

Name: Mira Moheb Attia

ID: 211002019

According to the CDC, heart disease is a leading cause of death for people of most races in the U.S. (African Americans, American Indians and Alaska Natives, and whites). About half of all Americans (47%) have at least 1 of 3 major risk factors for heart disease: high blood pressure, high cholesterol, and smoking. Other key indicators include diabetes status, obesity (high BMI), not getting enough physical activity, or drinking too much alcohol. Identifying and preventing the factors that have the greatest impact on heart disease is very important in healthcare. In turn, developments in computing allow the application of machine learning methods to detect "patterns" in the data that can predict a patient's condition.

Data columns: There are 40 columns in this DataFrame. Each column represents a different variable or attribute of the data. Here's a breakdown of what each column represents:

1. State: The state of the individual.
2. Sex: The gender of the individual.
3. GeneralHealth: The general health status of the individual.
4. PhysicalHealthDays: Number of days of physical health issues.
5. MentalHealthDays: Number of days of mental health issues.
6. LastCheckupTime: Time of last health check-up.
7. PhysicalActivities: Frequency of engaging in physical activities.
8. SleepHours: Average number of hours of sleep.
9. RemovedTeeth: Whether the individual has had teeth removed.
10. HadHeartAttack: Whether the individual has had a heart attack.
11. ... and so on, up to column 40.

Link: [Indicators of Heart Disease \(2022 UPDATE\) \(kaggle.com\)](https://www.kaggle.com/competitions/indicators-of-heart-disease)

• **Checking Missing Values:**

I have Missing Values, this code performs several data preprocessing steps:

1. Define a threshold for removing columns with too many missing values: The threshold is set at 50%, meaning any column with more than 50% missing values will be dropped.
2. Calculate the threshold number of missing values: This is calculated based on the total number of rows in the DataFrame.
3. Drop columns with more than the threshold number of missing values: Using the `dropna()` function with the `thresh` parameter set to `threshold_value`, columns exceeding the threshold are dropped.
4. Fill missing values for numeric columns with their mean: For numeric columns (columns with data types `number`), missing values are filled with the mean value of the respective column.
5. Fill missing values for categorical columns with their mode: For categorical columns (columns with data types `object` or `category`), missing values are filled with the mode (most frequent value) of the respective column.

6. Display the remaining columns after dropping: This prints out the column names of the DataFrame after dropping columns with too many missing values.

```
[6]: #Checkin for missing values
missing_values = df.isnull().sum()
print(missing_values)
```

State	0
Sex	0
GeneralHealth	1198
PhysicalHealthDays	10927
MentalHealthDays	9067
LastCheckupTime	8308
PhysicalActivities	1093
SleepHours	5453
RemovedTeeth	11360
HadHeartAttack	3065
HadAngina	4405
HadStroke	1557
HadAsthma	1773
HadSkinCancer	3143
HadCOPD	2219
HadDepressiveDisorder	2812
HadKidneyDisease	1926
HadArthritis	2633
HadDiabetes	1087
DeafOrHardOfHearing	20647
BlindOrVisionDifficulty	21564
DifficultyConcentrating	24240
DifficultyWalking	24012
DifficultyDressingBathing	23915
DifficultyErrands	25656
SmokerStatus	35462
ECigaretteUsage	35660
ChestScan	56046
RaceEthnicityCategory	14057
AgeCategory	9079
HeightInMeters	28652
WeightInKilograms	42078
BMI	48806
AlcoholDrinkers	46574

Remaining missing values:	
State	0
Sex	0
GeneralHealth	0
PhysicalHealthDays	0
MentalHealthDays	0
LastCheckupTime	0
PhysicalActivities	0
SleepHours	0
RemovedTeeth	0
HadHeartAttack	0
HadAngina	0
HadStroke	0
HadAsthma	0
HadSkinCancer	0
HadCOPD	0
HadDepressiveDisorder	0
HadKidneyDisease	0
HadArthritis	0
HadDiabetes	0
DeafOrHardOfHearing	0
BlindOrVisionDifficulty	0
DifficultyConcentrating	0
DifficultyWalking	0
DifficultyDressingBathing	0
DifficultyErrands	0
SmokerStatus	0
ECigaretteUsage	0
ChestScan	0
RaceEthnicityCategory	0
AgeCategory	0
HeightInMeters	0
WeightInKilograms	0
BMI	0
AlcoholDrinkers	0
HIVTesting	0
FluVaxLast12	0

• Handling Duplicate Rows:

