```
### Name: Naheed Sajjad
### Student ID:A00326697
### FinalProject_MachineLearning
```

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np # linear algebra
import matplotlib.pyplot as plt
import seaborn  as sns
import warnings
warnings.filterwarnings("ignore")
import copy
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix,
## Data used "heart_2022_with_nans.csv" downloaded from "https://www.kaggle.com/datasets/kamilpytlak/personal-key-ind
```

```
## Step 1: Load the dataset
df = pd.read_csv("heart_2022_with_nans.csv")
# Display first few rows
df.head()
```

	State	Sex	GeneralHealth	PhysicalHealthDays	MentalHealthDays	LastCheckupTime	PhysicalActivities	SleepHours
0	Alabama	Female	Very good	0.0	0.0	Within past year (anytime less than 12 months ...	No	8.0
1	Alabama	Female	Excellent	0.0	0.0	NaN	No	6.0
2	Alabama	Female	Very good	2.0	3.0	Within past year (anytime less than 12 months ...	Yes	5.0
3	Alabama	Female	Excellent	0.0	0.0	Within past year (anytime less than 12 months ...	Yes	7.0
4	Alabama	Female	Fair	2.0	0.0	Within past year (anytime less than 12 months ...	Yes	9.0

5 rows × 40 columns

```
## Step 2: Understand Dataset Structure
# Check dataset shape (no of rows*no of columns)
df.shape
```

```
(445132, 40)
```

```
# Display column names
df.columns
```

```
Index(['State', 'Sex', 'GeneralHealth', 'PhysicalHealthDays',
       'MentalHealthDays', 'LastCheckupTime', 'PhysicalActivities',
       'SleepHours', 'RemovedTeeth', 'HadHeartAttack', 'HadAngina',
       'HadStroke', 'HadAsthma', 'HadSkinCancer', 'HadCOPD',
       'HadDepressiveDisorder', 'HadKidneyDisease', 'HadArthritis',
       'HadDiabetes', 'DeafOrHardOfHearing', 'BlindOrVisionDifficulty',
       'DifficultyConcentrating', 'DifficultyWalking',
       'DifficultyDressingBathing', 'DifficultyErrands', 'SmokerStatus',
       'ECigaretteUsage', 'ChestScan', 'RaceEthnicityCategory', 'AgeCategory',
       'HeightInMeters', 'WeightInKilograms', 'BMI', 'AlcoholDrinkers',
       'HIVTesting', 'FluVaxLast12', 'PneumoVaxEver', 'TetanusLast10Tdap',
       'HighRiskLastYear', 'CovidPos'],
      dtype='object')
```

```
## Step 3: Handle Missing Values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 445132 entries, 0 to 445131
Data columns (total 40 columns):
 #   Column                    Non-Null Count   Dtype
```

```
 ---  ------                      --------------   -----
  0   State                       445132 non-null  object
  1   Sex                         445132 non-null  object
  2   GeneralHealth               443934 non-null  object
  3   PhysicalHealthDays          434205 non-null  float64
  4   MentalHealthDays            436065 non-null  float64
  5   LastCheckupTime             436824 non-null  object
  6   PhysicalActivities          444039 non-null  object
  7   SleepHours                  439679 non-null  float64
  8   RemovedTeeth                433772 non-null  object
  9   HadHeartAttack              442067 non-null  object
  10  HadAngina                   440727 non-null  object
  11  HadStroke                   443575 non-null  object
  12  HadAsthma                   443359 non-null  object
  13  HadSkinCancer               441989 non-null  object
  14  HadCOPD                     442913 non-null  object
  15  HadDepressiveDisorder       442320 non-null  object
  16  HadKidneyDisease            443206 non-null  object
  17  HadArthritis                442499 non-null  object
  18  HadDiabetes                 444045 non-null  object
  19  DeafOrHardOfHearing         424485 non-null  object
  20  BlindOrVisionDifficulty     423568 non-null  object
  21  DifficultyConcentrating     420892 non-null  object
  22  DifficultyWalking           421120 non-null  object
  23  DifficultyDressingBathing   421217 non-null  object
  24  DifficultyErrands           419476 non-null  object
  25  SmokerStatus                409670 non-null  object
  26  ECigaretteUsage             409472 non-null  object
  27  ChestScan                   389086 non-null  object
  28  RaceEthnicityCategory       431075 non-null  object
  29  AgeCategory                 436053 non-null  object
  30  HeightInMeters              416480 non-null  float64
  31  WeightInKilograms           403054 non-null  float64
  32  BMI                         396326 non-null  float64
  33  AlcoholDrinkers             398558 non-null  object
  34  HIVTesting                  379005 non-null  object
  35  FluVaxLast12                398011 non-null  object
  36  PneumoVaxEver               368092 non-null  object
  37  TetanusLast10Tdap           362616 non-null  object
  38  HighRiskLastYear            394509 non-null  object
  39  CovidPos                    394368 non-null  object
dtypes: float64(6), object(34)
memory usage: 135.8+ MB
```

