Deliverable-3

Choose Machine Learning Framework:

The final approach involved using **Amazon SageMaker** to execute different models for the chosen dataset and retrieve performance results of various algorithms.

Develop and Train Models:

Loading Data:

Loading a cleaned and well defined heart health dataset from previous iterations that is stored in an Amazon S3 bucket into a Pandas DataFrame for analysis.

[8]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore", category=Warning)
```

[9]:
```
bucket = 'group15awsprojectbucket'


file_key = 'datafolder/heart_2020_cleaned.csv'

# Construct the full S3 path
data_location = 's3://{}/{}'.format(bucket, file_key)

# Read the data into a Pandas DataFrame
df = pd.read_csv(data_location)


df.head()
```

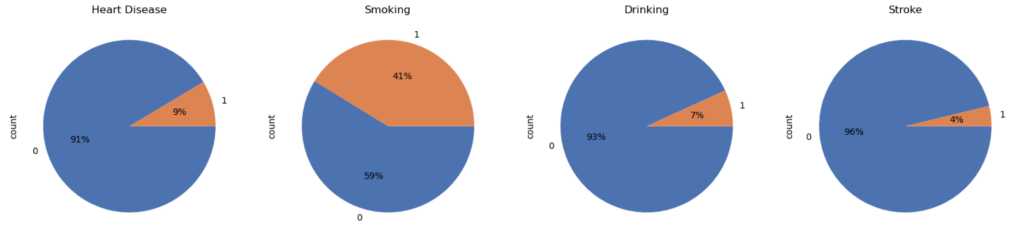
[9]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	Physica
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	

Data Preprocessing and Visualization for Health Attributes:

Visually represent the distribution of categorical data related to health factors ('Heart Disease', 'Smoking', 'Drinking', 'Stroke') using pie charts. Each pie chart represents the proportion of different categories within each factor in the dataset.

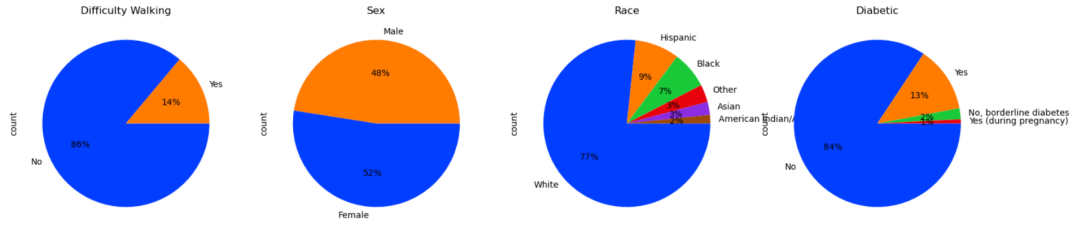
[25]:
```
plt.figure(figsize=(20,20))
colors = sns.color_palette('deep')

plt.subplot(1, 4, 1)
df['HeartDisease'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Heart Disease')

plt.subplot(1, 4, 2)
df['Smoking'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Smoking')

plt.subplot(1, 4, 3)
df['AlcoholDrinking'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Drinking')

plt.subplot(1, 4, 4)
df['Stroke'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Stroke')

plt.show()
```



Visual representation of the distribution of categorical data related to health factors ('DiffWalking', 'Sex', 'Race', 'Diabetic") using pie charts. Each pie chart represents the proportion of different categories within each factor in the dataset.

[21]:
```
df['HeartDisease'] = df['HeartDisease'].replace(['No', 'Yes'], [0, 1])
df['Smoking'] = df['Smoking'].replace(['No', 'Yes'], [0, 1])
df['AlcoholDrinking'] = df['AlcoholDrinking'].replace(['No', 'Yes'], [0, 1])
df['Stroke'] = df['Stroke'].replace(['No', 'Yes'], [0, 1])
```

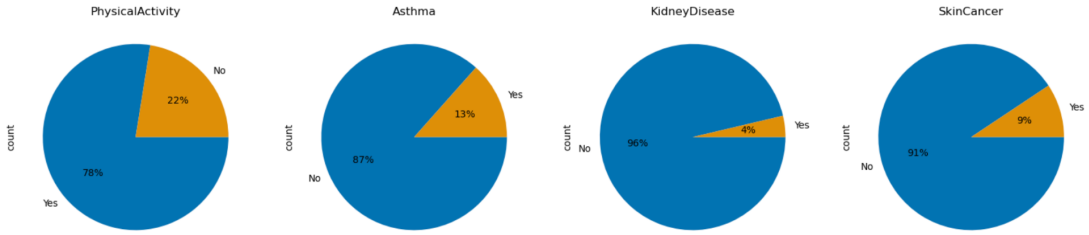
[23]:
```
plt.figure(figsize=(20,20))
colors = sns.color_palette('bright')

plt.subplot(1, 4, 1)
df['DiffWalking'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Difficulty Walking')

plt.subplot(1, 4, 2)
df['Sex'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Sex')

plt.subplot(1, 4, 3)
df['Race'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Race')

plt.subplot(1, 4, 4)
df['Diabetic'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
plt.title('Diabetic')

plt.show()
```



Visual representation of the distribution of categorical data related to health factors ('PhysicalActivity', 'Asthma', 'KidneyDisease', 'SkinCancer') using pie charts. Each pie chart represents the proportion of different categories within each factor in the dataset.