1. Identify duplicate rows: `df.duplicated()` checks for duplicate rows in the DataFrame. Each duplicate row is marked as `True`, and non-duplicate rows are marked as `False`.
2. Count the total number of duplicate rows: `df.duplicated().sum()` calculates the sum of duplicate rows, which is 305 in this case.
3. Drop duplicate rows: `df.drop_duplicates(keep='first', inplace=True)` removes duplicate rows from the DataFrame. The parameter `keep='first'` keeps the first occurrence of duplicated rows and removes the subsequent duplicates.
4. Recheck for duplicate rows: `df.duplicated().sum()` is called again to confirm that all duplicates have been removed. The result is 0, indicating that there are no more duplicate rows in the DataFrame.

```
[8]: # Duplicate values
      df.duplicated()
      df.duplicated().sum()
```

```
[8]: 305
```

```
[9]: df.drop_duplicates(keep='first',inplace=True)
```

```
[10]: df.duplicated().sum()
```

```
[10]: 0
```

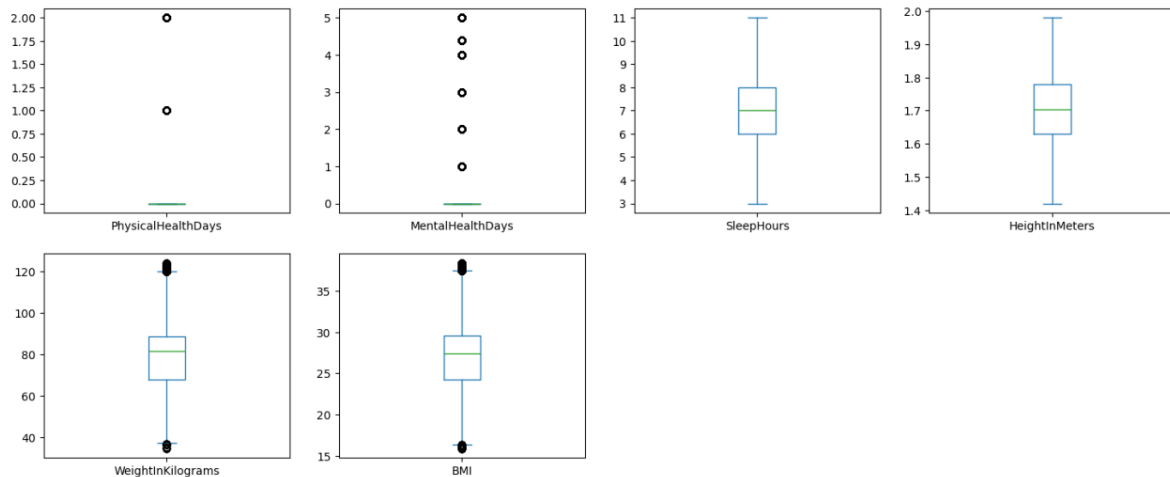
• Detecting Outliers:

1. Conversion to numeric data type: Certain columns (`PhysicalHealthDays`, `MentalHealthDays`, `SleepHours`, `HeightInMeters`, `WeightInKilograms`, `BMI`) are converted to numeric data type using `pd.to_numeric()`. Errors are coerced to `NaN`, meaning any non-numeric values will be replaced with `NaN`.
2. Calculation of quartiles: First and third quartiles (`q1` and `q3`) are calculated for the specified columns.
3. Calculation of Interquartile Range (IQR): The interquartile range (IQR) is calculated for each specified column.
4. Calculation of lower and upper bounds for outliers: Lower and upper bounds for outliers are calculated based on the IQR and a multiplier of 1.5.
5. Identification of outliers: Outliers are identified using the calculated bounds for each specified column. Any data point falling outside the calculated bounds is considered an outlier.

6. Handling outliers: Outliers are handled by removing them. The DataFrame data retains only the rows where the values fall within the calculated bounds for each specified column.