```
# Check for missing values
df.isnull().sum() #No missing value detected.
```

	0
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
HIVTesting	66127
FluVaxLast12	47121
PneumoVaxEver	77040
TetanusLast10Tdap	82516
HighRiskLastYear	50623
CovidPos	50764

dtype: int64

```
# remove any rows that contain missing values (NaN) and to modify the DataFrame directly.
df.dropna(inplace=True)
```

```
## Step 4: Check for Duplicate Records
# Check for duplicates
```

```
df.duplicated().sum()
```

```
np.int64(9)
```

```
#parameters keep the first occurrence of each duplicated row and remove subsequent.
# Performs the operation directly on the existing DataFrame.
df.drop_duplicates(keep='first', inplace=True)
```

```
df.shape
```

```
(246013, 40)
```

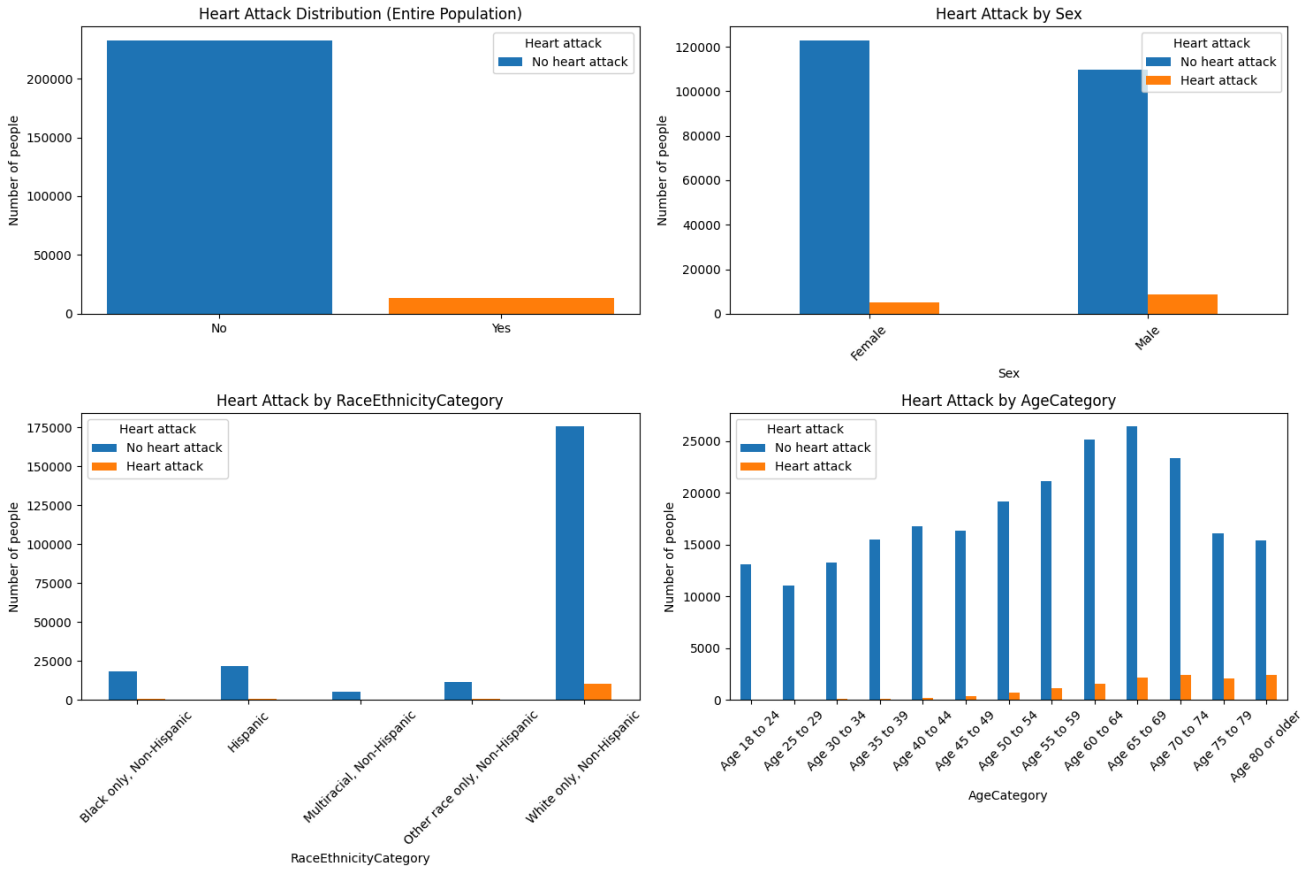
```
## Step 5: Summary Statistics of Data
df.describe()
