Documentation Flow

Project - 3 {Regression Modelling Exercise}

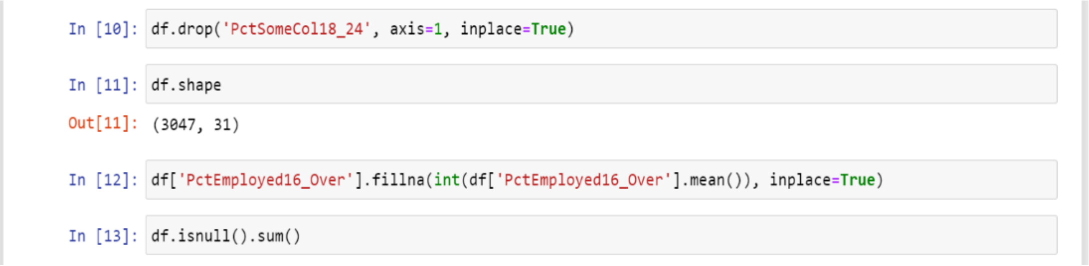
1. **Problem Statement->** We are provided with real time regression dataset and we have to perform EDA on that to find out and compare various machine learning algorithms and find which has the better rms value and accuracy and perform graphical visualization for better understanding of the data.

2. **EDA->** After the data analysis part i.e. importing the dataset and removing null values. We used three machine learning algorithms and did comparative study namely- Logistic regression, Random Forest and XGBoost. We found out the accuracy, root mean squared error and mean squared error for each of them.

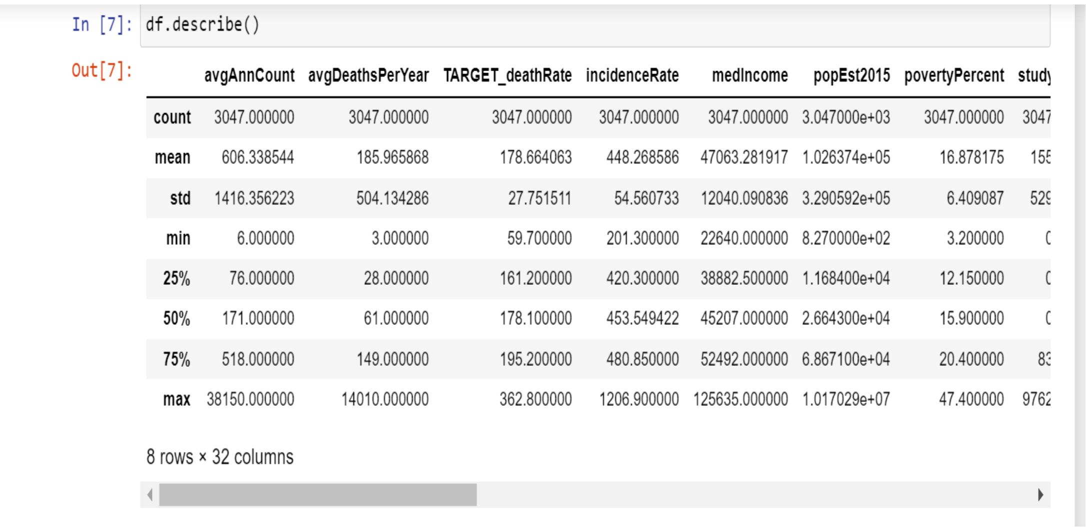


```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import confusion_matrix
        import scipy.stats as stats

In [2]: df = pd.read_csv("cancer_reg.csv", encoding = "ISO-8859-1")

In [3]: df.head()

Out[3]:
```

	avgAnnCount	avgDeathsPerYear	TARGET_deathRate	incidenceRate	medIncome	popEst2015	povertyPercent	studyPerCap
0	1397.0	469	164.9	489.8	61898	260131	11.2	499.748204
1	173.0	70	161.3	411.6	48127	43269	18.6	23.111234
2	102.0	50	174.7	349.7	49348	21026	14.6	47.560164



```
In [10]: df.drop('PctSomeCol18_24', axis=1, inplace=True)

In [11]: df.shape

Out[11]: (3047, 31)

In [12]: df['PctEmployed16_Over'].fillna(int(df['PctEmployed16_Over'].mean()), inplace=True)

In [13]: df.isnull().sum()
```