I repeated this step twice

```
[15]: data2.plot(kind = "box" , subplots = True , figsize = (18,15) , layout = (4,4))  
plt.show()
```



It is known that being overweight is linked to heart disease

- **Checking Data Consistency and Distribution:**

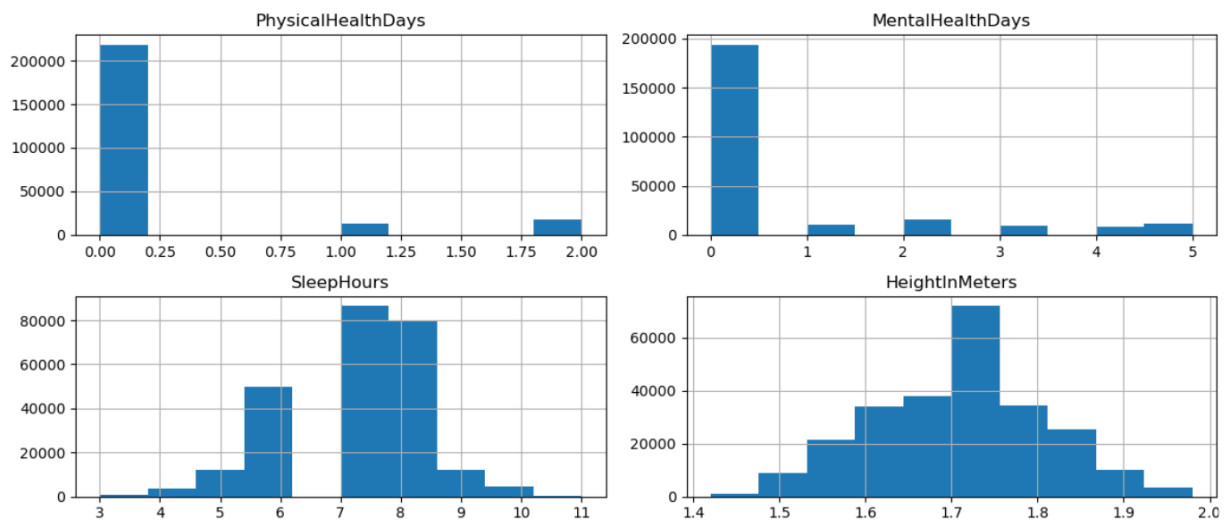
Categorical Features:

1. Sex: Gender of the individual.
2. GeneralHealth: General health status of the individual.
3. PhysicalHealthDays: Number of days of physical health issues.
4. LastCheckupTime: Time of last health check-up.
5. PhysicalActivities: Frequency of engaging in physical activities.
6. RemovedTeeth: Whether the individual has had teeth removed.
7. HadHeartAttack: Whether the individual has had a heart attack.
8. HadAngina: Whether the individual has had angina.
9. HadStroke: Whether the individual has had a stroke.
10. HadAsthma: Whether the individual has had asthma.
11. HadSkinCancer: Whether the individual has had skin cancer.
12. HadCOPD: Whether the individual has had Chronic Obstructive Pulmonary Disease (COPD).
13. HadDepressiveDisorder: Whether the individual has had a depressive disorder.
14. HadKidneyDisease: Whether the individual has had kidney disease.
15. HadArthritis: Whether the individual has had arthritis.
16. HadDiabetes: Whether the individual has had diabetes.