```

	PhysicalHealthDays	MentalHealthDays	SleepHours	HeightInMeters	WeightInKilograms	BMI
count	246013.000000	246013.000000	246013.000000	246013.000000	246013.000000	246013.000000
mean	4.119055	4.167292	7.021312	1.705150	83.615522	28.668258
std	8.405803	8.102796	1.440698	0.106654	21.323232	6.514005
min	0.000000	0.000000	1.000000	0.910000	28.120000	12.020000
25%	0.000000	0.000000	6.000000	1.630000	68.040000	24.270000
50%	0.000000	0.000000	7.000000	1.700000	81.650000	27.460000
75%	3.000000	4.000000	8.000000	1.780000	95.250000	31.890000
max	30.000000	30.000000	24.000000	2.410000	292.570000	97.650000

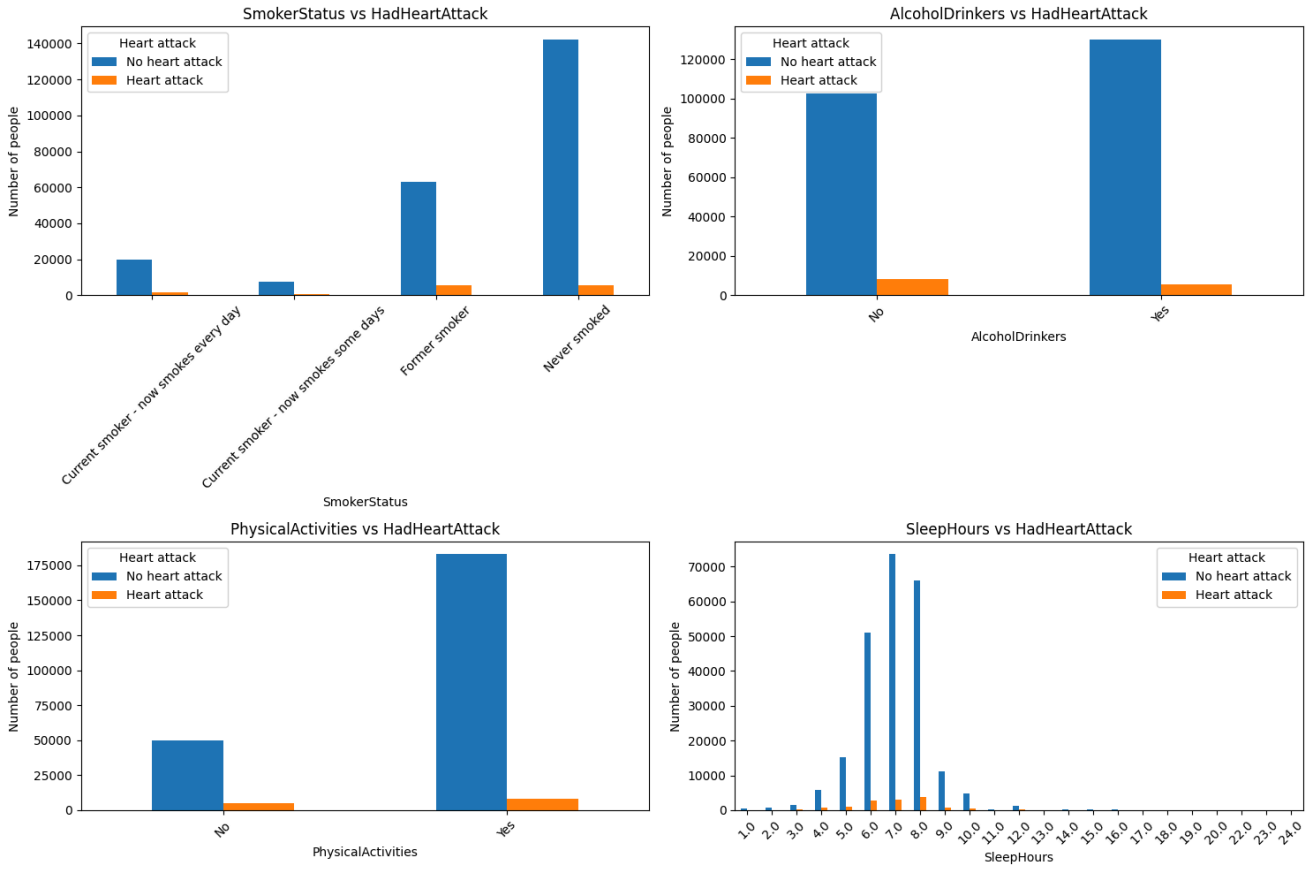
```
#Check for unique values in each column
for col in df.columns:
    print(f"Value counts for column '{col}':")
    print(df[col].value_counts())
    print("-" * 30)    #repeats the string "-" 30 times.
```

