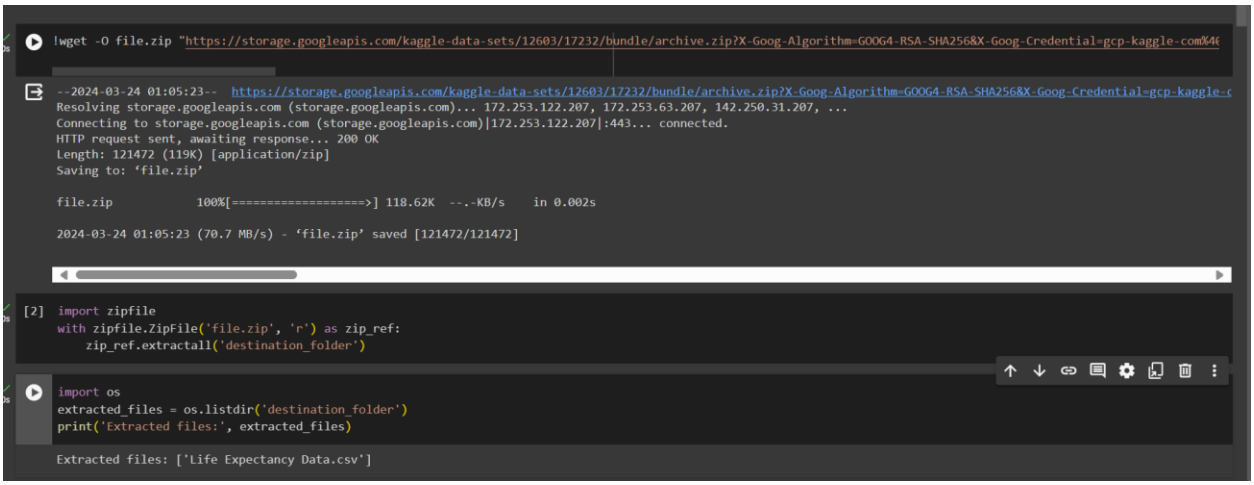
MOHAMED AFTHAB

C0891945

LIFE EXPECTANCY (WHO)

Importing Dataset from Kaggle



Importing necessary Libraries

Data Overview

```
life_xp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Country                          2938 non-null   object 
 1   Year                             2938 non-null   int64  
 2   Status                           2938 non-null   object 
 3   Life expectancy                  2928 non-null   float64
 4   Adult Mortality                  2928 non-null   float64
 5   infant deaths                    2938 non-null   int64  
 6   Alcohol                          2744 non-null   float64
 7   percentage expenditure           2938 non-null   float64
 8   Hepatitis B                      2385 non-null   float64
 9   Measles                          2938 non-null   int64  
 10   BMI                             2904 non-null   float64
 11  under-five deaths                2938 non-null   int64  
 12  Polio                            2919 non-null   float64
 13  Total expenditure                2712 non-null   float64
 14  Diphtheria                       2919 non-null   float64
 15   HIV/AIDS                        2938 non-null   float64
 16  GDP                              2490 non-null   float64
 17  Population                       2286 non-null   float64
 18   thinness  1-19 years            2904 non-null   float64
 19   thinness 5-9 years             2904 non-null   float64
 20  Income composition of resources  2771 non-null   float64
 21  Schooling                        2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
```

Descriptive Summary of each column like count, mean, standard deviation, min, max and percentiles

```
life_xp.describe()
```

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphthe
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.000000	2938.000000	2904.000000	2938.000000	2919.000000	2712.00000	2919.000
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.940461	2419.592240	38.321247	42.035739	82.550188	5.93819	82.324
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.070016	11467.272489	20.044034	160.445548	23.428046	2.49832	23.716
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.000000	0.000000	1.000000	0.000000	3.000000	0.37000	2.000
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.000000	0.000000	19.300000	0.000000	78.000000	4.26000	78.000
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.000000	17.000000	43.500000	4.000000	93.000000	5.75500	93.000
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.000000	360.250000	56.200000	28.000000	97.000000	7.49250	97.000
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.000000	212183.000000	87.300000	2500.000000	99.000000	17.60000	99.000

First few rows of the Dataset

```
life_xp.head()
```

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	...	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Popula
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	1154	...	6.0	8.16	65.0	0.1	584.259210	337364
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	492	...	58.0	8.18	62.0	0.1	612.696514	3275
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	430	...	62.0	8.13	64.0	0.1	631.744976	317316
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	2787	...	67.0	8.52	67.0	0.1	669.959000	36969
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	3013	...	68.0	7.87	68.0	0.1	63.537231	29785

5 rows × 22 columns

Cleaning trailing and leading white spaces in the column and displaying the count of unique values in each column

```
[9] life_xp.columns = life_xp.columns.str.strip()
```

```
for col in life_xp.columns:
    print(str(life_xp[col].value_counts()))
    print("----------------------------------------")
```

```
482.0        1
43.0         1
3978.0       1
12222251.0   1
Name: Population, Length: 2278, dtype: int64
----------------------------------------
1.0     74
1.9     65
0.8     64
0.7     63
1.2     62
        ..
16.5     1
16.7     1
16.9     1
17.1     1
15.8     1
Name: thinness  1-19 years, Length: 200, dtype: int64
----------------------------------------
0.9     69
1.1     67
0.5     63
1.9     63
1.0     62
        ..
16.9     1
17.2     1
17.4     1
17.6     1
27.9     1
```

DATA CLEANING

Finding both null and missing values and displaying its count

```
missing_values = life_xp.isnull().sum()
null_values = life_xp.isna().sum()

print("Missing values count for each column:")
print(missing_values)

print("Null value count for each column:")
print(null_values)
```