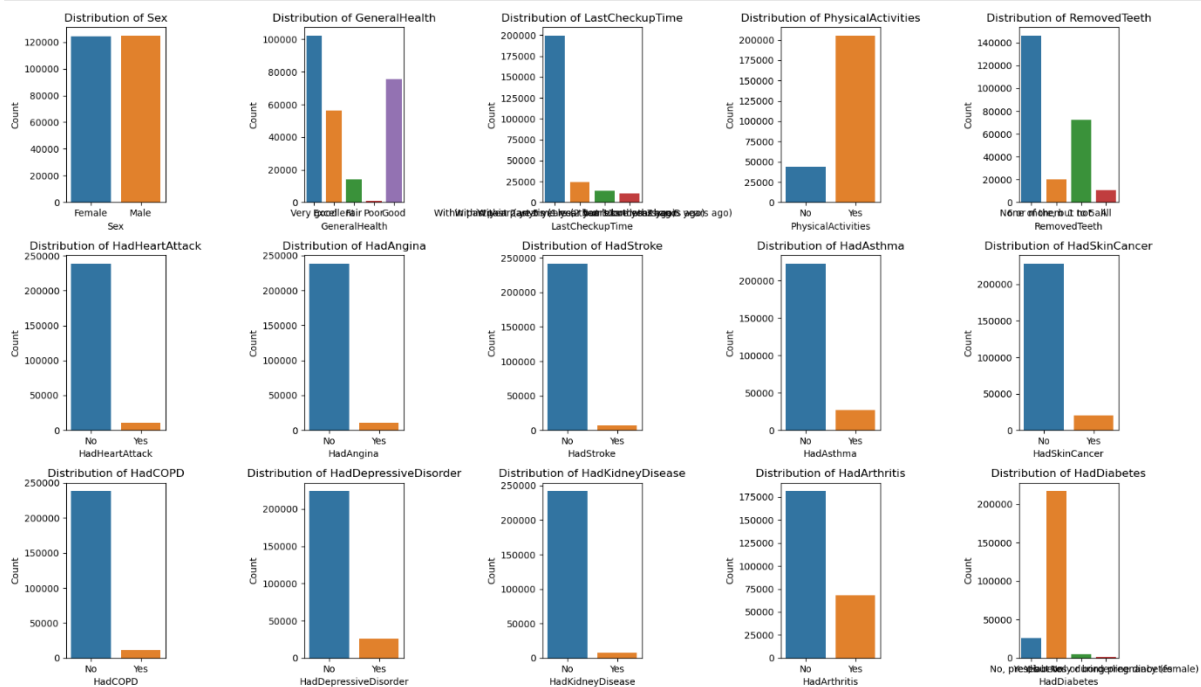
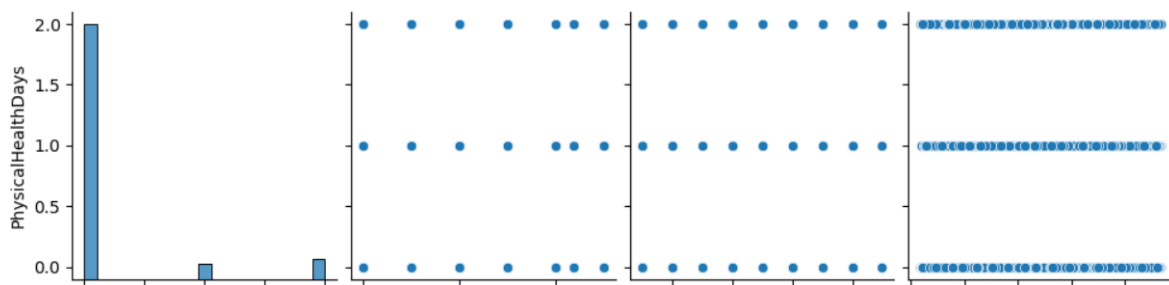
17. DeafOrHardOfHearing: Whether the individual is deaf or hard of hearing.
18. BlindOrVisionDifficulty: Whether the individual has blindness or vision difficulty.
19. DifficultyConcentrating: Whether the individual has difficulty concentrating.
20. DifficultyWalking: Whether the individual has difficulty walking.
21. DifficultyDressingBathing: Whether the individual has difficulty dressing or bathing.
22. DifficultyErrands: Whether the individual has difficulty running errands.
23. SmokerStatus: Smoking status of the individual.
24. ECigaretteUsage: E-cigarette usage status of the individual.
25. ChestScan: Whether the individual has undergone a chest scan.
26. RaceEthnicityCategory: Race or ethnicity category of the individual.
27. AlcoholDrinkers: Whether the individual consumes alcohol.
28. HIVTesting: Whether the individual has undergone HIV testing.
29. FluVaxLast12: Whether the individual has received a flu vaccine in the last 12 months.
30. PneumoVaxEver: Whether the individual has ever received a pneumonia vaccine.
31. TetanusLast10Tdap: Whether the individual has received a tetanus vaccine in the last 10 years.
32. HighRiskLastYear: Whether the individual has been at high risk for any health condition in the last year.
33. CovidPos: Whether the individual has tested positive for COVID-19.

Numerical Features:

1. State: State of residence of the individual.
2. MentalHealthDays: Number of days of mental health issues.
3. SleepHours: Average number of hours of sleep.
4. AgeCategory: Age category of the individual.
5. HeightInMeters: Height of the individual in meters.
6. WeightInKilograms: Weight of the individual in kilograms.
7. BMI: Body Mass Index (BMI) of the individual.