```
[27]: df['DiffWalking'] = df['DiffWalking'].replace(['No', 'Yes'], [0, 1])
      df['Sex'] = df['Sex'].replace(['Female', 'Male'], [0, 1])
```

```
[31]: plt.figure(figsize=(20,20))
      colors = sns.color_palette('colorblind')

      plt.subplot(1, 4, 1)
      df['PhysicalActivity'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
      plt.title('PhysicalActivity')

      plt.subplot(1, 4, 2)
      df['Asthma'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
      plt.title('Asthma')

      plt.subplot(1, 4, 3)
      df['KidneyDisease'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
      plt.title('KidneyDisease')

      plt.subplot(1, 4, 4)
      df['SkinCancer'].value_counts().plot.pie(counterclock=False, autopct='%.0f%%', colors = colors)
      plt.title('SkinCancer')

      plt.show()
```



EDA: Exploring Heart Disease Correlations with Demographics and Health Factors

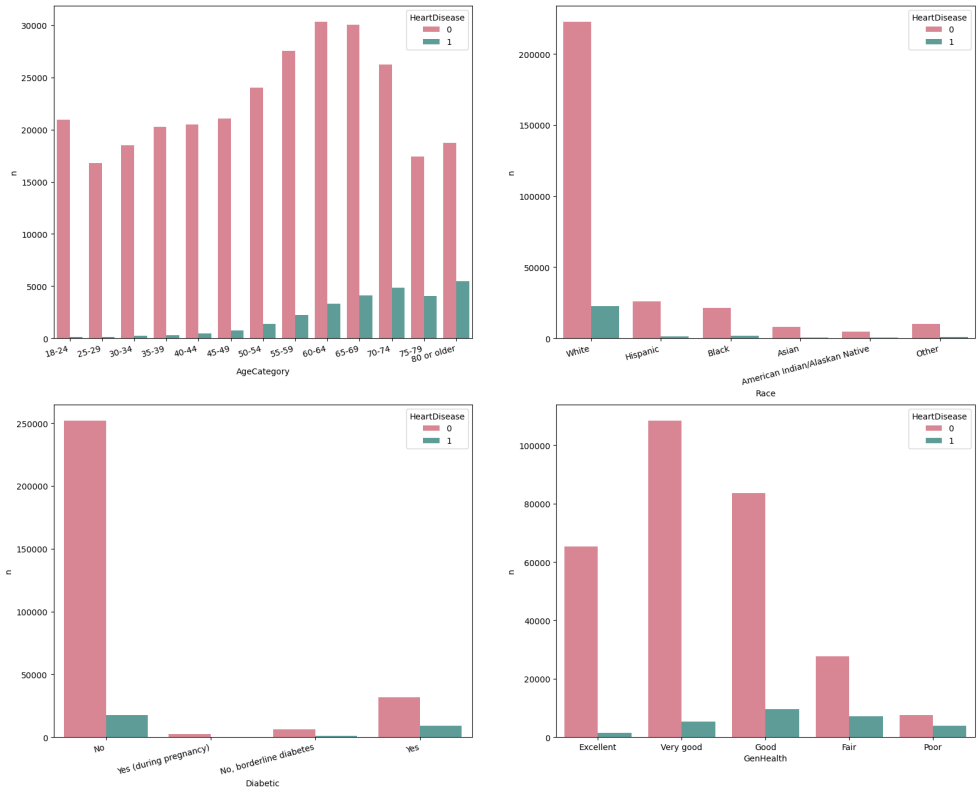
It performs data preparation, aggregation, and visualization to explore how heart disease is related to demographic factors ('AgeCategory', 'Race') and health-related variables ('Diabetic', 'GenHealth') in the dataset.