```
Yes, received tetanus shot, but not Tdap                        19870
Name: count, dtype: int64
------------------------------
Value counts for column 'HighRiskLastYear':
HighRiskLastYear
No     235437
Yes     10576
Name: count, dtype: int64
------------------------------
Value counts for column 'CovidPos':
CovidPos
No                                                             167297
Yes                                                             70324
Tested positive using home test without a health professional    8392
Name: count, dtype: int64
------------------------------
```

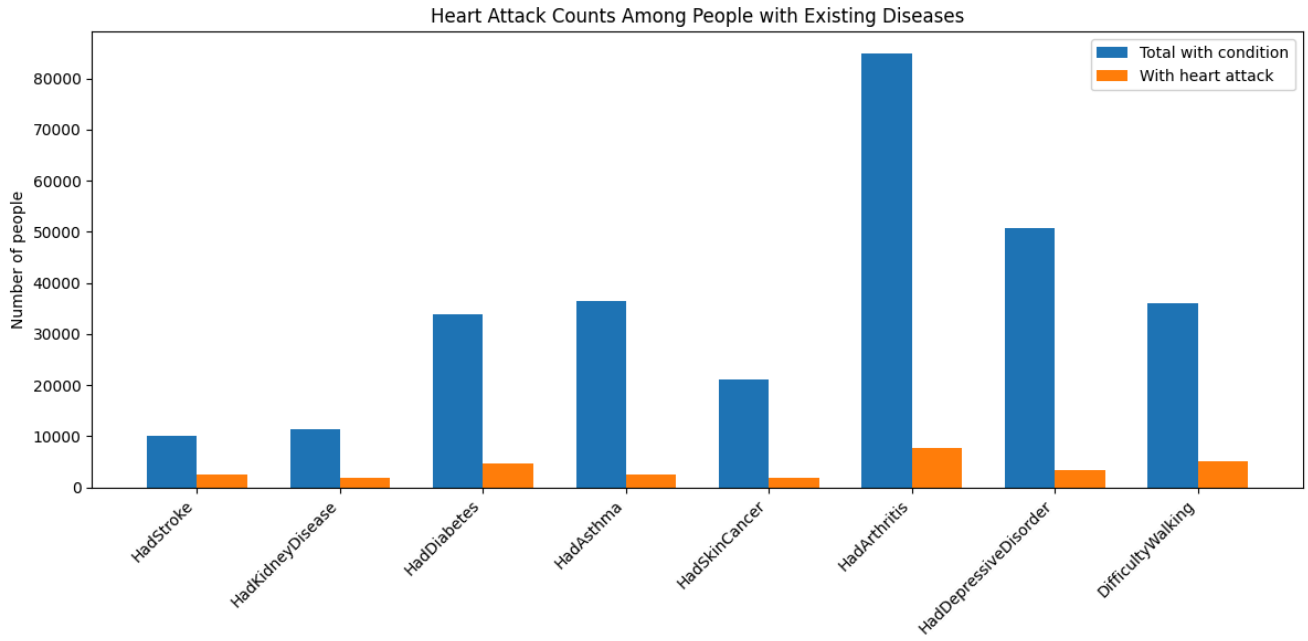
```
## Step-6: Exploratory Data Analysis Visualizations
#df["HadHeartAttackNum"] = df["HadHeartAttack"].map({"No": 0, "Yes": 1})
#df['HadHeartAttack'].value_counts().plot(kind = 'bar')
heart_col = "HadHeartAttack"
def clustered_plot_counts(ax, df, category_col, heart_col):
    # count (category, heart attack) combinations
    grouped = df.groupby([category_col, heart_col]).size().unstack(fill_value=0)

    # make sure both "No" and "Yes" columns exist and are ordered
    for val in ["No", "Yes"]:
        if val not in grouped.columns:
            grouped[val] = 0
    grouped = grouped[["No", "Yes"]]

    # clustered bar chart with counts and fixed colors
    grouped.plot(kind='bar', ax=ax, color=["tab:blue", "tab:orange"])

    ax.set_title(f"Heart Attack by {category_col}")
    ax.set_ylabel("Number of people")
    ax.tick_params(axis='x', rotation=45)

    ax.legend(["No heart attack", "Heart attack"], title="Heart attack")

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# 1) Distribution of heart attack (entire population) - counts
counts = df[heart_col].value_counts()