Pairplot of Numerical Variables



- **statistical results for different analyses**

- T-Statistic and P-Value: The T-statistic is a measure of the difference between the means of two groups, relative to the spread of data points. The P-value indicates the probability of observing a T-statistic as extreme as the one calculated, assuming that the null hypothesis (typically, that there is no difference between the groups) is true. However, in the provided result, both the T-statistic and P-value are reported as "nan," which likely means there was an issue with the calculation or the data.
- Pearson Correlation Coefficient and P-Value: The Pearson correlation coefficient measures the strength and direction of the linear relationship between two variables. The P-value associated with the correlation coefficient indicates the probability of observing the calculated correlation coefficient under the assumption that there is no true correlation in the population. In the provided result, the Pearson correlation coefficient is approximately 0.0154, indicating a very weak positive correlation. The P-value is reported as 1.7652166287574312e-14, which is extremely small, suggesting strong evidence against the null hypothesis of no correlation.
- Confidence Interval for Mean Physical Health Days: This provides an interval estimate for the mean number of physical health days. The confidence interval is (0.1928, 0.1971), suggesting that we are 95% confident that the true mean number of physical health days falls within this interval.

```
[21]: # Statistical Summaries
# Test for differences in means (e.g., General Health between Smokers and Non-Smokers)
smokers = data2[data2['SmokerStatus'] == 'Smoker']
non_smokers = data2[data2['SmokerStatus'] == 'Non-Smoker']

# Check for missing data and drop if necessary to ensure equal lengths
smokers.dropna(subset=['GeneralHealth'], inplace=True)
non_smokers.dropna(subset=['GeneralHealth'], inplace=True)

[22]: # Perform t-test
t_stat, p_value = ttest_ind(smokers['GeneralHealth'], non_smokers['GeneralHealth'])
print(f"T-Statistic: {t_stat}, P-Value: {p_value}")

# Calculate correlation between two numerical variables (e.g., BMI and Physical Health Days)
corr_coefficient, p_value = pearsonr(data2['BMI'], data2['PhysicalHealthDays'])
print(f"Pearson Correlation Coefficient: {corr_coefficient}, P-Value: {p_value}")

# Compute confidence intervals for key statistics (e.g., mean of Physical Health Days)
mean_physical_health_days = data2['PhysicalHealthDays'].mean()
std_physical_health_days = data2['PhysicalHealthDays'].std()
n_samples = len(data2['PhysicalHealthDays'])
confidence_interval = 1.96 * (std_physical_health_days / (n_samples ** 0.5)) # Assuming 95% confidence level
lower_bound = mean_physical_health_days - confidence_interval
upper_bound = mean_physical_health_days + confidence_interval
print(f"Confidence Interval for Mean Physical Health Days: ({lower_bound}, {upper_bound})")

T-Statistic: nan, P-Value: nan
Pearson Correlation Coefficient: 0.015353034961415393, P-Value: 1.7652166287574312e-14
Confidence Interval for Mean Physical Health Days: (0.19283258514753338, 0.19712939325876827)
```


- Feature Engineering:

creates a new feature called HealthRiskScore based on the presence of various health conditions.