```
[33]: df['PhysicalActivity'] = df['PhysicalActivity'].replace(['No', 'Yes'], [0, 1]) # do you play any sports?
      df['Asthma'] = df['Asthma'].replace(['No', 'Yes'], [0, 1])
      df['KidneyDisease'] = df['KidneyDisease'].replace(['No', 'Yes'], [0, 1])
      df['SkinCancer'] = df['SkinCancer'].replace(['No', 'Yes'], [0, 1])
```

```
[34]: HeartDisease_AgeCategory = df.groupby(['AgeCategory', 'HeartDisease'], as_index=False).agg(n = ('AgeCategory', 'count'))
      HeartDisease_Race = df.groupby(['Race', 'HeartDisease'], as_index=False).agg(n = ('Race', 'count'))
      HeartDisease_Diabetic = df.groupby(['Diabetic', 'HeartDisease'], as_index=False).agg(n = ('Diabetic', 'count'))
      HeartDisease_GenHealth = df.groupby(['GenHealth', 'HeartDisease'], as_index=False).agg(n = ('GenHealth', 'count'))
```

Grouping according to variables

```
plt.figure(figsize=(20, 16))

plt.subplot(2, 2, 1)
plt.xticks(rotation=15, ha='right')
sns.barplot(data=HeartDisease_AgeCategory, x='AgeCategory', y='n', hue="HeartDisease", palette='husl')

plt.subplot(2, 2, 2)
plt.xticks(rotation=15, ha='right')
sns.barplot(data=HeartDisease_Race, x='Race', y='n', hue="HeartDisease", palette='husl',
            order=['White', 'Hispanic', 'Black', 'Asian', 'American Indian/Alaskan Native', 'Other'])

plt.subplot(2, 2, 3)
plt.xticks(rotation=15, ha='right')
sns.barplot(data=HeartDisease_Diabetic, x='Diabetic', y='n', hue="HeartDisease",
            palette='husl', order=['No', 'Yes (during pregnancy)', 'No, borderline diabetes', 'Yes'])

plt.subplot(2, 2, 4)
sns.barplot(data=HeartDisease_GenHealth, x='GenHealth', y='n', hue="HeartDisease",
            palette='husl', order=['Excellent', 'Very good', 'Good', 'Fair', 'Poor'])

plt.show()
```



Bar Plot representing heart disease distribution in accordance to each variable

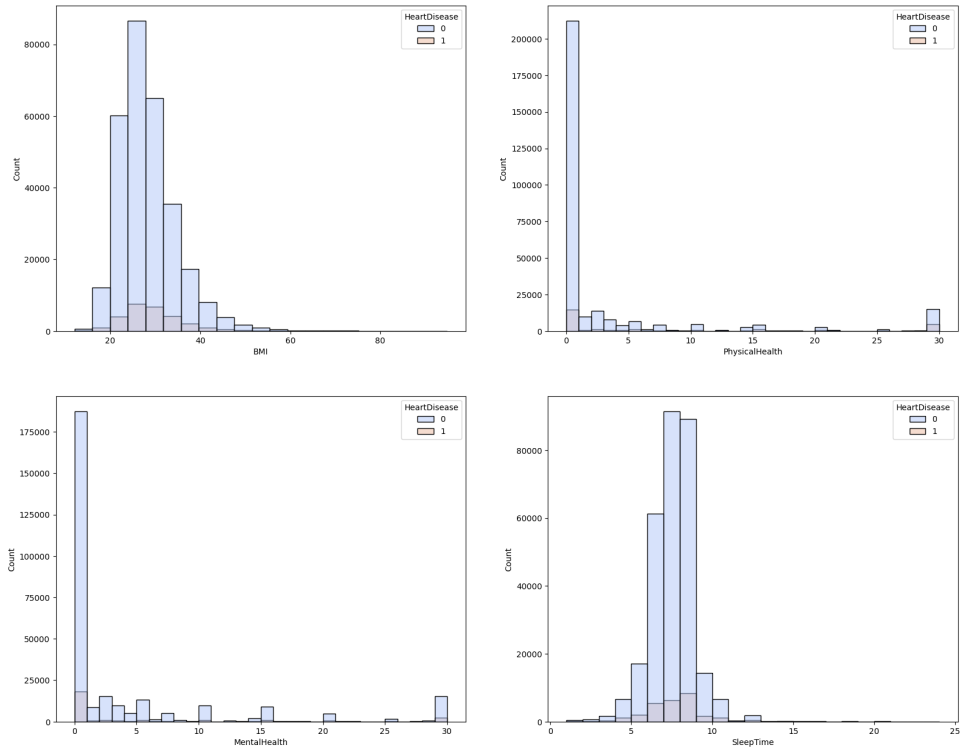
Comparative Analysis of Health Metrics between Heart Disease Groups

It shows a side-by-side comparison of the distributions of these health-related metrics, allowing for a visual assessment of any potential differences or patterns between the groups with respect to heart disease status.