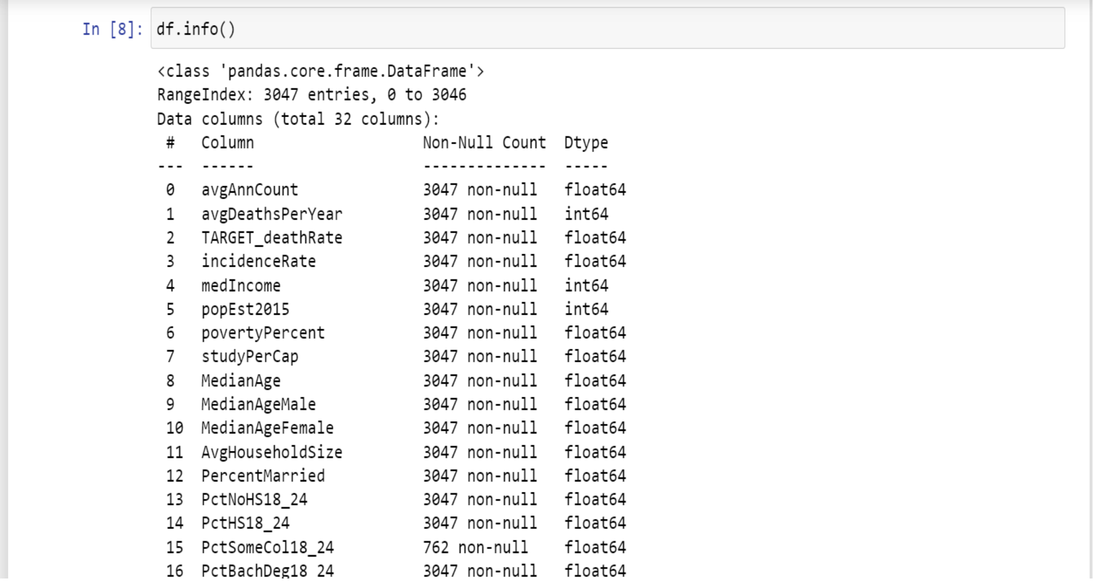


In [7]: df.describe()

Out[7]:

	avgAnnCount	avgDeathsPerYear	TARGET_deathRate	incidenceRate	medIncome	popEst2015	povertyPercent	study
count	3047.000000	3047.000000	3047.000000	3047.000000	3047.000000	3.047000e+03	3047.000000	3047
mean	606.338544	185.965868	178.664063	448.268586	47063.281917	1.026374e+05	16.878175	155
std	1416.356223	504.134286	27.751511	54.560733	12040.090836	3.290592e+05	6.409087	529
min	6.000000	3.000000	59.700000	201.300000	22640.000000	8.270000e+02	3.200000	0
25%	76.000000	28.000000	161.200000	420.300000	38882.500000	1.168400e+04	12.150000	0
50%	171.000000	61.000000	178.100000	453.549422	45207.000000	2.664300e+04	15.900000	0
75%	518.000000	149.000000	195.200000	480.850000	52492.000000	6.867100e+04	20.400000	83
max	38150.000000	14010.000000	362.800000	1206.900000	125635.000000	1.017029e+07	47.400000	9762