# ensure "No" and "Yes" both present and ordered
for val in ["No", "Yes"]:
    if val not in counts.index:
        counts[val] = 0
counts = counts[["No", "Yes"]]

axes[0, 0].bar(counts.index, counts.values, color=["tab:blue", "tab:orange"])
axes[0, 0].set_title("Heart Attack Distribution (Entire Population)")
axes[0, 0].set_ylabel("Number of people")
axes[0, 0].tick_params(axis='x', rotation=0)

# consistent legend
axes[0, 0].legend(["No heart attack", "Heart attack"], title="Heart attack")

# 2) Heart Attack vs Sex (counts, clustered)
clustered_plot_counts(axes[0, 1], df, "Sex", heart_col)

# 3) Heart Attack vs RaceEthnicityCategory (counts, clustered)
clustered_plot_counts(axes[1, 0], df, "RaceEthnicityCategory", heart_col)

# 4) Heart Attack vs AgeCategory (counts, clustered)
clustered_plot_counts(axes[1, 1], df, "AgeCategory", heart_col)

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




```
heart_col = "HadHeartAttack"

def plot_factor_vs_heart(ax, df, factor_col, heart_col):
    grouped = df.groupby([factor_col, heart_col]).size().unstack(fill_value=0)

    # make sure columns are ordered ["No", "Yes"] if both exist
    for v in ["No", "Yes"]:
        if v not in grouped.columns:
            grouped[v] = 0
    grouped = grouped[["No", "Yes"]]
    grouped.plot(kind="bar", ax=ax)
    ax.set_title(f"{factor_col} vs {heart_col}")
    ax.set_ylabel("Number of people")
    ax.tick_params(axis="x", rotation=45)
    ax.legend(["No heart attack", "Heart attack"], title="Heart attack")

factors = ["SmokerStatus", "AlcoholDrinkers", "PhysicalActivities", "SleepHours"]

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

for ax, factor in zip(axes.flatten(), factors):
    plot_factor_vs_heart(ax, df, factor, heart_col)

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

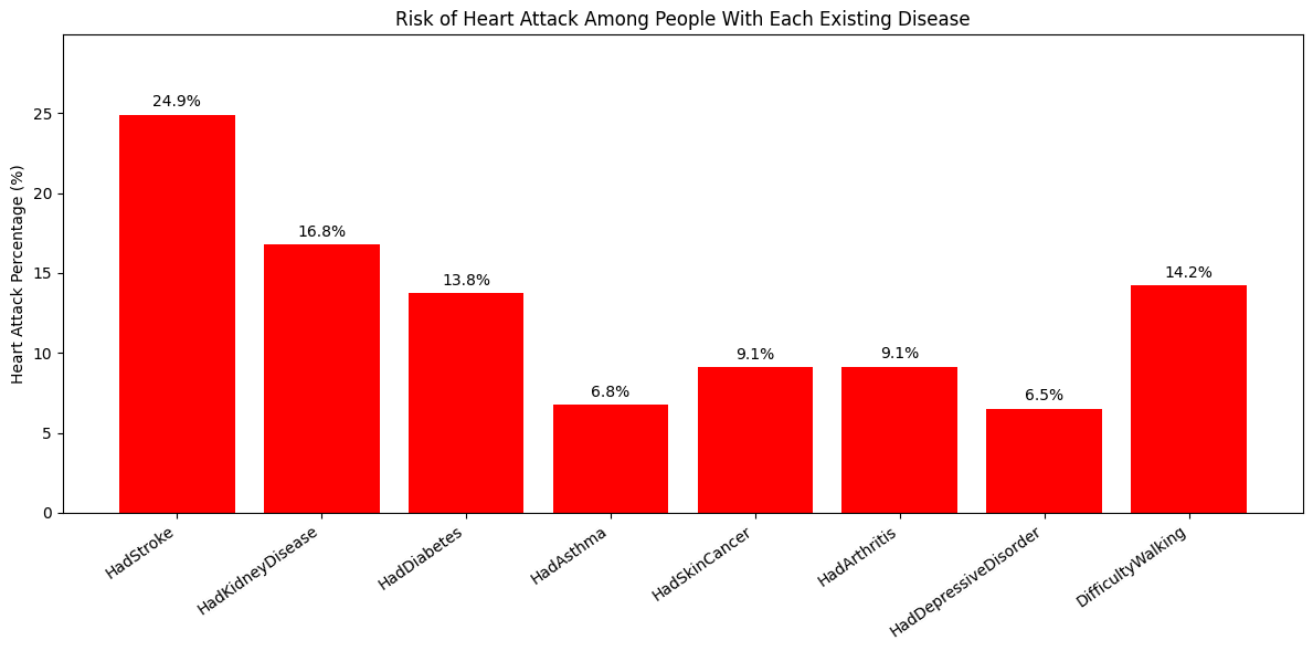