```
df['AgeCategory'] = df['AgeCategory'].replace(['18-24', '25-29', '30-34', '35-39', '40-44', '25-29',
                                               '30-34', '35-40', '41-44', '45-49', '50-54', '55-59',
                                               '60-64', '65-69', '70-74', '75-79', '80 or older'],
                                              [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
df['Race'] = df['Race'].replace(['White', 'Hispanic', 'Black', 'Asian', 'American Indian/Alaskan Native', 'Other'],
                                [0, 1, 2, 3, 4, 5])
df['Diabetic'] = df['Diabetic'].replace(['No', 'Yes (during pregnancy)', 'No, borderline diabetes', 'Yes'],
                                        [0, 1, 2, 3])
df['GenHealth'] = df['GenHealth'].replace(['Excellent', 'Very good', 'Good', 'Fair','Poor'],
                                          [0, 1, 2, 3, 4])
```

```
df_num = df[['HeartDisease', 'BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime']]
print(df_num.shape)
df_num.head()

plt.figure(figsize=(20,16))

plt.subplot(2, 2, 1)
sns.histplot(df_num, x='BMI', hue='HeartDisease', binwidth=4, palette='coolwarm')

plt.subplot(2, 2, 2)
sns.histplot(df_num, x='PhysicalHealth', hue='HeartDisease', binwidth=1, palette='coolwarm')

plt.subplot(2, 2, 3)
sns.histplot(df_num, x='MentalHealth', hue='HeartDisease', binwidth=1, palette='coolwarm')

plt.subplot(2, 2, 4)
sns.histplot(df_num, x='SleepTime', hue='HeartDisease', binwidth=1, palette='coolwarm')

plt.show()
```

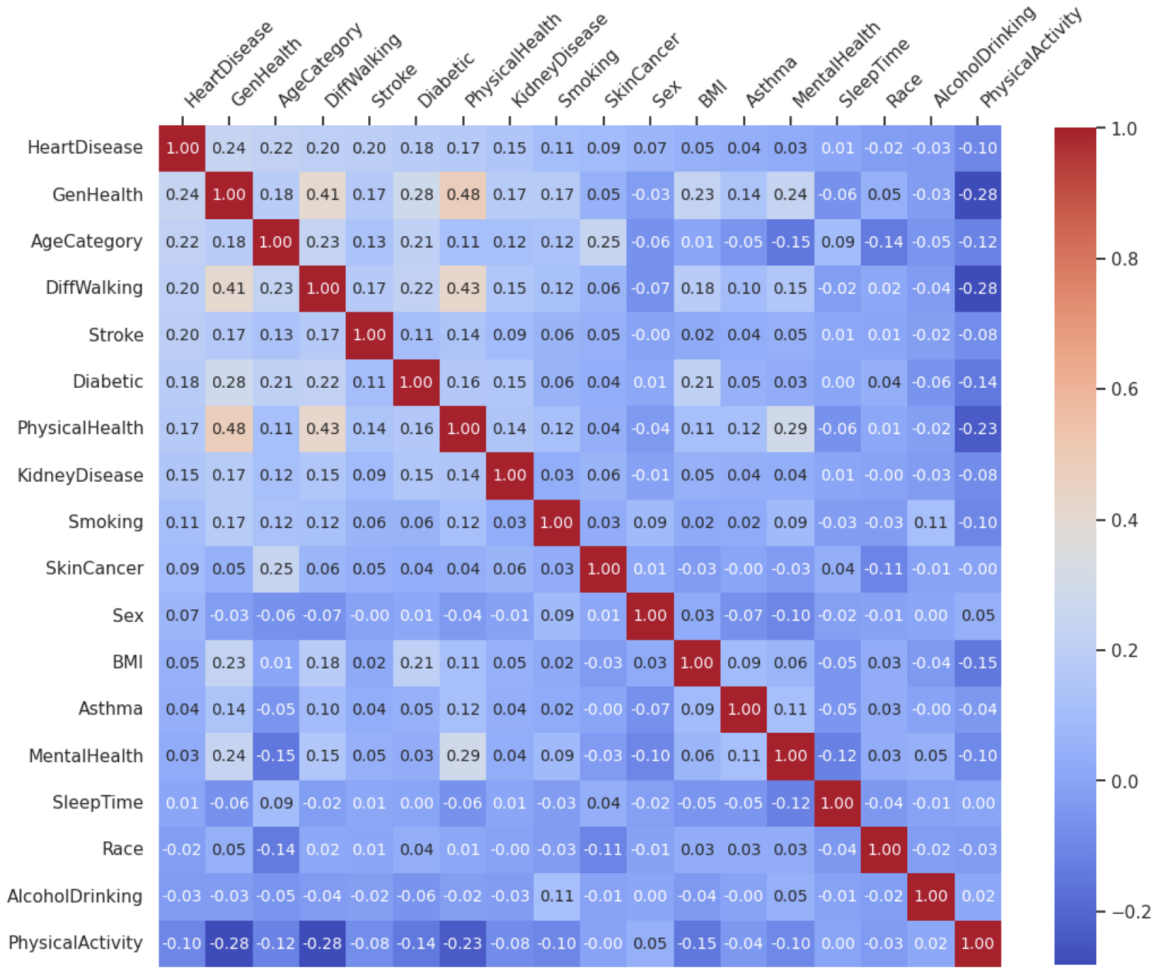
```
(319795, 5)
```



Correlation Heatmap of Top Features Related to Heart Disease:

It generates a correlation heatmap to visualize the relationships between the top features associated with heart disease in a dataset.