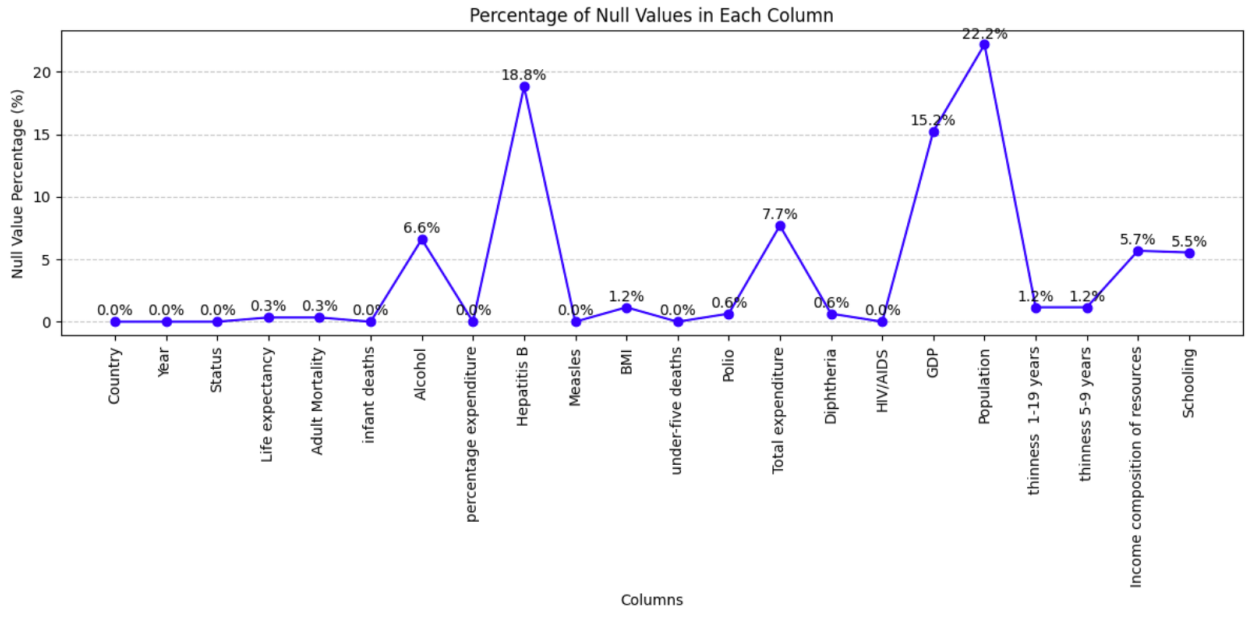
```
Missing values count for each column:
Country                             0
Year                                0
Status                              0
Life expectancy                    10
Adult Mortality                    10
infant deaths                       0
Alcohol                           194
percentage expenditure              0
Hepatitis B                       553
Measles                             0
BMI                                34
under-five deaths                   0
Polio                              19
Total expenditure                 226
Diphtheria                         19
HIV/AIDS                            0
GDP                               448
Population                        652
thinness  1-19 years               34
thinness 5-9 years                 34
Income composition of resources   167
Schooling                         163
dtype: int64
```

```
Null value count for each column:
Country                             0
Year                                0
Status                              0
Life expectancy                    10
Adult Mortality                    10
infant deaths                       0
Alcohol                           194
percentage expenditure              0
Hepatitis B                       553
Measles                             0
BMI                                34
under-five deaths                   0
Polio                              19
Total expenditure                 226
Diphtheria                         19
HIV/AIDS                            0
GDP                               448
Population                        652
thinness  1-19 years               34
thinness 5-9 years                 34
Income composition of resources   167
Schooling                         163
dtype: int64
```

Plotting a line graph with null and missing values percentage

```
null_percentages = life_xp.isnull().mean(axis=0) * 100

plt.figure(figsize=(12, 6))
plt.plot(null_percentages, marker='o', color='blue', linestyle='-')
plt.title('Percentage of Null Values in Each Column')
plt.xlabel('Columns')
plt.ylabel('Null Value Percentage (%)')
plt.xticks(rotation=90)
plt.grid(axis='y', linestyle='--', alpha=0.7)
for x, y in enumerate(null_percentages):
    plt.text(x, y + 0.5, f'{y:.1f}%', ha='center')
plt.tight_layout()
plt.show()
```

Output



Handling both null and missing values by replacing it with mean

```
[13] for col in life_xp.columns:
         if life_xp[col].isnull().any():
             life_xp[col] = life_xp[col].fillna(life_xp[col].mean())
```

```
missing_values = life_xp.isnull().sum()
null_values = life_xp.isna().sum()

print("Missing values count for each column:")
print(missing_values)

print("Null value count for each column:")
print(null_values)
```

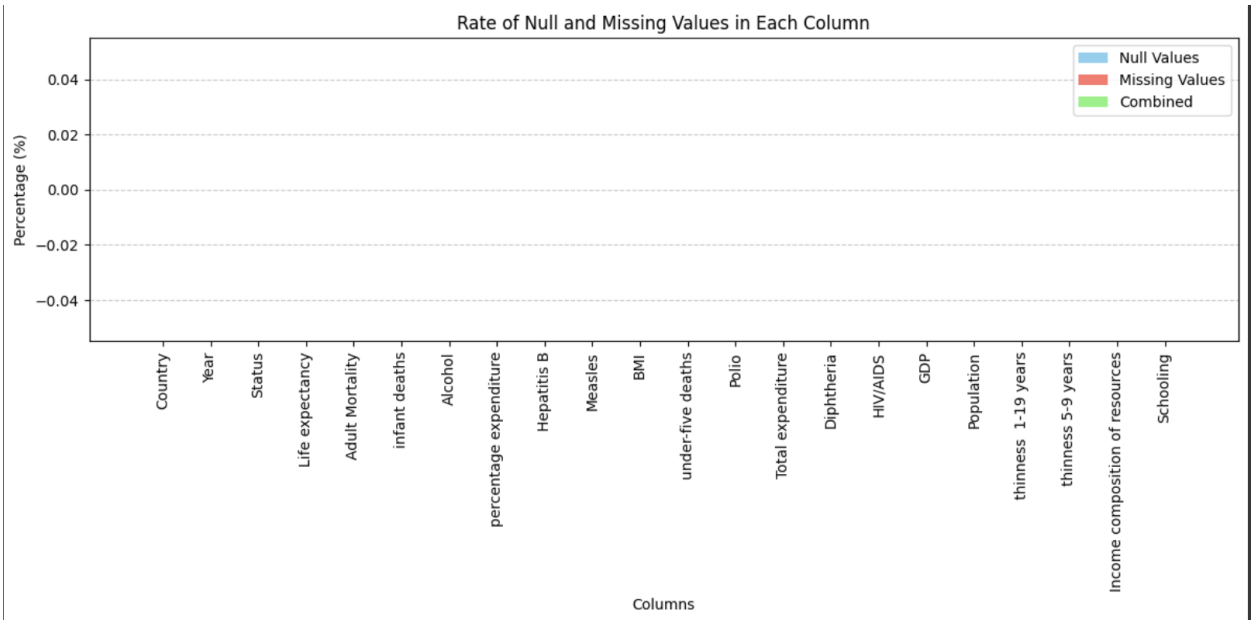
Now we don’t have any null and missing values

```
Missing values count for each column:
Country                            0
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
BMI                                0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
HIV/AIDS                           0
GDP                                0
Population                         0
thinness  1-19 years               0
thinness 5-9 years                 0
Income composition of resources    0
Schooling                          0
dtype: int64
Null value count for each column:
Country                            0
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
BMI                                0
under-five deaths                  0
Polio                              0
```