1. Definition of Health Risk Score Calculation Function: The function `calculate_health_risk(row)` takes a row of the DataFrame as input and calculates a health risk score based on the presence of certain health conditions. The score starts at 0 and increments by 1 for each health condition that the individual has.
2. List of Health Conditions: The function uses a list called `health_conditions` to specify the health conditions to be considered in the calculation. These include 'HadHeartAttack', 'HadAngina', 'HadStroke', 'HadAsthma', 'HadSkinCancer', 'HadCOPD', 'HadDepressiveDisorder', 'HadKidneyDisease', 'HadArthritis', and 'HadDiabetes'.
3. Loop Through Health Conditions: The function iterates through each health condition in the list and checks if the individual has that condition (i.e., if the value in the respective column is 'Yes'). If the condition is present, the score is incremented by 1.
4. Assigning Health Risk Score to Each Row: The function returns the calculated health risk score for the input row.
5. Application of the Function to Create the New Feature: The function `calculate_health_risk` is applied to each row of the DataFrame `data2` using the `apply()` function along `axis=1`, meaning it operates row-wise.
6. Verification of the New Feature: The print statement verifies the new feature `HealthRiskScore` by displaying the first few rows of this column in the DataFrame.

```
: # Create a Health Risk Score as a combination of various health indicators
def calculate_health_risk(row):
    score = 0
    health_conditions = ['HadHeartAttack', 'HadAngina', 'HadStroke', 'HadAsthma', 'HadSkinCancer',
                        'HadCOPD', 'HadDepressiveDisorder', 'HadKidneyDisease', 'HadArthritis', 'HadDiabetes']

    for condition in health_conditions:
        if row[condition] == 'Yes':
            score += 1

    return score

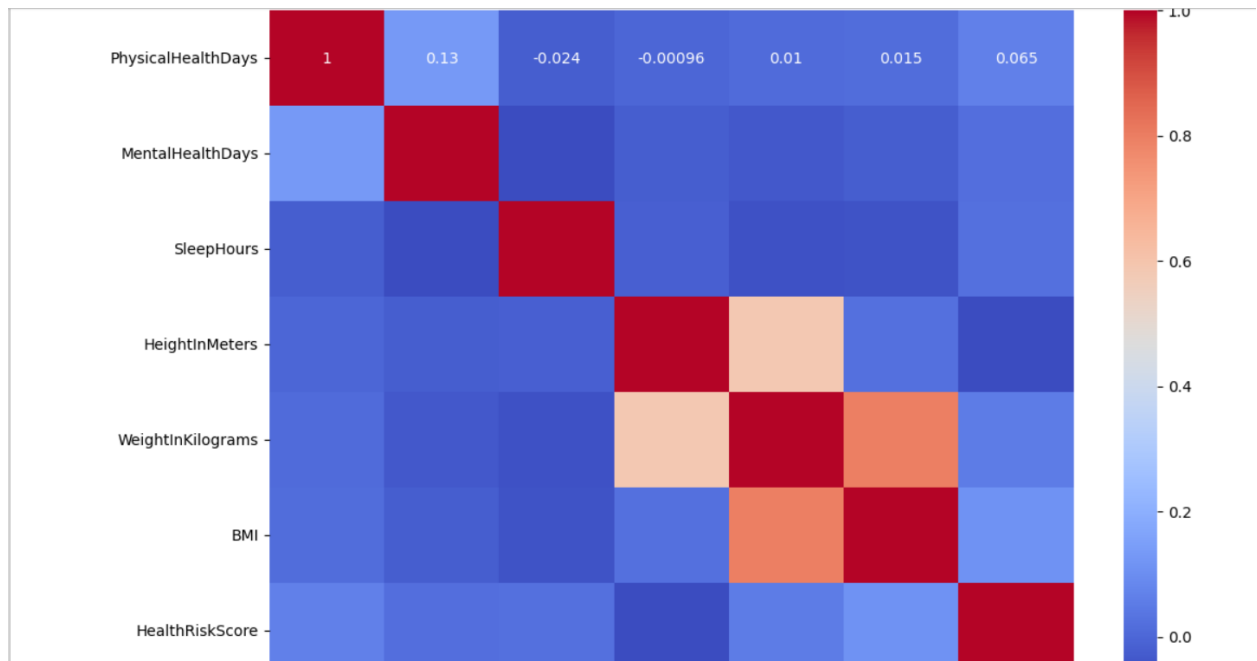
data2['HealthRiskScore'] = data2.apply(calculate_health_risk, axis=1)

# Verify the new feature
print(data2[['HealthRiskScore']].head())
```

```
HealthRiskScore
0                1
1                1
2                1
3                2
4                0
```

This code checks if the 'HealthRiskScore' column exists in the DataFrame data2. If the column exists, it proceeds to analyze its correlation with other numeric features in the dataset. Here's what each part of the code does:

1. Check for 'HealthRiskScore' column: The code checks if the 'HealthRiskScore' column is present in the DataFrame. If it's not found, it prints a message indicating that the column is not present.
2. Filter Numeric Columns: If the 'HealthRiskScore' column exists, the code selects only the numeric columns from the DataFrame data2.
3. Calculate Correlation Matrix: It calculates the correlation matrix for the selected numeric features. The correlation matrix shows the pairwise correlation coefficients between all pairs of numeric features.
4. Plot Heatmap: It plots a heatmap of the correlation matrix using seaborn. This heatmap provides a visual representation of the correlation between different numeric features. Higher correlation coefficients are represented by warmer colors (e.g., red), while lower correlation coefficients are represented by cooler colors (e.g., blue).
5. Select Highly Correlated Features: It sets a threshold for correlation (correlation_threshold) and selects features that have a correlation coefficient above this threshold with the 'HealthRiskScore'. If the 'HealthRiskScore' column is found in the correlation matrix, it prints the list of highly correlated features.



- **Evaluation Results: Using the evaluation metrics that were derived from the models:**

- **Random Forest:** Achieved an accuracy of 45.63%, with precision and recall also around 45.63%.
- **Gradient Boosting:** Demonstrated improved performance with an accuracy of 47.14%, precision of 46.56%, and recall of 47.14%.
- **Decision Tree:** Showed lower performance compared to other models, with an accuracy of 37.14% and similar precision and recall.
- **Logistic Regression:** Achieved an accuracy of 47.11%, with precision and recall also around 47.11%.
- **XGBoost:** Exhibited the highest accuracy among all models at 47.18%, with precision slightly lower at 46.25% but recall matching accuracy at 47.18%.

the precision and recall scores for each machine learning model:

- **Random Forest:**
 - Precision: 0.4503
 - Recall: 0.4589
- **Gradient Boosting:**
 - Precision: 0.4656
 - Recall: 0.4714
- **Decision Tree:**
 - Precision: 0.3675
 - Recall: 0.3691
- **Logistic Regression:**
 - Precision: 0.4653
 - Recall: 0.4711
- **XGBoost:**
 - Precision: 0.4625
 - Recall: 0.4718

```

from sklearn.metrics import precision_score, recall_score

# Iterate over each model
for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predict the labels for the test data
    y_pred = model.predict(X_test)

    # Calculate precision and recall scores
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    # Print precision and recall scores for each model
    print(f"{name} Precision:", precision)
    print(f"{name} Recall:", recall)

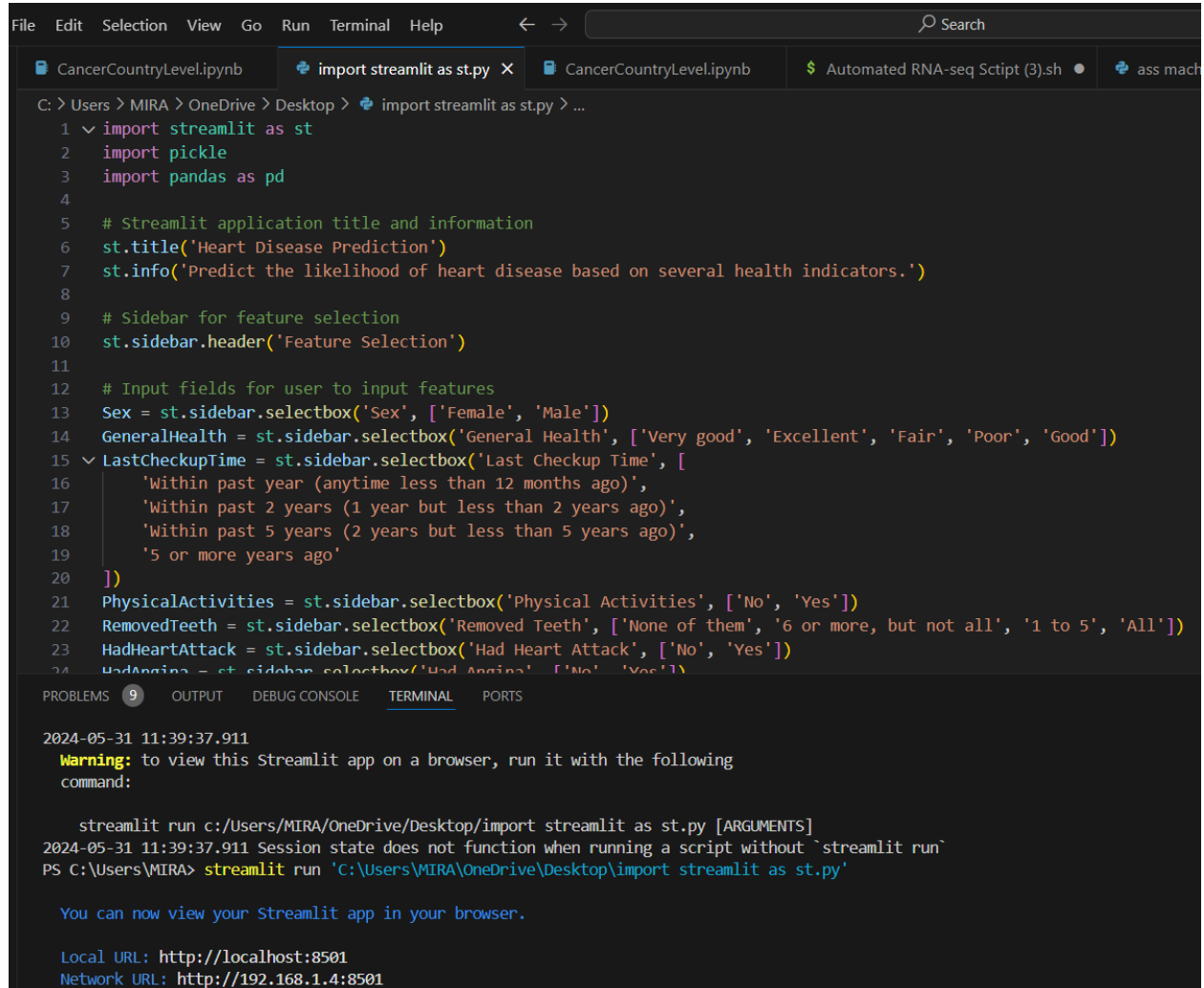
```

```

Random Forest Precision: 0.450309519719263
Random Forest Recall: 0.45890067579762167
Gradient Boosting Precision: 0.4655503947053316
Gradient Boosting Recall: 0.4714139611366234
Decision Tree Precision: 0.3675381669122728
Decision Tree Recall: 0.3690617041329938
Logistic Regression Precision: 0.4652535429561157
Logistic Regression Recall: 0.47113321435017147
XGBoost Precision: 0.462499550192187
XGBoost Recall: 0.47181502797441194