```
plt.figure(figsize=(12, 12))
k = 18
cols = corrmat.nlargest(k, 'HeartDisease')['HeartDisease'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10},
                 yticklabels=cols.values, xticklabels=cols.values, cmap="coolwarm", cbar_kws={"shrink": 0.8})
hm.xaxis.tick_top()
plt.xticks(rotation=45, ha='left')
plt.show()
```



SMOTE Analysis:

SMOTE (Synthetic Minority Over-sampling Technique) is an approach to address class imbalance in machine learning datasets.Class imbalance can lead to biased models that predominantly predict the majority class. By creating synthetic examples of the minority class, SMOTE helps to balance the class distribution. This allows the model to learn more about the minority class's properties and improves the model's ability to correctly identify instances of this class.When a model is trained on imbalanced data, it might overfit to the majority class. SMOTE reduces this risk by providing more examples of the minority class, which can lead to a more generalized.

In healthcare, it's crucial to minimize false negatives (failing to detect the disease when it is present). By giving the model more examples of the minority class, we're essentially aiding it in learning to detect more nuanced patterns associated with the disease, which could be vital for patient outcomes.

Feature-Target Relationship Visualization after SMOTE:

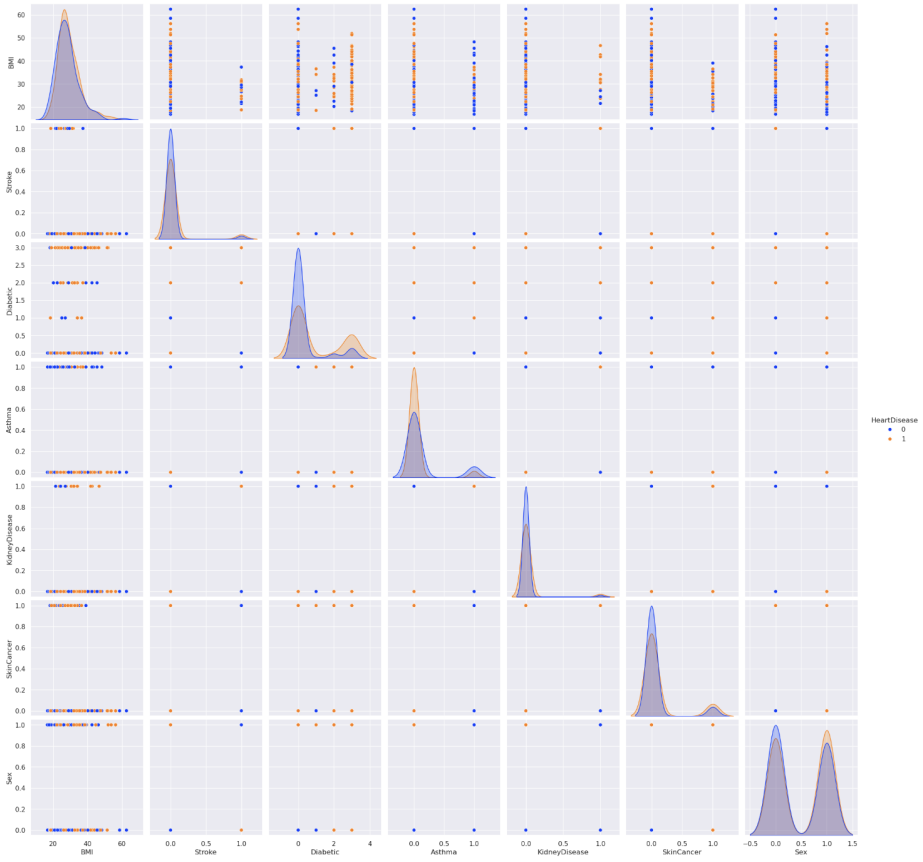
This below code demonstrates the process of balancing a dataset concerning heart disease occurrences using the SMOTE (Synthetic Minority Oversampling Technique) and visually analyzing the relationships between specific health-related features and heart disease occurrence.

```
target = df['HeartDisease']
feature = df.drop(['HeartDisease'], axis=1)

from imblearn.over_sampling import SMOTE
oversample = SMOTE()

feature_balanced, target_balanced = oversample.fit_resample(feature, target)
target_balanced.value_counts()
```

```
HeartDisease
0    292422
1    292422
Name: count, dtype: int64
```

Now the data is uniformly distributed, 0- no heart disease and 1- presence of heart disease

```
balanced_df_2 = balanced_df[['HeartDisease', 'BMI', 'Stroke', 'Diabetic',
                             'Asthma', 'KidneyDisease', 'SkinCancer', 'Sex']]

balanced_2_sample = balanced_df_2.sample(500)

sns.pairplot(balanced_2_sample, hue = "HeartDisease", size = 3, palette = 'bright')
plt.show()
```



Data Splitting:

It ensures data readiness by splitting it into train-test sets for validation, standardizing the features to have similar scales, and making it suitable for training predictive models.