Plotting a graph to make sure that its removed

```python
import matplotlib.pyplot as plt
import pandas as pd

total_rows = len(life_xp)
null_percentages = (life_xp.isnull().sum() / total_rows) * 100
missing_percentages = (life_xp.isna().sum() / total_rows) * 100

combined_percentages = [null + missing for null, missing in zip(null_percentages, missing_percentages)]

plt.figure(figsize=(12, 6))
bar_width = 0.3
index = range(len(life_xp.columns))

plt.bar(index, null_percentages, width=bar_width, color='skyblue', label='Null Values')
plt.bar([i + bar_width for i in index], missing_percentages, width=bar_width, color='salmon', label='Missing Values')
plt.bar([i + 2*bar_width for i in index], combined_percentages, width=bar_width, color='lightgreen', label='Combined')

plt.title('Rate of Null and Missing Values in Each Column')
plt.xlabel('Columns')
plt.ylabel('Percentage (%)')
plt.xticks([i + bar_width for i in index], life_xp.columns, rotation=90)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Output



Checking for the duplicates

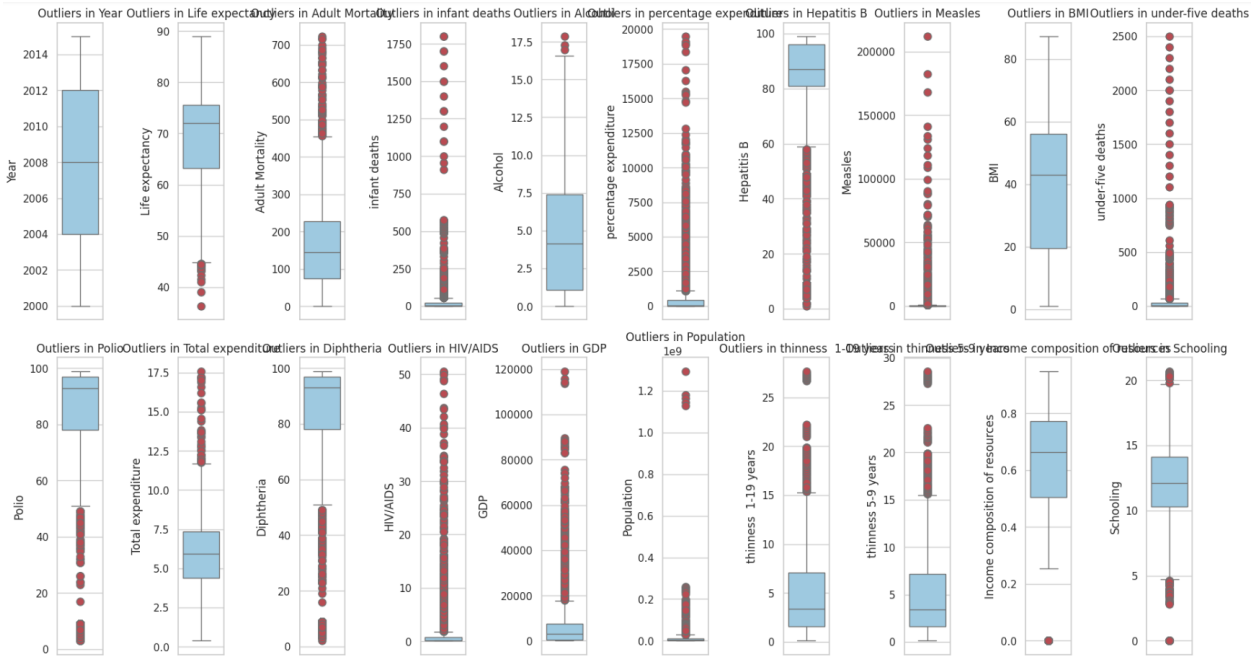
```
life_xp.duplicated().sum()
```