```

- GUI using streamlit



The image shows a VS Code editor window with a file explorer on the left and a code editor in the center. The file explorer shows a project named 'CancerCountryLevel.ipynb'. The code editor contains a Python script for a Streamlit application titled 'Heart Disease Prediction'. The script imports 'streamlit' as 'st', 'pickle', and 'pandas' as 'pd'. It sets the title and info, then creates a sidebar for feature selection. The sidebar contains five selectboxes: 'Sex' (options: 'Female', 'Male'), 'General Health' (options: 'Very good', 'Excellent', 'Fair', 'Poor', 'Good'), 'Last Checkup Time' (options: 'Within past year (anytime less than 12 months ago)', 'Within past 2 years (1 year but less than 2 years ago)', 'Within past 5 years (2 years but less than 5 years ago)', '5 or more years ago'), 'Physical Activities' (options: 'No', 'Yes'), 'Removed Teeth' (options: 'None of them', '6 or more, but not all', '1 to 5', 'All'), and 'Had Heart Attack' (options: 'No', 'Yes').

```
C: > Users > MIRA > OneDrive > Desktop > import streamlit as st.py > ...
1  import streamlit as st
2  import pickle
3  import pandas as pd
4
5  # Streamlit application title and information
6  st.title('Heart Disease Prediction')
7  st.info('Predict the likelihood of heart disease based on several health indicators.')
8
9  # Sidebar for feature selection
10 st.sidebar.header('Feature Selection')
11
12 # Input fields for user to input features
13 Sex = st.sidebar.selectbox('Sex', ['Female', 'Male'])
14 GeneralHealth = st.sidebar.selectbox('General Health', ['Very good', 'Excellent', 'Fair', 'Poor', 'Good'])
15 LastCheckupTime = st.sidebar.selectbox('Last Checkup Time', [
16     'Within past year (anytime less than 12 months ago)',
17     'Within past 2 years (1 year but less than 2 years ago)',
18     'Within past 5 years (2 years but less than 5 years ago)',
19     '5 or more years ago'
20 ])
21 PhysicalActivities = st.sidebar.selectbox('Physical Activities', ['No', 'Yes'])
22 RemovedTeeth = st.sidebar.selectbox('Removed Teeth', ['None of them', '6 or more, but not all', '1 to 5', 'All'])
23 HadHeartAttack = st.sidebar.selectbox('Had Heart Attack', ['No', 'Yes'])
24 HadAngina = st.sidebar.selectbox('Had Angina', ['No', 'Yes'])
```

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS

2024-05-31 11:39:37.911
Warning: to view this Streamlit app on a browser, run it with the following command:

```
streamlit run c:/Users/MIRA/OneDrive/Desktop/import streamlit as st.py [ARGUMENTS]
```

2024-05-31 11:39:37.911 Session state does not function when running a script without `streamlit run`
PS C:\Users\MIRA> streamlit run 'C:\Users\MIRA\OneDrive\Desktop\import streamlit as st.py'

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8501>
Network URL: <http://192.168.1.4:8501>

×

Feature Selection

Sex

Female

General Health

Very good

Last Checkup Time

Within past year (anytime less t...

Physical Activities

No

Removed Teeth

None of them

Had Heart Attack

No

Had Angina

No

Heart Disease Prediction

Predict the likelihood of heart disease based on several health indicators.

- Conclusion:

In conclusion, after sequentially training and evaluating five machine learning models—Random Forest, Gradient Boosting, Decision Tree, Logistic Regression, and XGBoost—the XGBoost model demonstrated the highest accuracy at approximately 47.18%. Further hyperparameter tuning, focusing on `'max_depth'` and `'n_estimators'`, resulted in an optimized XGBoost model with an accuracy of around 47.27% on a separate test dataset. This model represents a valuable tool for predicting heart disease risk based on health indicators, with potential for future enhancements through additional data collection and optimization techniques.