8 rows × 32 columns

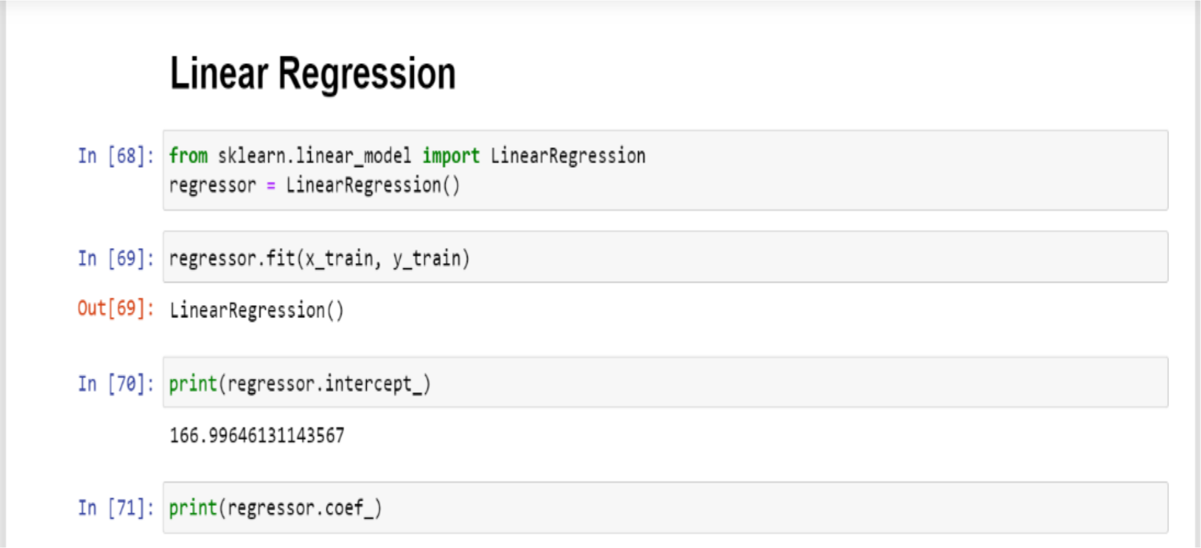


In [8]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3047 entries, 0 to 3046
Data columns (total 32 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   MedianAge                3047 non-null   float64
 9   MedianAgeMale            3047 non-null   float64
 10  MedianAgeFemale          3047 non-null   float64
 11  AvgHouseholdSize         3047 non-null   float64
 12  PercentMarried           3047 non-null   float64
 13  PctNoHS18_24             3047 non-null   float64
 14  PctHS18_24               3047 non-null   float64
 15  PctSomeCol18_24          762 non-null    float64
 16  PctBachDeg18_24          3047 non-null   float64
```

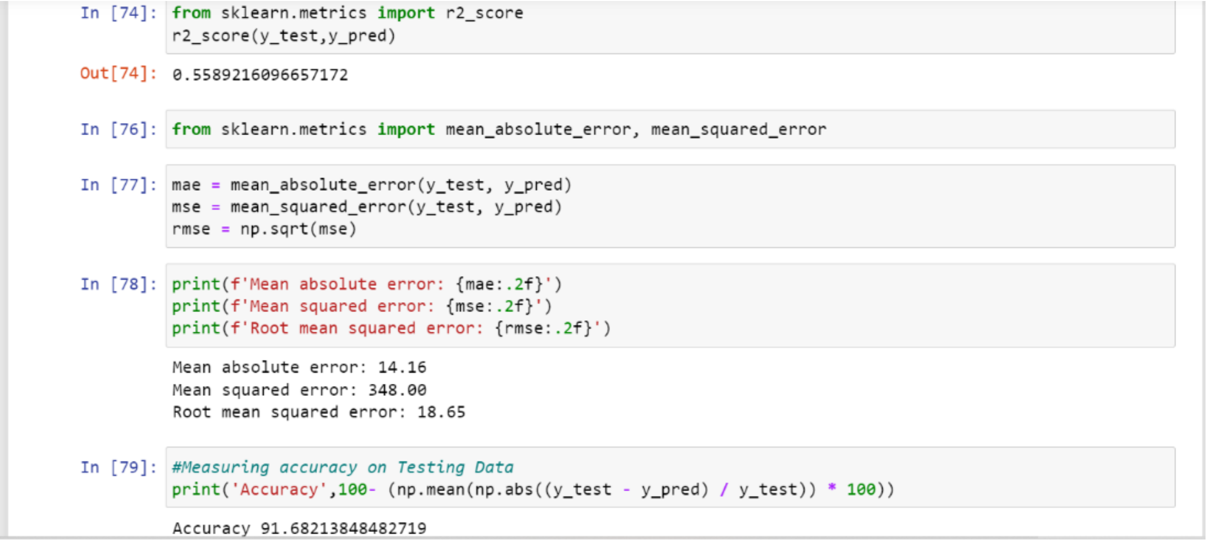
3. **REGRESSION MODELING->** A regression model provides a function that describes the relationship between one or more independent variables and a response, dependent, or target variable.

4. **COMPARATIVE STUDY->** We have used four machine learning algorithms for comparative study – Random Forest, XgBoost, Linear regression where we have observed that the root mean squared error value for Linear regression is the least and highest for XgBoost and on calculating the accuracy for each of them, we found that the linear regression was 91.68% accurate, whereas random forest was 88.51% accurate and XgBoost was 91.84% accurate which shows that XgBoost is the most accurate model.

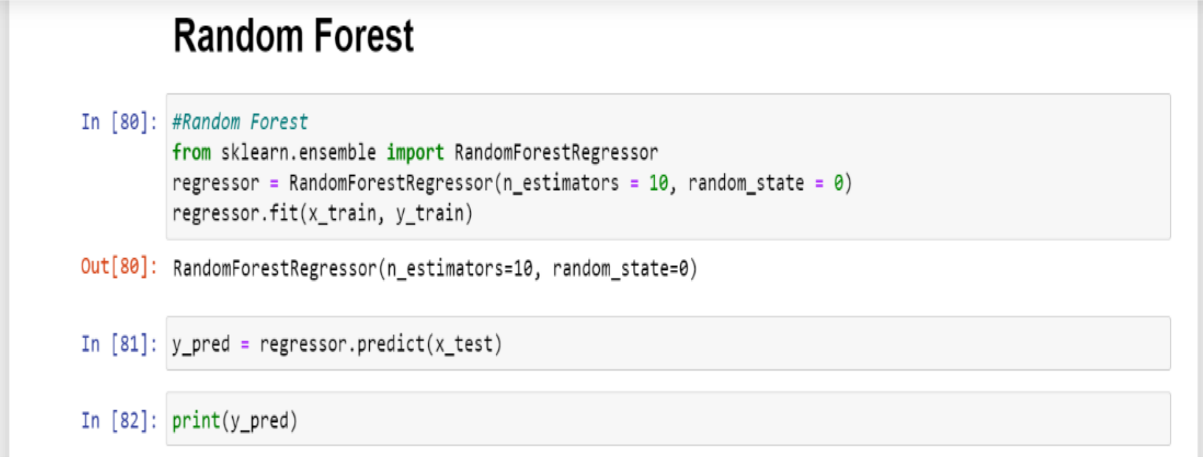


Linear Regression