```
0
```

Plotting a graph for representing the outliers in each column

```
num_columns = life_xp.select_dtypes(include=np.number).columns

plt.figure(figsize=(18, 10))
sns.set(style="whitegrid")
for i, col in enumerate(num_columns, start=1):
    plt.subplot(2, int(np.ceil(len(num_columns) / 2)), i)
    sns.boxplot(y=life_xp[col], color='skyblue', flierprops=dict(marker='o', markerfacecolor='r', markersize=8))
    plt.title(f'Outliers in {col}')
    plt.ylabel(col)
plt.tight_layout()
plt.show()
```

output



Handling outliers by replacing it with median values instead of dropping it.

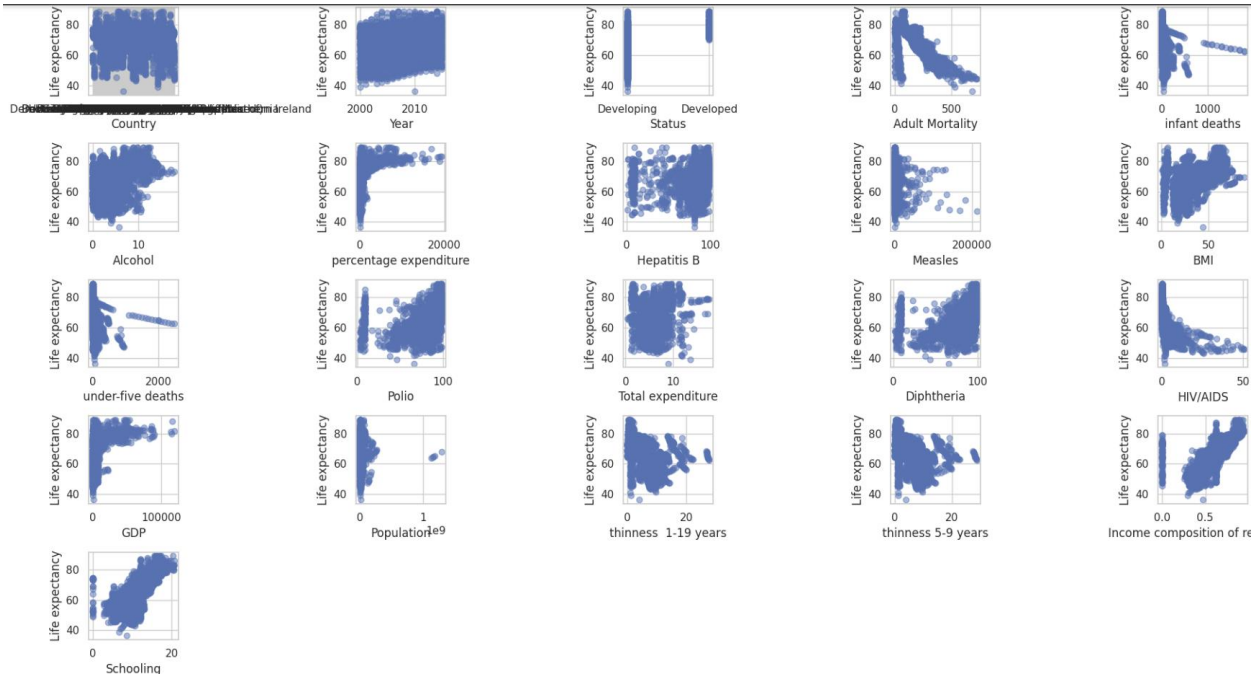
```
[ ]  def handle_outliers_with_iqr(data):
         # Calculate Q1 (25th percentile) and Q3 (75th percentile)
         Q1 = data.quantile(0.25)
         Q3 = data.quantile(0.75)
         # Calculate IQR (Interquartile Range)
         IQR = Q3 - Q1
         # Define the lower and upper bounds for outliers
         lower_bound = Q1 - 1.5 * IQR
         upper_bound = Q3 + 1.5 * IQR

         # Replace outliers with median
         for col in data.columns:
             outliers_mask = (data[col] < lower_bound[col]) | (data[col] > upper_bound[col])
             data.loc[outliers_mask, col] = data[col].median()

     # Handle outliers with IQR for numerical columns
     handle_outliers_with_iqr(life_xp[num_columns])
```

DATA VISUALIZATION

Scatter plot representing relationships between each column with the target variable Life expectancy

```
features = ['Country', 'Year', 'Status', 'Adult Mortality', 'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B', 'Measles', 'BMI', 'under-five deaths',
target = 'Life expectancy'

X = life_xp[features]
y = life_xp[target]

plt.figure(figsize=(18, 10))
for i, feature in enumerate(features, start=1):
    plt.subplot(5, 5, i)
    plt.scatter(X[feature], y, alpha=0.5)
    plt.xlabel(feature)
    plt.ylabel(target)
plt.tight_layout()
plt.show()
```

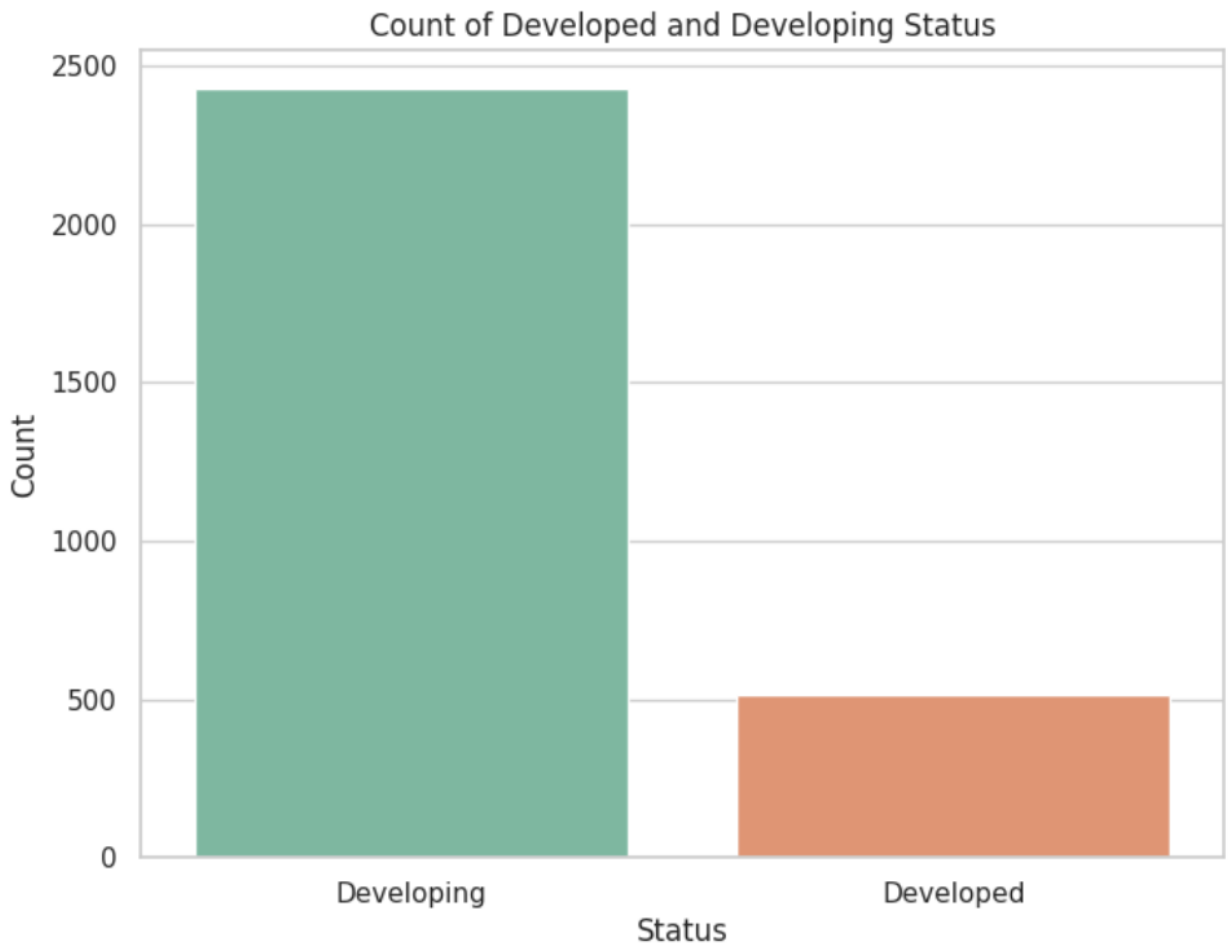
Output



Comparison between Developed and Developing country

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Status', data=life_xp, palette='Set2')
plt.title('Count of Developed and Developing Status')
plt.xlabel('Status')
plt.ylabel('Count')
plt.show()
```