```
# Conditions to compare
conditions = ["HadStroke", "HadKidneyDisease", "HadDiabetes", "HadAsthma", "HadSkinCancer",'HadArthritis',
       'HadDepressiveDisorder','DifficultyWalking']

total_counts = []
heartattack_counts = []

for cond in conditions:
    cond_df = df[df[cond] == 'Yes']                # people with the condition
    total = len(cond_df)
    heart_attack = (cond_df['HadHeartAttack'] == 'Yes').sum()

    total_counts.append(total)
    heartattack_counts.append(heart_attack)

# X locations
x = np.arange(len(conditions))
width = 0.35  # bar width

plt.figure(figsize=(12, 6))

plt.bar(x - width/2, total_counts, width, label='Total with condition')
plt.bar(x + width/2, heartattack_counts, width, label='With heart attack')

plt.xticks(x, conditions, rotation=45, ha='right')
plt.ylabel("Number of people")
plt.title("Heart Attack Counts Among People with Existing Diseases")

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

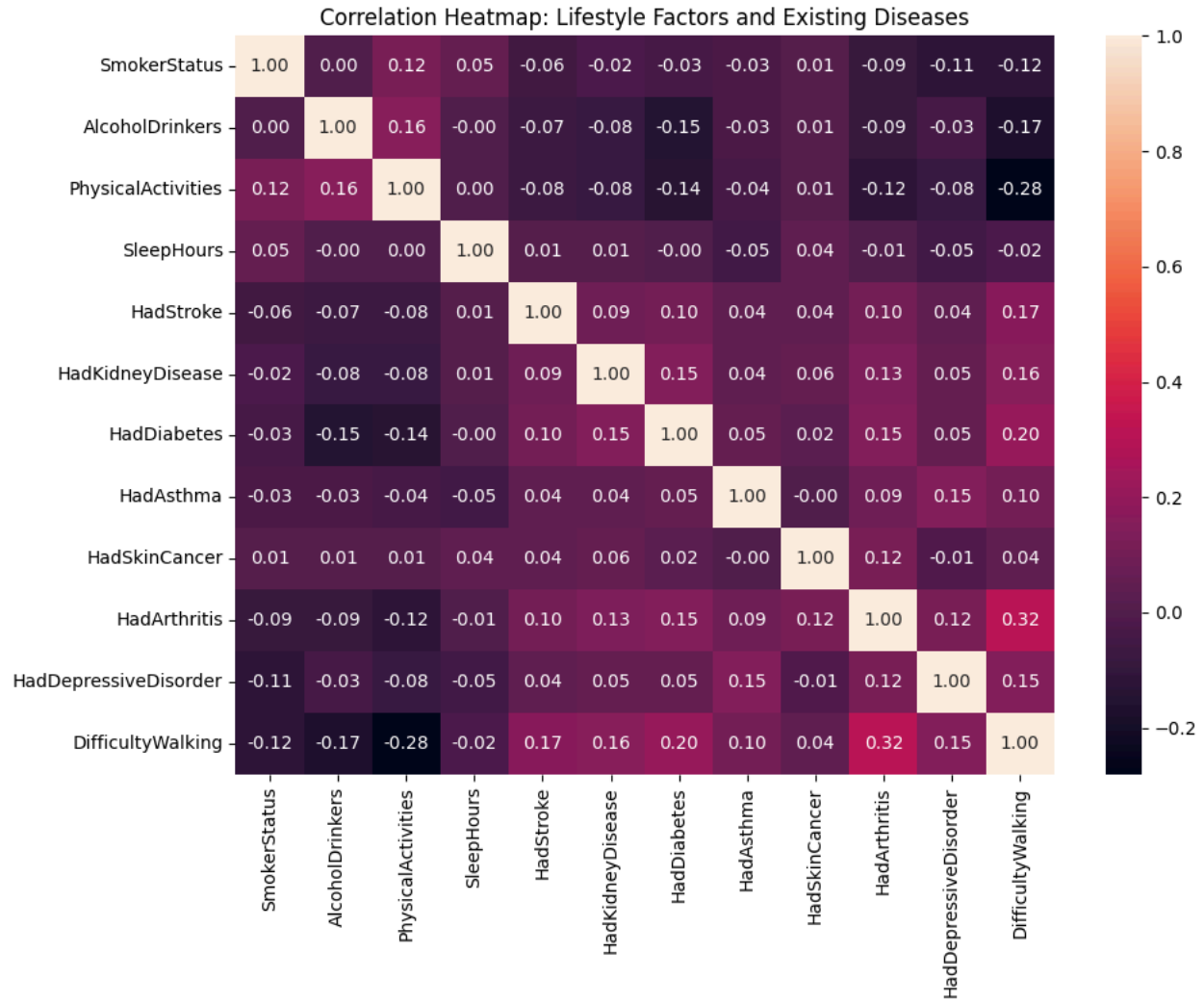



```
import pandas as pd
import matplotlib.pyplot as plt

# Columns for conditions and heart attack
conditions = [
    "HadStroke", "HadKidneyDisease", "HadDiabetes", "HadAsthma",
    "HadSkinCancer", "HadArthritis", "HadDepressiveDisorder",
    "DifficultyWalking"
]

heart_col = "HadHeartAttack"   # adjust if named differently

risk_percentages = []

for cond in conditions:
    # People who have this disease
    cond_yes = df[df[cond] == "Yes"]
    total_with_disease = len(cond_yes)

    if total_with_disease == 0:
        risk = 0
    else:
        heart_attack_count = (cond_yes[heart_col] == "Yes").sum()
        # % heart attack among people who have this disease
        risk = (heart_attack_count / total_with_disease) * 100

    risk_percentages.append(risk)

# ---- Plotting ----

plt.figure(figsize=(12, 6))
plt.bar(conditions, risk_percentages, color="red")

plt.ylabel("Heart Attack Percentage (%)")
plt.title("Risk of Heart Attack Among People With Each Existing Disease")

# Fix overlapping labels
plt.xticks(rotation=35, ha='right')

# Add value labels
for i, v in enumerate(risk_percentages):
    plt.text(i, v + 0.5, f"{v:.1f}%", ha='center')

plt.ylim(0, max(risk_percentages) + 5)
plt.tight_layout()
```

```
plt.show()
```




```
# Columns to include in the correlation heatmap
cols = [
    "SmokerStatus",
    "AlcoholDrinkers",
    "PhysicalActivities",
    "SleepHours",
    "HadStroke",
    "HadKidneyDisease",