```
In [68]: from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()

In [69]: regressor.fit(x_train, y_train)

Out[69]: LinearRegression()

In [70]: print(regressor.intercept_)

         166.99646131143567

In [71]: print(regressor.coef_)
```

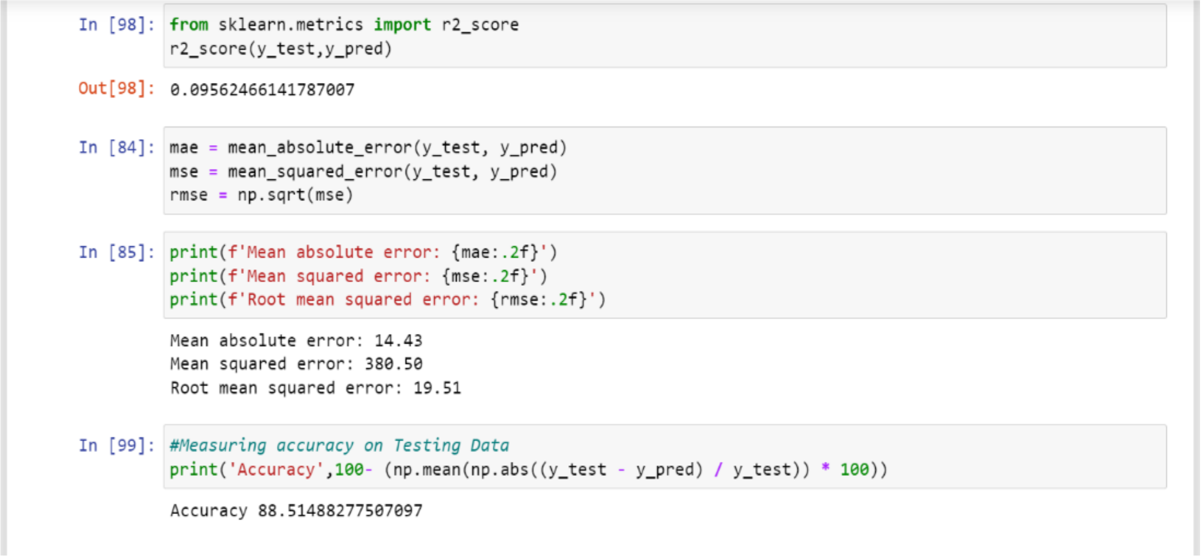


```
In [74]: from sklearn.metrics import r2_score
         r2_score(y_test,y_pred)

Out[74]: 0.5589216096657172

In [76]: from sklearn.metrics import mean_absolute_error, mean_squared_error

In [77]: mae = mean_absolute_error(y_test, y_pred)
         mse = mean_squared_error(y_test, y_pred)
         rmse = np.sqrt(mse)

In [78]: print(f'Mean absolute error: {mae:.2f}')
         print(f'Mean squared error: {mse:.2f}')
         print(f'Root mean squared error: {rmse:.2f}')

         Mean absolute error: 14.16
         Mean squared error: 348.00
         Root mean squared error: 18.65

In [79]: #Measuring accuracy on Testing Data
         print('Accuracy',100- (np.mean(np.abs((y_test - y_pred) / y_test)) * 100))

         Accuracy 91.68213848482719
```

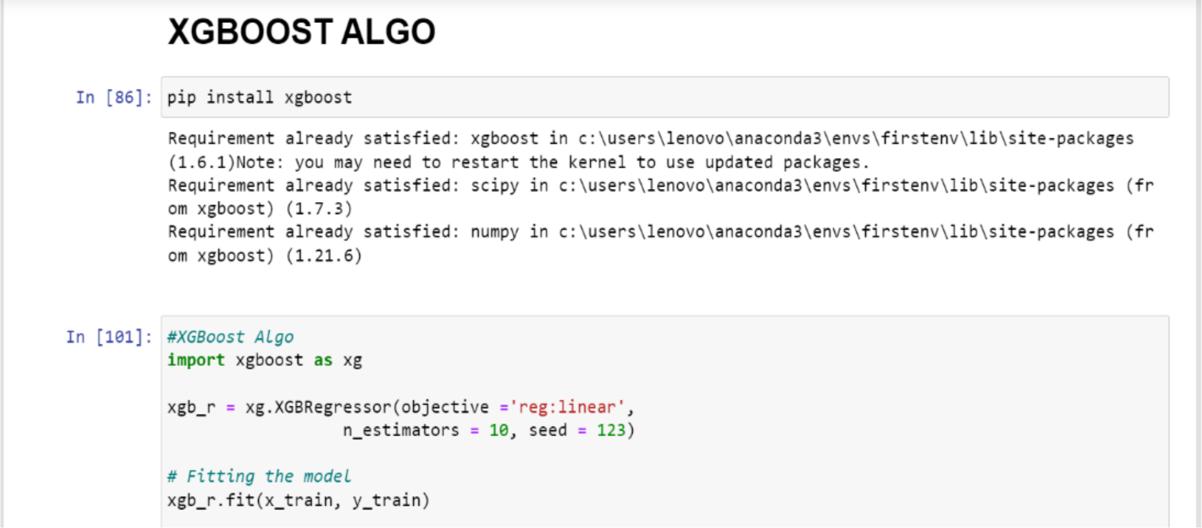


Random Forest