```
[58]: print(feature.shape)
      target.value_counts()

      (319795, 17)
[58]: HeartDisease
      0    292422
      1     27373
      Name: count, dtype: int64

[59]: print(feature_balanced.shape)
      target_balanced.value_counts()

      (584844, 17)
[59]: HeartDisease
      0    292422
      1    292422
      Name: count, dtype: int64

[60]: from sklearn.model_selection import train_test_split, KFold

      kf = KFold(n_splits = 5, shuffle = True, random_state = 2309)
      for tr_idx, te_idx in kf.split(feature):
          X_train, X_test = feature.iloc[tr_idx], feature.iloc[te_idx]
          y_train, y_test = target.iloc[tr_idx], target.iloc[te_idx]
      ------
      X_train.shape, X_test.shape, y_train.shape, y_test.shape

[60]: ((255836, 17), (63959, 17), (255836,), (63959,))
```

Feature Scaling:

Feature scaling is a method used to standardize the range of independent variables or features of data. In our code, we're using StandardScaler from Scikit-learn, which standardizes features by removing the mean and scaling to unit variance.It's a critical preprocessing step.

Process:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(data=X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(data=X_test_scaled, columns=X_test.columns)
X_test_scaled.describe()
```

	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	
count	63959.000000	63959.000000	63959.000000	63959.000000	63959.000000	63959.000000	63959.000000	63959.000000	63959.000000	6
mean	0.002419	-0.005898	-0.002511	-0.003358	0.002695	0.003344	0.000509	0.006066	-0.007145	
std	0.999743	0.998948	0.995694	0.991838	1.002877	1.003092	1.000539	1.000297	1.003752	
min	-2.555261	-0.838895	-0.270589	-0.198390	-0.423775	-0.489673	-0.401519	-0.950556	-2.087794	
25%	-0.675273	-0.838895	-0.270589	-0.198390	-0.423775	-0.489673	-0.401519	-0.950556	-1.015806	
50%	-0.154540	-0.838895	-0.270589	-0.198390	-0.423775	-0.489673	-0.401519	-0.950556	0.270581	
75%	0.484970	1.192044	-0.270589	-0.198390	-0.172084	-0.112329	-0.401519	1.052016	0.699376	
max	10.436321	1.192044	3.695636	5.040587	3.351583	3.283767	2.490541	1.052016	1.342569	

Algorithms Employed:

XGBoost,Random Forest,Decision Trees

XGBoost Classifier:

The below code illustrates the process of training an XGBoost Classifier, making predictions, and evaluating its performance using various metrics to assess its accuracy and effectiveness in predicting the target variable ('HeartDisease' in this case) based on the provided features.

```
[64]: import xgboost as xgb
      from xgboost import plot_importance

      xgb_class = xgb.XGBClassifier(random_state = 2309)
      xgb_class.fit(X_train_scaled, y_train)

      xgb_class.score(X_train_scaled, y_train)

[64]: 0.9220203567910693

[65]: pred = xgb_class.predict(X_test_scaled)
      pred[:20]

[65]: array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0])

[66]: y_test[:20].array

[66]: <NumpyExtensionArray>
      [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
      Length: 20, dtype: int64

[67]: xgb_class.score(X_test_scaled, y_test)

[67]: 0.9160712331337263

[68]: from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

      print(confusion_matrix(y_test, pred))

      [[58048   468]
       [ 4900   543]]

[70]: print(classification_report(y_test, pred))

                    precision    recall  f1-score   support

                 0       0.92      0.99      0.96     58516
                 1       0.54      0.10      0.17      5443

          accuracy                           0.92     63959
         macro avg       0.73      0.55      0.56     63959
      weighted avg       0.89      0.92      0.89     63959
```

Random Forest Classifier:

The below code demonstrates the process of implementing a Random Forest Classifier, training it on the provided data, making predictions, and assessing its performance using evaluation metrics like accuracy, confusion matrix, and classification report.

```
[71]: from sklearn.ensemble import RandomForestClassifier

[73]: rf = RandomForestClassifier(n_estimators= 50 , max_depth= 5 , max_features=12)

[77]: rf.fit(X_train_scaled , y_train);

[81]: print (rf.score(X_test_scaled, y_test))
      0.9161337731984552

[83]: pred = rf.predict(X_test_scaled)
      pred[:20]

[83]: array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

[92]: print(confusion_matrix(y_test, pred))
      [[58292   224]
       [ 5141   302]]

[93]: print(classification_report(y_test, pred))
                    precision    recall  f1-score   support

                 0       0.92      1.00      0.96     58516
                 1       0.57      0.06      0.10      5443

          accuracy                           0.92     63959
         macro avg       0.75      0.53      0.53     63959
      weighted avg       0.89      0.92      0.88     63959
```

Decision Tree Classifier: Initial & Optimized Model Evaluation

The below code showcases the evaluation of an initial Decision Tree model, the process of hyperparameter tuning using Grid Search, and the evaluation of an optimized Decision Tree model with the best hyperparameters found.