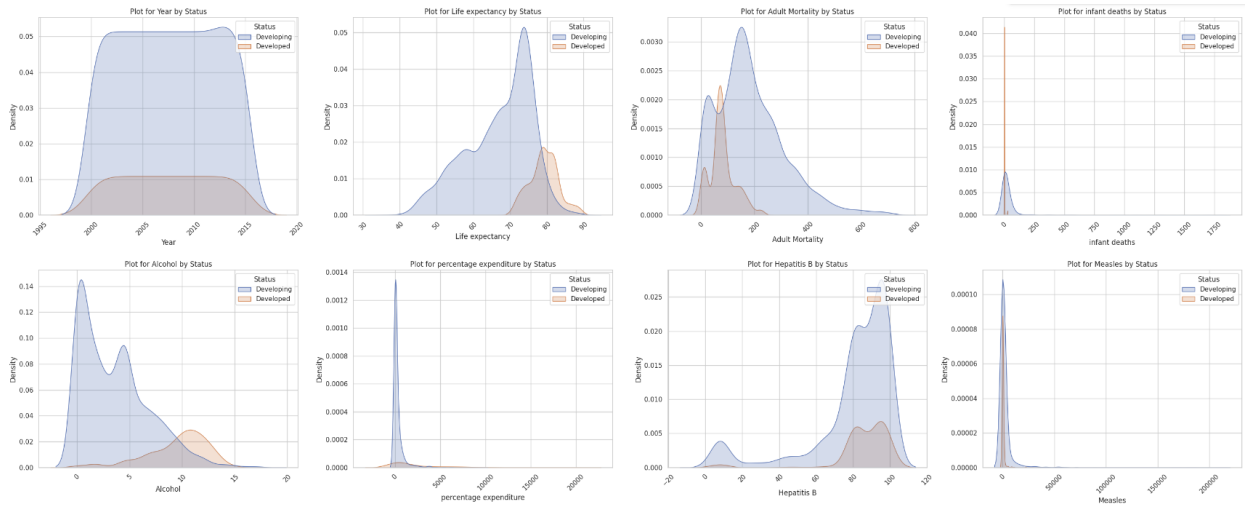
output

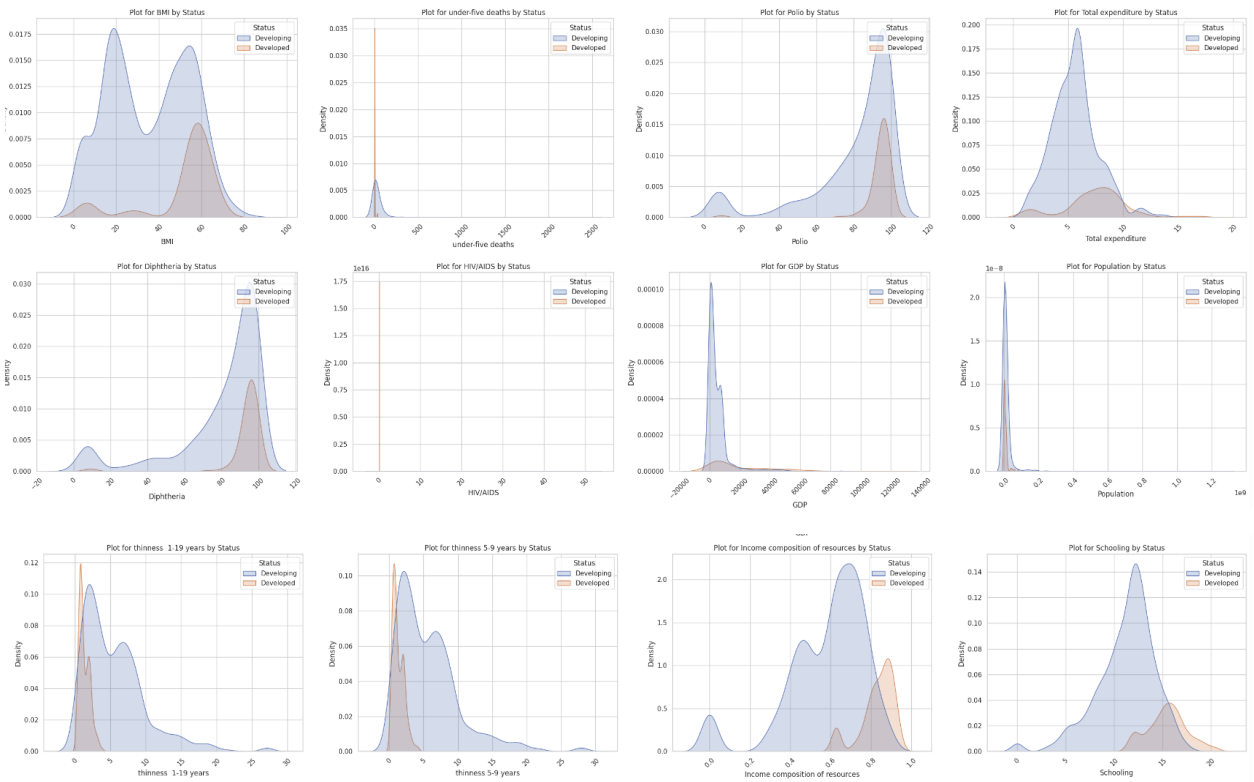


Graph representing relation of each column with target variable Life expectancy on the basis of both Developed and Developing Countries

```
plt.figure(figsize=(30,30))

counter = 0

for col in num_columns:
    plt.subplot(5,4,counter+1)
    g = sns.kdeplot(x=col, data=life_xp, fill=True, hue="Status")
    plt.xticks(rotation=45)
    plt.title(f'Plot for {col} by Status')