```
In [80]: #Random Forest
         from sklearn.ensemble import RandomForestRegressor
         regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
         regressor.fit(x_train, y_train)

Out[80]: RandomForestRegressor(n_estimators=10, random_state=0)

In [81]: y_pred = regressor.predict(x_test)

In [82]: print(y_pred)
```

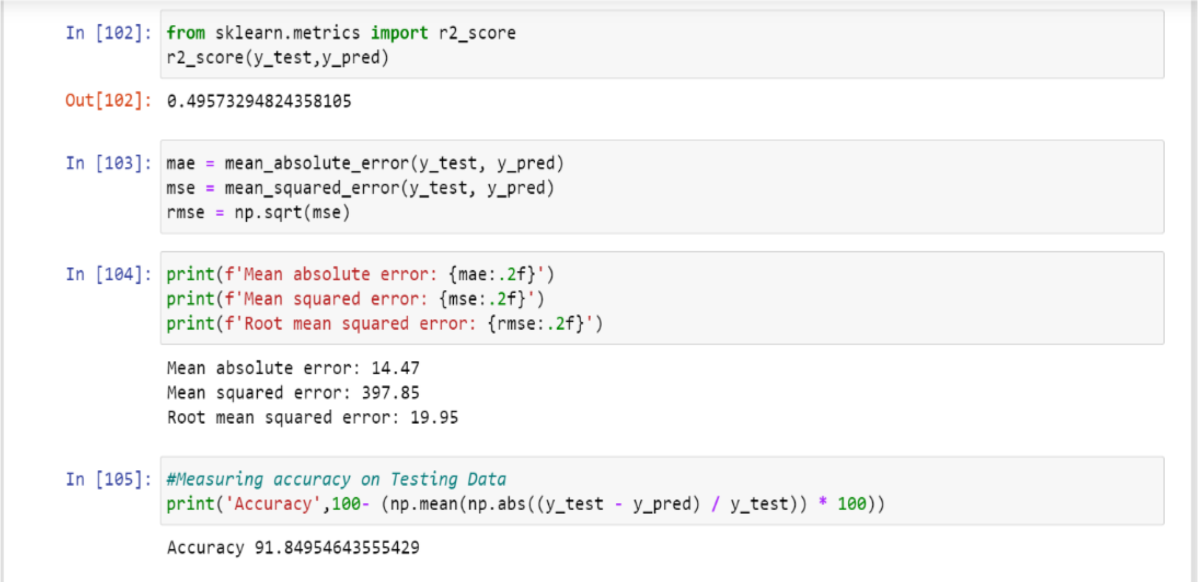


```
In [98]: from sklearn.metrics import r2_score
         r2_score(y_test,y_pred)

Out[98]: 0.09562466141787007

In [84]: mae = mean_absolute_error(y_test, y_pred)
         mse = mean_squared_error(y_test, y_pred)
         rmse = np.sqrt(mse)

In [85]: print(f'Mean absolute error: {mae:.2f}')
         print(f'Mean squared error: {mse:.2f}')
         print(f'Root mean squared error: {rmse:.2f}')

         Mean absolute error: 14.43
         Mean squared error: 380.50
         Root mean squared error: 19.51

In [99]: #Measuring accuracy on Testing Data
         print('Accuracy',100- (np.mean(np.abs((y_test - y_pred) / y_test)) * 100))