    "HadDiabetes",
    "HadAsthma",
    "HadSkinCancer",
    "HadArthritis",
    "HadDepressiveDisorder",
    "DifficultyWalking"
]

# Work on a copy with only these columns
corr_df = df[cols].copy()

# Convert non-numeric columns (e.g. Yes/No, categories) to numeric codes
for c in corr_df.columns:
    if not pd.api.types.is_numeric_dtype(corr_df[c]):
        corr_df[c] = corr_df[c].astype("category").cat.codes

# Compute correlation matrix
corr_matrix = corr_df.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f")
plt.title("Correlation Heatmap: Lifestyle Factors and Existing Diseases")
plt.tight_layout()
plt.show()
```




```
## Step 7: Data Encoding
# Map 'Male' to 1 and 'Female' to 0 for 'Sex' column
df['Sex'] = df['Sex'].map({'Male': 1, 'Female': 0})
df['HadHeartAttack'] = df['HadHeartAttack'].map({'Yes': 1, 'No': 0})

# Ordinal encoding for 'AgeCategory' & GeneralHealth' (assuming ordering based on age groups & health status)
age_order = ['Age 18 to 24', 'Age 25 to 29', 'Age 30 to 34', 'Age 35 to 39', 'Age 40 to 44', 'Age 45 to 49',
             'Age 50 to 54', 'Age 55 to 59','Age 60 to 64', 'Age 65 to 69', 'Age 70 to 74', 'Age 75 to 79',
             'Age 80 or older']
df['AgeCategory'] = df['AgeCategory'].astype('category').cat.set_categories(age_order, ordered=True).cat.codes

health_order = ['Poor', 'Fair', 'Good', 'Very good', 'Excellent']
df['GeneralHealth'] = df['GeneralHealth'].astype('category').cat.set_categories(health_order, ordered=True).cat.code

# One-Hot Encoding for remaining object columns
object_cols = df.select_dtypes(include='object').columns
df = pd.get_dummies(df, columns=object_cols, drop_first=True)
# object_cols list of column names that are categorical, usually with dtype object
#get_dummies turns each categorical col into multiple 0/1 columns (one-hot encoding-eg., Gender_Female Gender_Male)
#Boolean columns (T/F) have dtype = bool, so they are not in object_cols & get_dummies simply never touches them
```

```
df
```

	Sex	GeneralHealth	PhysicalHealthDays	MentalHealthDays	SleepHours	HadHeartAttack	AgeCategory	HeightInMete
342	0	3	4.0	0.0	9.0	0	9	1.
343	1	3	0.0	0.0	6.0	0	10	1.
345	1	3	0.0	0.0	8.0	0	11	1.
346	0	1	5.0	0.0	9.0	0	12	1.
347	0	2	3.0	15.0	5.0	0	12	1.
...	...	...	...	...	...	...	...	
445117	1	3	0.0	0.0	6.0	0	8	1.
445123	0	1	0.0	7.0	7.0	0	1	1.
445124	1	2	0.0	15.0	7.0	0	9	1.
445128	0	4	2.0	2.0	7.0	0	6	1.
445130	1	3	0.0	0.0	5.0	1	10	1.