    plt.xlabel(col)
    plt.ylabel('Density')
    counter += 1

plt.tight_layout()
plt.show()
```

Output





Lineplot

```
plt.figure(figsize=(20, 20))
for i, feature in enumerate(features, start=1):
    plt.subplot(5, 5, i)
    sns.lineplot(x=feature, y='Life expectancy', data=life_xp)
    plt.xlabel(feature)
    plt.ylabel('Life expectancy')
plt.tight_layout()
plt.show()
```

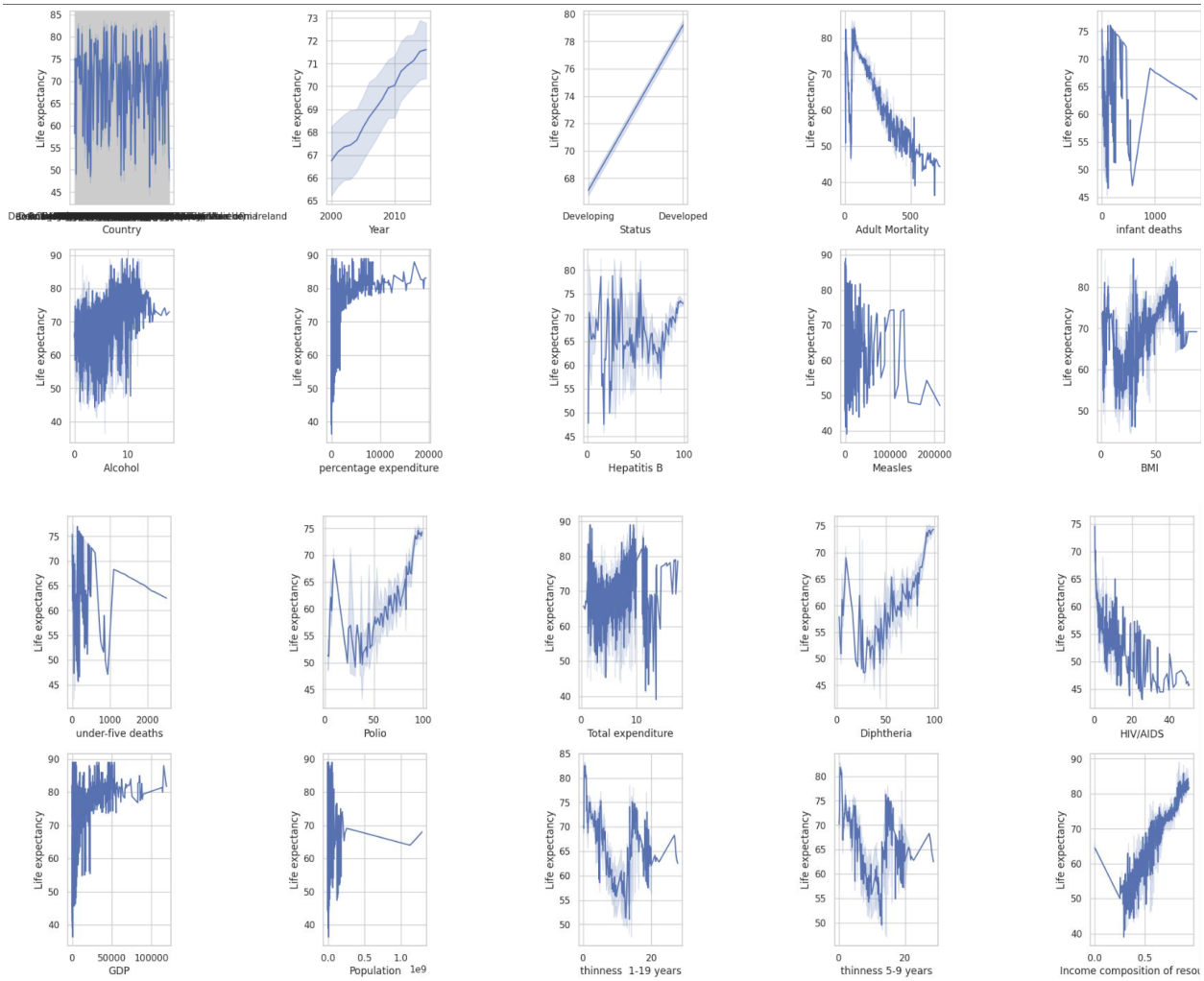
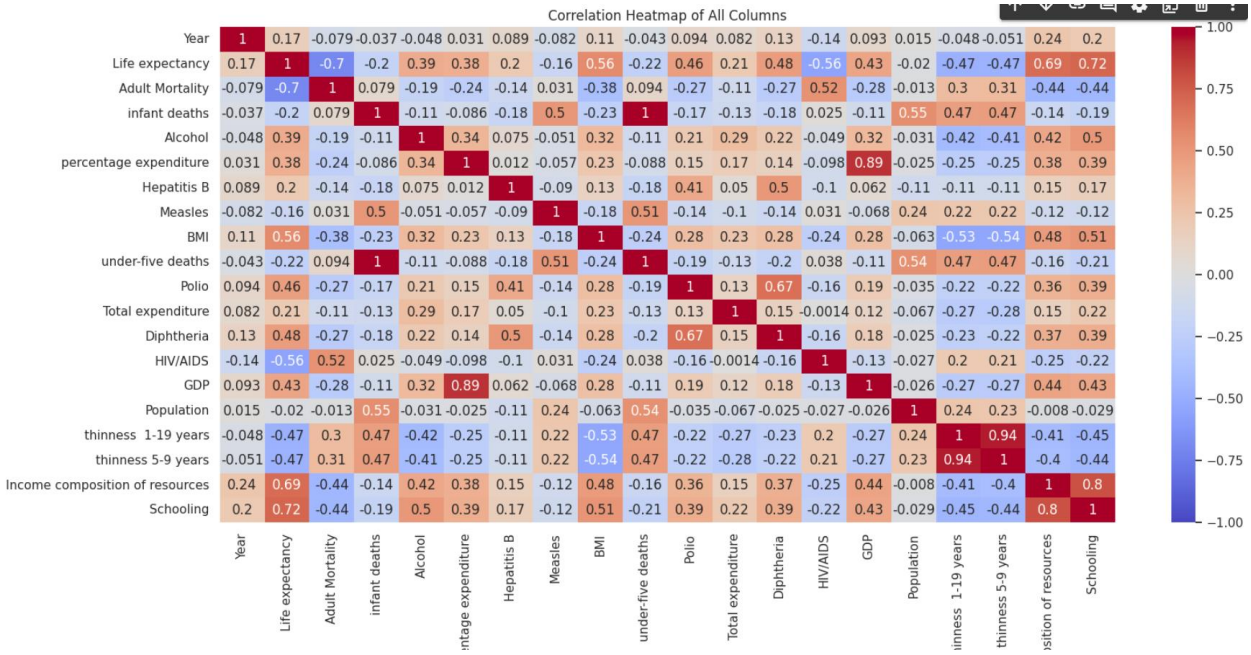