Hyperparameter Tuning:

Hyperparameter tuning involves adjusting the parameters of the model to improve performance. It is the process of finding the most optimal parameters for a machine learning model. These parameters, known as hyperparameters, are set prior to the learning process and govern the behavior of the learning algorithm.

In SageMaker, this can be done using its hyperparameter optimization (HPO) feature. To implement hyperparameter tuning,the steps are:

- Hyperparameter tuningDefine a range of values for each hyperparameter.
- Use a method like grid search, random search, or Bayesian optimization (as provided by SageMaker) to explore different combinations of these hyperparameters.
- Evaluate model performance for each combination to find the best set of hyperparameters.

[115]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
```

[116]:
```
dt = DecisionTreeClassifier(max_depth=5, max_features=18)
dt.fit(X_train_scaled, y_train)
print("Initial Model Score:", dt.score(X_test_scaled, y_test))
```

```
Initial Model Score: 0.9161181381822731
```

[117]:
```
pred = dt.predict(X_test_scaled)
print("Confusion Matrix:\n", confusion_matrix(y_test, pred))
print("Classification Report:\n", classification_report(y_test, pred))
```

```
Confusion Matrix:
 [[58292   224]
 [ 5141   302]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     58516
           1       0.57      0.06      0.10      5443

    accuracy                           0.92     63959
   macro avg       0.75      0.53      0.53     63959
weighted avg       0.89      0.92      0.88     63959
```

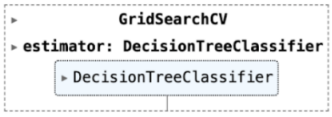
Results before Hyper Parameter Tuning

[118]:
```
# parameter grid for hyperparameter tuning
param_grid_dt = {
    'max_depth': [3, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}
```

[119]:
```
# Initialize Grid Search
grid_search_dt = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=param_grid_dt, cv=5, scoring='accuracy')
```

[120]:
```
# Fit grid search
grid_search_dt.fit(X_train_scaled, y_train)
```

[120]:



[121]:
```
# Best parameters
best_params = grid_search_dt.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_samples_split': 10}
```

Hyper Parameter Tuning

[122]:
```
# Train and evaluate the decision tree with the best parameters
dt_optimized = DecisionTreeClassifier(**best_params)
dt_optimized.fit(X_train_scaled, y_train)

print("Optimized Model Score:", dt_optimized.score(X_test_scaled, y_test))
```

```
Optimized Model Score: 0.9151644021951563
```

Results after Hyper Parameter Tuning

The model performed well without hyper tuning.And hyper tuning was not applied to XGBoost as it requires a lot of execution time and is time consuming.

Evaluation and Validation:

The metrics chosen for evaluation is:

ROC Curve Analysis for Multiple Models

Choosing ROC (Receiver Operating Characteristic) curves and AUC (Area Under the Curve) scores as evaluation metrics, especially for classification problems, is grounded in several strong reasons:

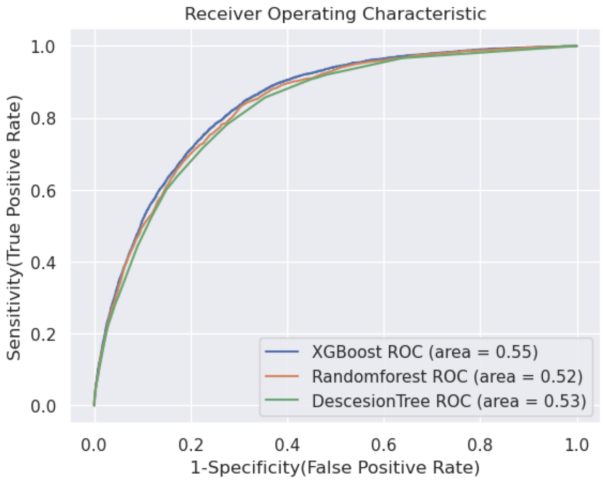
- Performance at Various Thresholds:The ROC curve provides a comprehensive view of a model's performance across a range of classification thresholds. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR), allowing us to evaluate the trade-offs between correctly predicting the positive class and incorrectly predicting the negative class as positive.
- Imbalanced Classes:
  ROC curves are useful in this situation as we have imbalanced classes. In such cases, metrics like accuracy can be misleading, but the ROC curve remains a reliable indicator of the model's ability to distinguish between classes.
- Comparability:
  We have multiple models, ROC curves offer a clear visual tool to compare their performance. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.

AUC as a Single Metric:
The AUC provides a single number summary of the model performance.
Robustness to Cutoff Changes:
Unlike accuracy, precision, or recall, which depend on a specific classification threshold, the AUC provides a measure that is independent of any particular threshold. This is particularly useful when we do not have a clear idea of where to set the threshold, or when the cost/seriousness of false positives and false negatives varies.

- Versatility:
  ROC and AUC are applicable to a wide range of classification problems and are not sensitive to the distribution of the test dataset, making them versatile tools for model evaluation.

ROC and AUC are chosen for their ability to provide a comprehensive, comparable, and threshold-independent assessment of a model's performance, especially in scenarios with imbalanced classes or when comparing multiple models.

[101]:
```
for m in models:
    model = m['model']
    y_pred=model.predict(X_test_scaled)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test_scaled)[:,1])
        # Calculate Area under the curve to display on the plot
    auc = metrics.roc_auc_score(y_test,model.predict(X_test_scaled))
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