         Accuracy 88.51488277507097
```



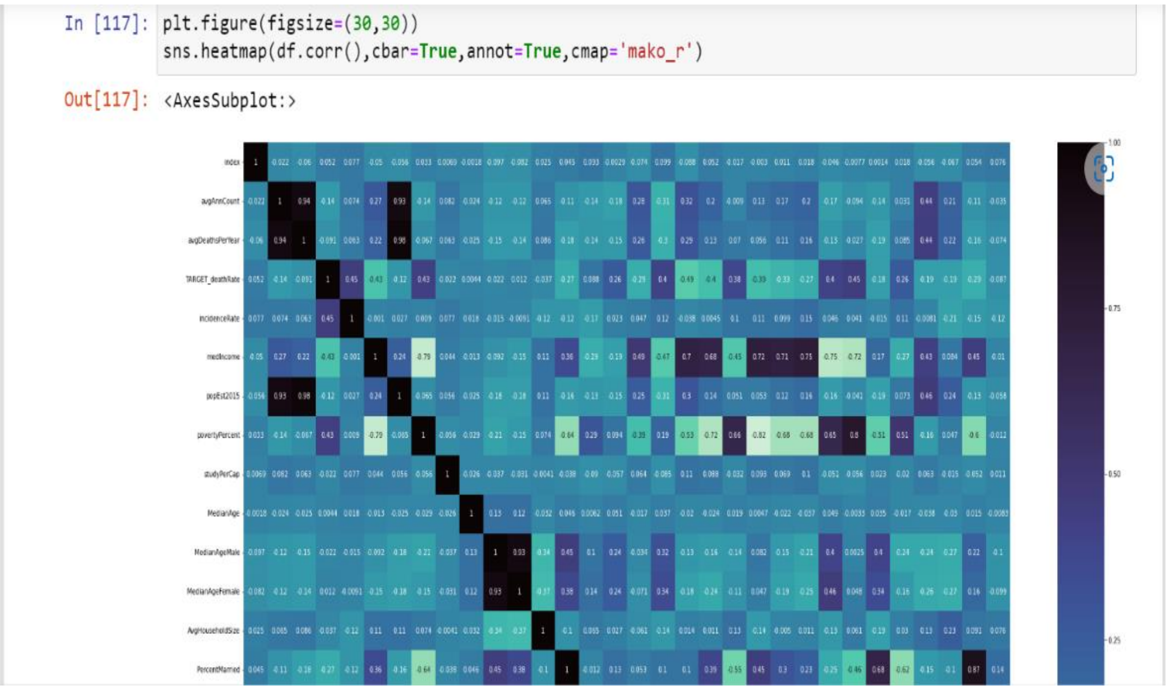
XGBOOST ALGO

```
In [86]: pip install xgboost

         Requirement already satisfied: xgboost in c:\users\lenovo\anaconda3\envs\firstenv\lib\site-packages
         (1.6.1)Note: you may need to restart the kernel to use updated packages.
         Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\envs\firstenv\lib\site-packages (fr
         om xgboost) (1.7.3)
         Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\envs\firstenv\lib\site-packages (fr
         om xgboost) (1.21.6)


In [101]: #XGBoost Algo
          import xgboost as xg

          xgb_r = xg.XGBRegressor(objective ='reg:linear',
                            n_estimators = 10, seed = 123)

          # Fitting the model
          xgb_r.fit(x_train, y_train)
```



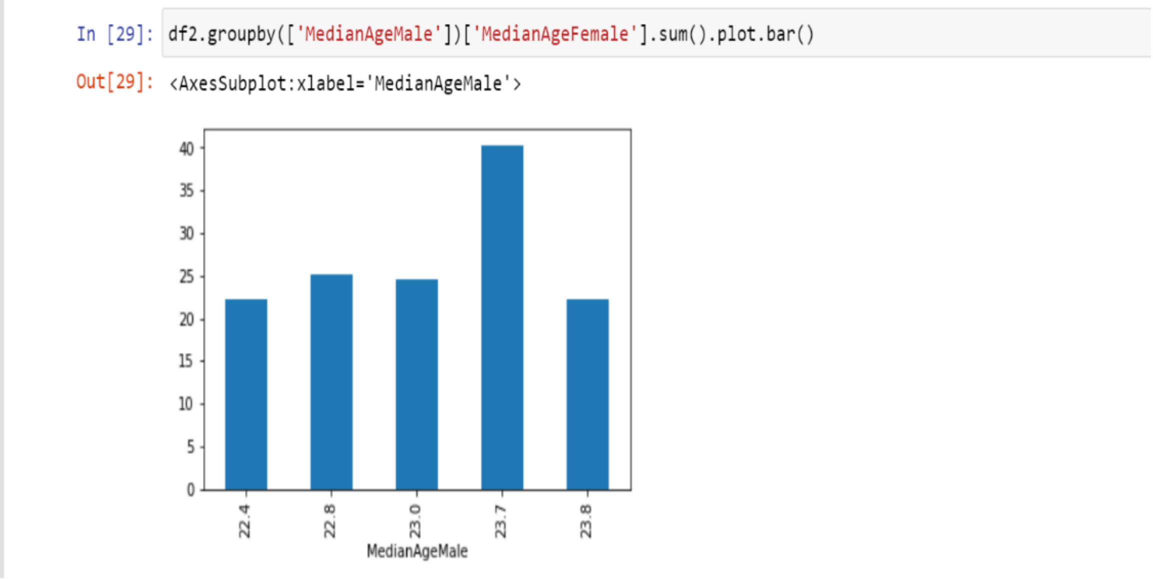
```
In [102]: from sklearn.metrics import r2_score
          r2_score(y_test,y_pred)

Out[102]: 0.49573294824358105

In [103]: mae = mean_absolute_error(y_test, y_pred)
          mse = mean_squared_error(y_test, y_pred)
          rmse = np.sqrt(mse)

In [104]: print(f'Mean absolute error: {mae:.2f}')
          print(f'Mean squared error: {mse:.2f}')
          print(f'Root mean squared error: {rmse:.2f}')