Illustration of correlation between columns using a heatmap

```
corr_matrix = life_xp.corr()

plt.figure(figsize=(18, 12))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of All Columns')
plt.show()
```

Output



Eliminating the white spaces in the column name and replaces value greater than 1000 in column ‘infant deaths’, ‘Measles’, ‘under-five deaths’

```
def cleaning(data):
        # strip column names
        data = data.rename(columns = lambda x:x.strip())

        # Remove wrong values
        cols = ["infant deaths", "Measles", "under-five deaths"]
        for col in cols:
            data.loc[data[col]>1000, col] = np.nan

        return data
```

DATA ENCODING AND MODEL CREATION

Importing necessary libraries

```
# tools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# models
from sklearn.tree import DecisionTreeRegressor, ExtraTreeRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
from sklearn.linear_model import LinearRegression


# metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import confusion_matrix ,ConfusionMatrixDisplay
```

Splitting them to features (x) and target (y) variables and further splitting into testing and training set with a ratio of 80% - 20%

```
from sklearn.model_selection import train_test_split

X = life_xp.drop(columns=["Life expectancy","Year", "Country"])
y = life_xp["Life expectancy"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[32] print("The Shape of X_train is :",X_train.shape)
     print("The Shape of y_train is :",y_train.shape)
     print("The Shape of X_test is :",X_test.shape)
     print("The Shape of y_test is :",y_test.shape)
```

```
The Shape of X_train is : (2350, 19)
The Shape of y_train is : (2350,)
The Shape of X_test is : (588, 19)
The Shape of y_test is : (588,)
```

Identifies Categorical column then encodes categorical variable ‘status’ and also identifies Numerical columns as well. Further more Standardise the numerical features in the training set and applying same to training set. Printing first few rows in the training set

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

X_train['Status'] = encoder.fit_transform(X_train['Status'])
X_test['Status'] = encoder.transform(X_test['Status'])
```

```
[35] num_cols = [col for col in X.columns if X[col].dtype != 'object']
print(num_cols)
```

```
['Adult Mortality', 'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B', 'Measles', 'BMI', 'under-five deaths', 'Polio', 'Total expenditure', 'Diphtheri
```