246013 rows × 108 columns

```
# Select boolean columns & convert to integer (True to 1, False to 0)
boolean_cols = df.select_dtypes(include='bool').columns
for col in boolean_cols:
    df[col] = df[col].astype(int)
```

```
## Step-8: Splitting Dataset- Declare feature vector and target variable
X = df.drop('HadHeartAttack', axis=1)
y = df['HadHeartAttack']

# Split data into separate training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Step-9: Training RF Model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

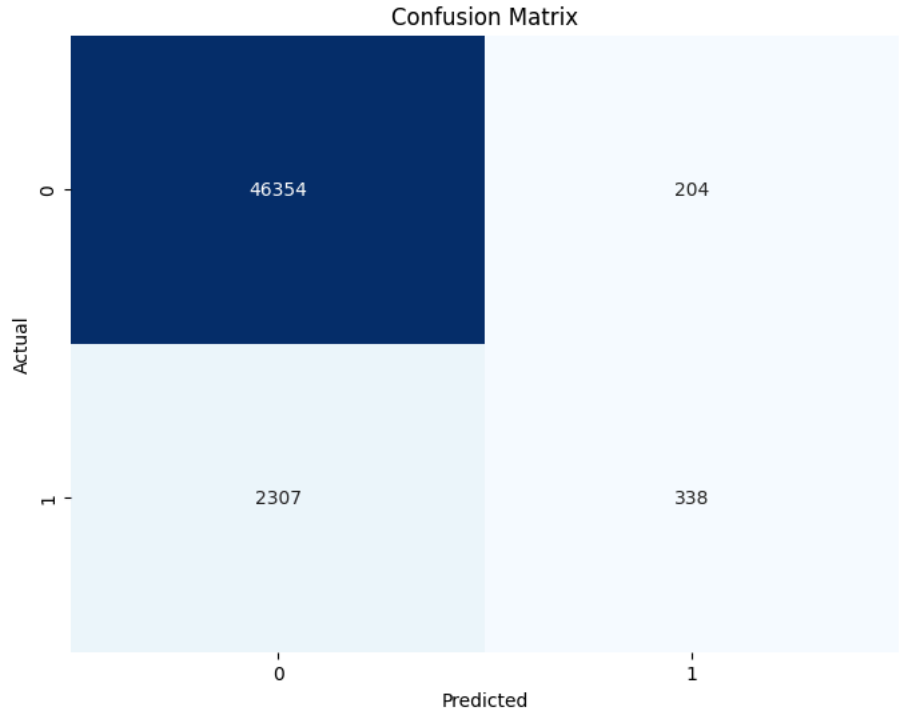
```
# Evaluation Metrics
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy*100:.2f}%") #accuracy is high because the majority class dominates, not because the mode
print(f"Precision: {precision*100:.2f}%") # Precision: 62.36%
print(f"Recall: {recall*100:.2f}%") # Recall: 12.78%, Out of all REAL heart attack cases, our model catches only 12.
# It shows the model:
# Is biased toward predicting No heart attack
# Is failing to learn the minority class
# Suffers heavily from class imbalance
# Our model catches only 12.78% of actual heart attack cases.
print(f"F1 Score: {f1*100:.2f}%")
```

```
Accuracy: 94.90%
Precision: 62.36%
Recall: 12.78%
F1 Score: 21.21%
```

```
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     46558
           1       0.62      0.13      0.21      2645

    accuracy                           0.95     49203
   macro avg       0.79      0.56      0.59     49203
weighted avg       0.93      0.95      0.93     49203
```

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
from imblearn.over_sampling import SMOTE
## Splitting Dataset and SMOTE
X = df.drop('HadHeartAttack', axis=1)
y = df['HadHeartAttack']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Apply SMOTE to the testing data
X_test_smote, y_test_smote = smote.fit_resample(X_test, y_test)
##Training RF Model SMOTE
model_smote = RandomForestClassifier(random_state=42)
model_smote.fit(X_train_smote, y_train_smote)


## Evaluation Metrics using SMOTE Data
y_pred_smote = model_smote.predict(X_test_smote)

accuracy_smote = accuracy_score(y_test_smote, y_pred_smote)
precision_smote = precision_score(y_test_smote, y_pred_smote)
recall_smote = recall_score(y_test_smote, y_pred_smote)
f1_smote = f1_score(y_test_smote, y_pred_smote)
```

```
print("\nEvaluation with SMOTE:")
print(f"Accuracy: {accuracy_smote*100:.2f}%")
print(f"Precision: {precision_smote*100:.2f}%")
print(f"Recall: {recall_smote*100:.2f}%")
```

```
Evaluation with SMOTE:
Accuracy: 91.06%
Precision: 97.50%
```