          Mean absolute error: 14.47
          Mean squared error: 397.85
          Root mean squared error: 19.95

In [105]: #Measuring accuracy on Testing Data
          print('Accuracy',100- (np.mean(np.abs((y_test - y_pred) / y_test)) * 100))

          Accuracy 91.84954643555429
```

5. **INFERENCE->** The graphical visualization of the dataset and their inference.



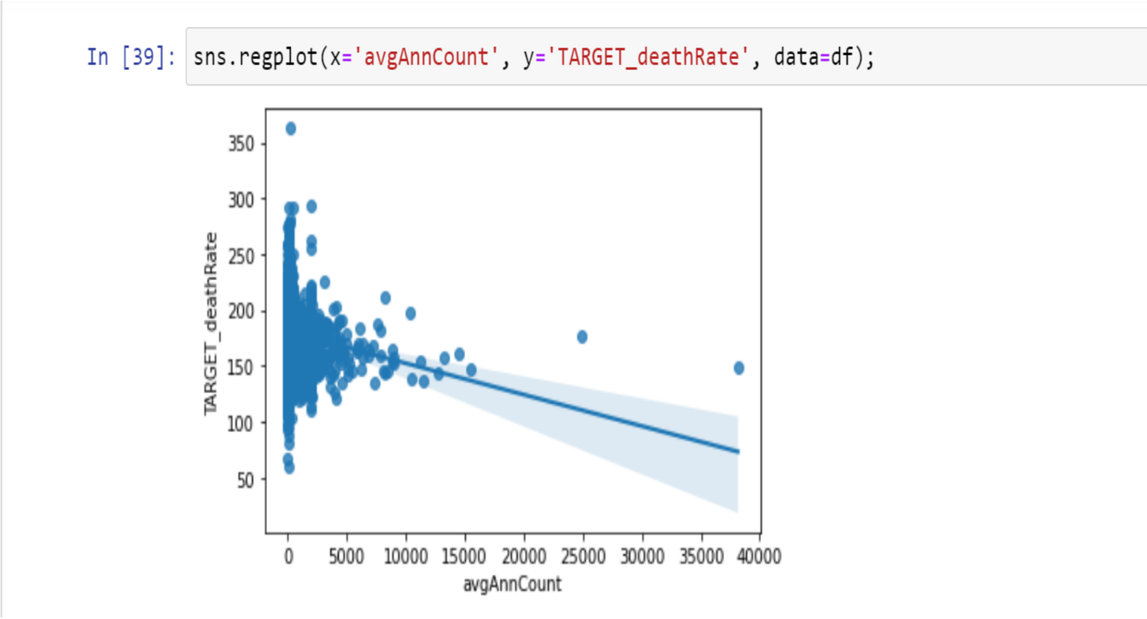
```
In [117]: plt.figure(figsize=(30,30))
          sns.heatmap(df.corr(),cbar=True,annot=True,cmap='mako_r')

Out[117]: <AxesSubplot:>
```

Inference-> This graph shows the correlation values between the variables**.**



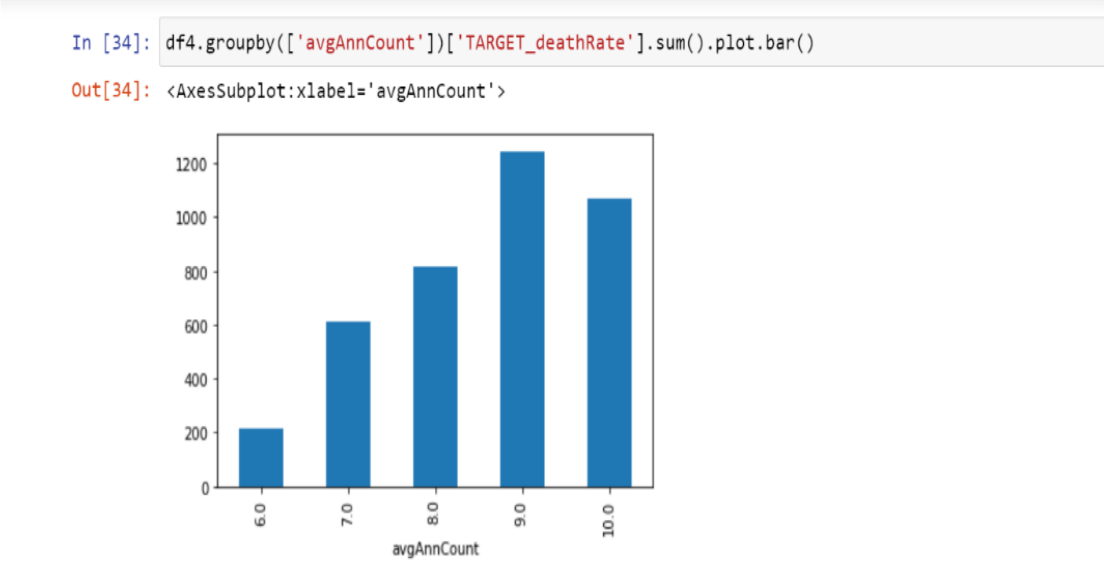
```
In [29]: df2.groupby(['MedianAgeMale'])['MedianAgeFemale'].sum().plot.bar()