```
[36] from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train[num_cols] = sc.fit_transform(X_train[num_cols])
X_test[num_cols] = sc.transform(X_test[num_cols])
```

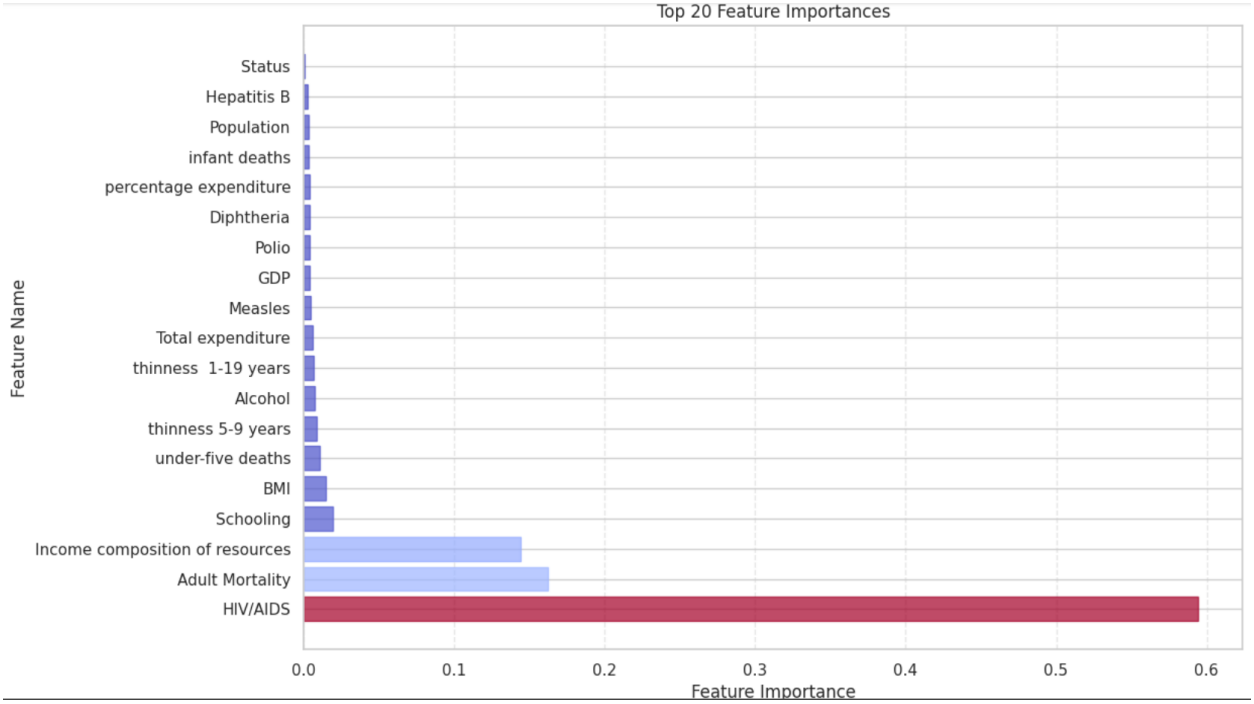
```
X_train.head()
```

	Status	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	thi
456	1	-0.318479	-0.257514	0.176102	-0.192120	0.665664	-0.200110	-0.654211	-0.262215	0.657270	-0.679478	0.658741	-0.225454	-0.327142	-0.240039	0.7
462	1	-0.110535	-0.257514	-0.201213	-0.291864	-0.007317	-0.200110	-0.824603	-0.262215	0.356695	-0.309913	-3.093781	-0.187469	-0.467819	-0.240922	1.0
2172	1	-0.182516	-0.257514	1.497991	-0.369082	0.799728	-0.200110	0.297983	-0.262215	0.700210	0.682516	0.700904	-0.320418	0.006416	0.013396	-0.1
2667	1	-1.222237	-0.230683	-0.848040	-0.064479	0.755040	-0.198729	1.039691	-0.242595	0.657270	0.549638	0.658741	-0.320418	-0.244175	-0.227062	0.3
381	1	-0.566413	-0.257514	-1.145786	0.113299	0.799728	-0.200110	-0.528922	-0.262215	0.700210	-1.053196	0.490088	-0.320418	0.720946	0.013396	0.4

Training a model using Random Forest Regressor and Visualise Top features in ascending manner

```
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import pandas as pd

# Assuming X_train, y_train, and X are already defined
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

feature_importance = model.feature_importances_

# Create a DataFrame to associate feature names with their importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importance})

# Sort the features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(12, 8))
bars = plt.barh(feature_importance_df['Feature'][:20], feature_importance_df['Importance'][:20], color='skyblue')

for bar in bars:
    bar.set_alpha(0.7)  # Set transparency
    bar.set_color(plt.cm.coolwarm(bar.get_width() / max(feature_importance_df['Importance'])))  # Use coolwarm color gradient

plt.xlabel('Feature Importance')
plt.ylabel('Feature Name')
plt.title('Top 20 Feature Importances')
plt.grid(axis='x', linestyle='--', alpha=0.5)  # Add grid lines
plt.show()
```

Output



Calculate both RMSE and R2 score for evaluating the model

```
[43] from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error, r2_score

     model_name = []
     RMSE = []
     R2_score = []

     models = [
         LinearRegression(),
     ]

     for model in models:
         model.fit(X_train, y_train)
         prediction = model.predict(X_test)

         model_name.append(model.__class__.__name__)
         RMSE.append(str(mean_squared_error(prediction, y_test, squared=False)))
         R2_score.append(str(r2_score(y_test, prediction) * 100) + " %")
```

```
model_life_xp = pd.DataFrame({"Model-Name":model_name, "RMSE": RMSE, "R2_Score":R2_score})
model_life_xp = model_life_xp.set_index('Model-Name')
model_life_xp.sort_values("R2_Score", ascending = False)
```

Training the model with a random state of 42 and fits into training data and calculating the R squared score for final model

```
[46] from sklearn.ensemble import ExtraTreesRegressor

     best_model = ExtraTreesRegressor(random_state=42)
     final_model = best_model.fit(X_train, y_train)
```

```
[47] from sklearn.model_selection import cross_val_score
     Random_R2s = cross_val_score(final_model, X_train, y_train, scoring="r2", cv=20)
```

```
[48] pd.Series(Random_R2s).describe()

count    20.000000
mean      0.966842
std       0.011481
min       0.931706
25%       0.961344
50%       0.967234
75%       0.974973
max       0.985083
dtype: float64
```

Calculating RMSE and R squared scores using the predicted values of the targeted variables

```
y_hat = final_model.predict(X_test)

rmse = mean_squared_error(y_test, y_hat, squared=False)
print('rmse: ' + str( rmse ))
r2 = r2_score(y_test, y_hat)
print("R2   : " +str(r2))
```

```
rmse: 1.4503276261324192
R2   : 0.9757206267676519
```

IN SUMMARY, THE FINAL MODEL IS OF ACCURACY (R-SQUARED SCORE) IS 0.9757