XGBoost Classifier Training and Evaluation with Balanced Data

This helps in understanding the behavior and performance of an XGBoost Classifier trained on balanced data by assessing its predictive accuracy, identifying important features, and visualizing their importance in the model.

[105]:
```
import xgboost as xgb
from xgboost import plot_importance

xgb_class = xgb.XGBClassifier(random_state = 2309)
xgb_class.fit(X_train_balanced_scaled, y_train_balanced)

xgb_class.score(X_train_balanced_scaled, y_train_balanced)
```

[105]:
```
0.8458993408509947
```

[106]:
```
pred_balanced = xgb_class.predict(X_test_balanced_scaled)
pred_balanced[:20]
```

[106]:
```
array([0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0])
```

[107]:
```
y_test_balanced[:20].array
```

[107]:
```
<NumpyExtensionArray>
[0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
Length: 20, dtype: int64
```

[108]:
```
xgb_class.score(X_test_balanced_scaled, y_test_balanced)
```

[108]:
```
0.8412471787155461
```

[109]:
```
print(confusion_matrix(y_test_balanced, pred_balanced))
```

```
[[49257  9106]
 [ 9463 49142]]
```

[110]:
```
print(classification_report(y_test_balanced, pred_balanced))
```

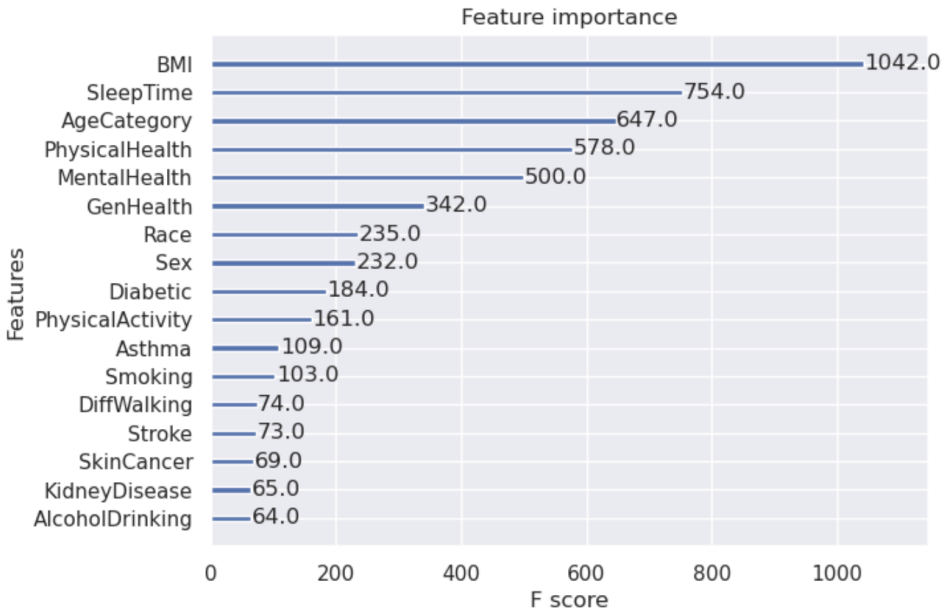
```
              precision    recall  f1-score   support

           0       0.84      0.84      0.84     58363
           1       0.84      0.84      0.84     58605

    accuracy                           0.84    116968
   macro avg       0.84      0.84      0.84    116968
weighted avg       0.84      0.84      0.84    116968
```

Understanding the feature importance to understand the major cause of heart diseases:

[111]:
```
for col , val in sorted(
zip(X_train_balanced_scaled.columns, xgb_class.feature_importances_ ),
    key = lambda x : x[1], reverse=True)[:5] : print(f"{col:10}{val:10.3f}")
```

```
AgeCategory     0.180
SleepTime      0.146
AlcoholDrinking     0.123
GenHealth      0.105
PhysicalActivity     0.073
```

[112]:
```
fig, ax = plt.subplots(figsize=(7,5))
xgb.plot_importance(xgb_class, ax=ax)
plt.show()
```



The results show that BMI,sleep time,age category and Physical health play a crucial role in deciding the heart health.