Out[29]: <AxesSubplot:xlabel='MedianAgeMale'>
```

Inference-> This graph shows the relation between the MedianAgeMale and MedianAgeFemale



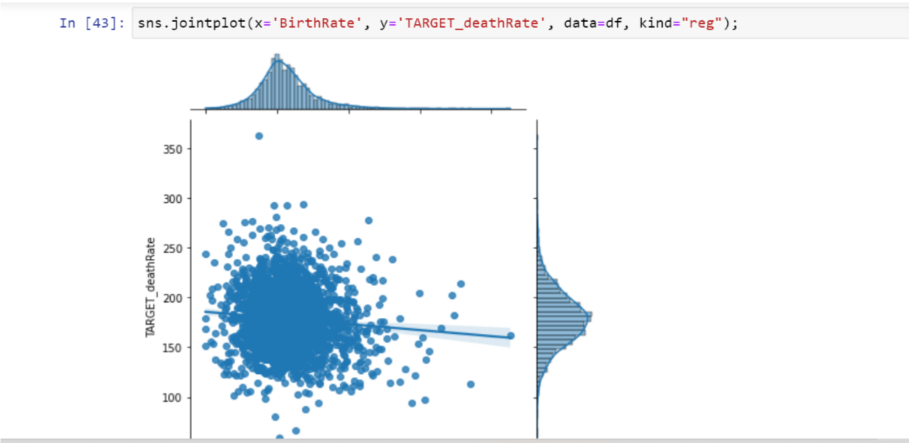
```
In [39]: sns.regplot(x='avgAnnCount', y='TARGET_deathRate', data=df);
```

Inference-> This graph shows the best fit of the data (best fit line).



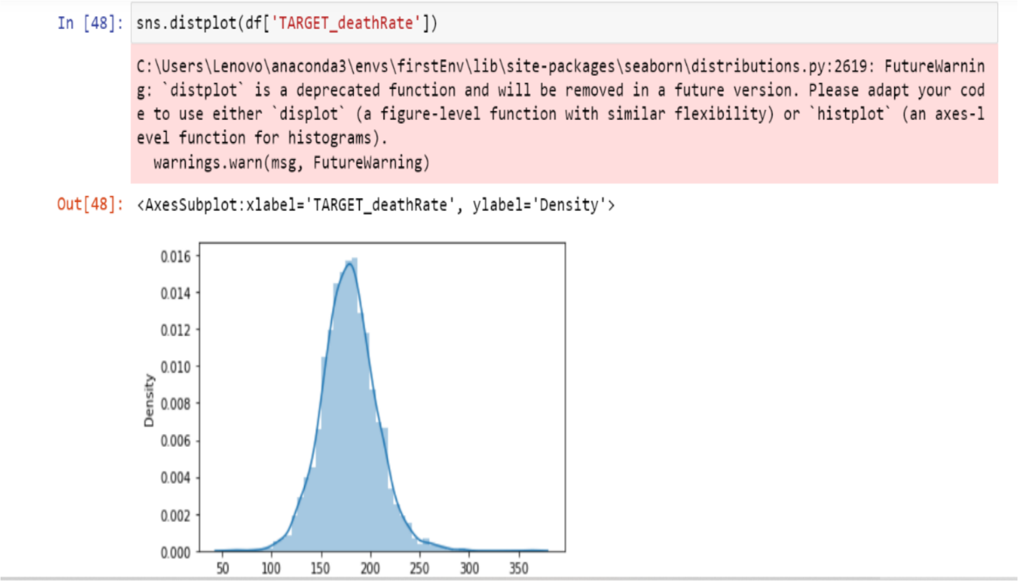
```
In [34]: df4.groupby(['avgAnnCount'])['TARGET_deathRate'].sum().plot.bar()

Out[34]: <AxesSubplot:xlabel='avgAnnCount'>
```

Inference-> This graph shows the relation between the average count and target death rate.



```
In [43]: sns.jointplot(x='BirthRate', y='TARGET_deathRate', data=df, kind="reg");
```

Inference-> This graph shows the relation between the x and y i.e., how the dependent variable(y) varies with the independent variable(x).



```
In [48]: sns.distplot(df['TARGET_deathRate'])

C:\Users\Lenovo\anaconda3\envs\firstEnv\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please adapt your cod
e to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-l
evel function for histograms).
  warnings.warn(msg, FutureWarning)

Out[48]: <AxesSubplot:xlabel='TARGET_deathRate', ylabel='Density'>
```

Inference-> This graph shows the distribution plot of target death rate and tells us where the target values fall in a distribution.