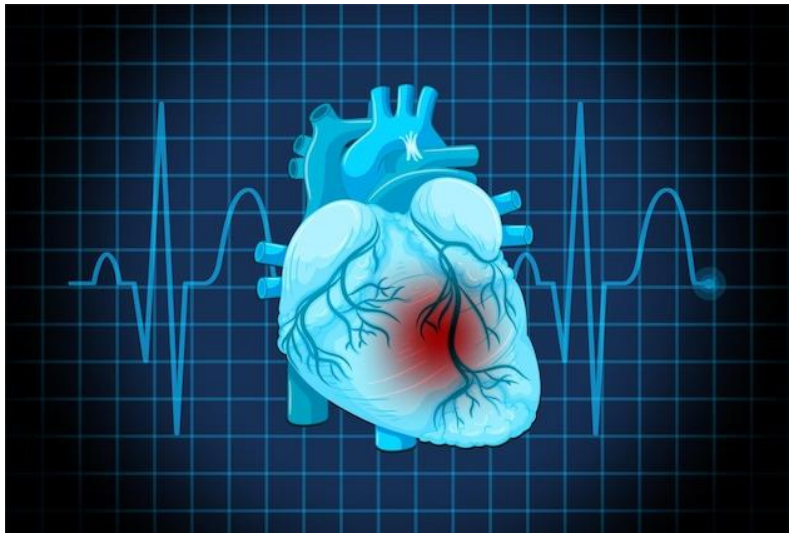


CARDIOVASCULAR DISEASE RISK PREDICTION

AML Project Deliverable 2

Avraham Kaminker, Blanca Herreros de Tejada Lobo, Eldar Djangirov, Rashmi Chelliah, Yuang Fan

DATASET USED

CDC Behavioral Risk Factor Surveillance System (BRFSS) 2021

303 columns

438,693 rows

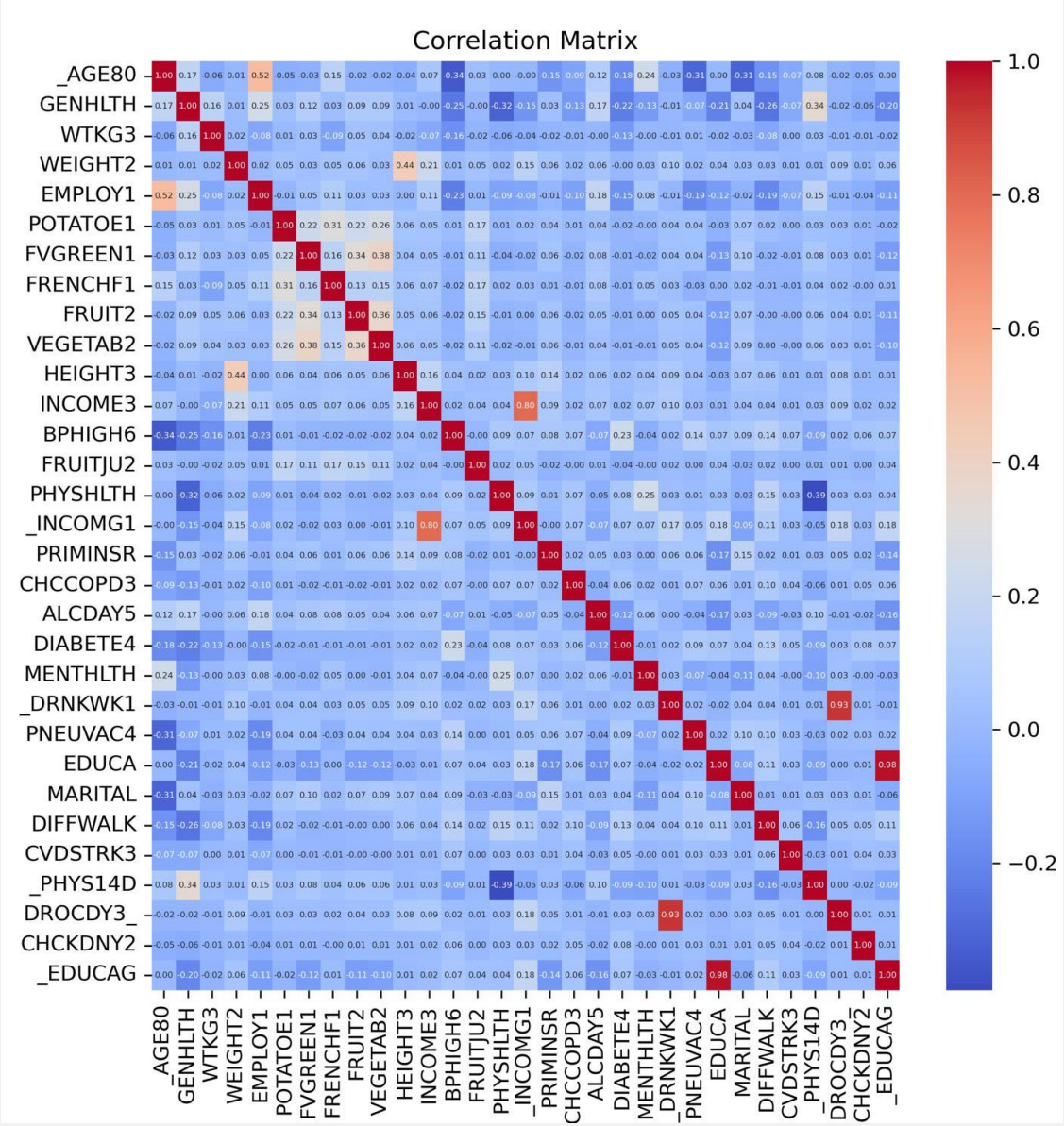
Goal: build a high performing model that predicts heart disease with high recall, and find the most predictive features of heart disease

INITIAL DATA CLEANING

Target Variable	CVDCRHD4 (Ever Diagnosed with Angina or Coronary Heart Disease)
Drop Columns	With >10% of data missing
Remove	Obviously irrelevant survey specific features
Apply	Imputer mean filling strategy for NaN values
Map	CVDCRHD4 to 0/1 (no/yes) values

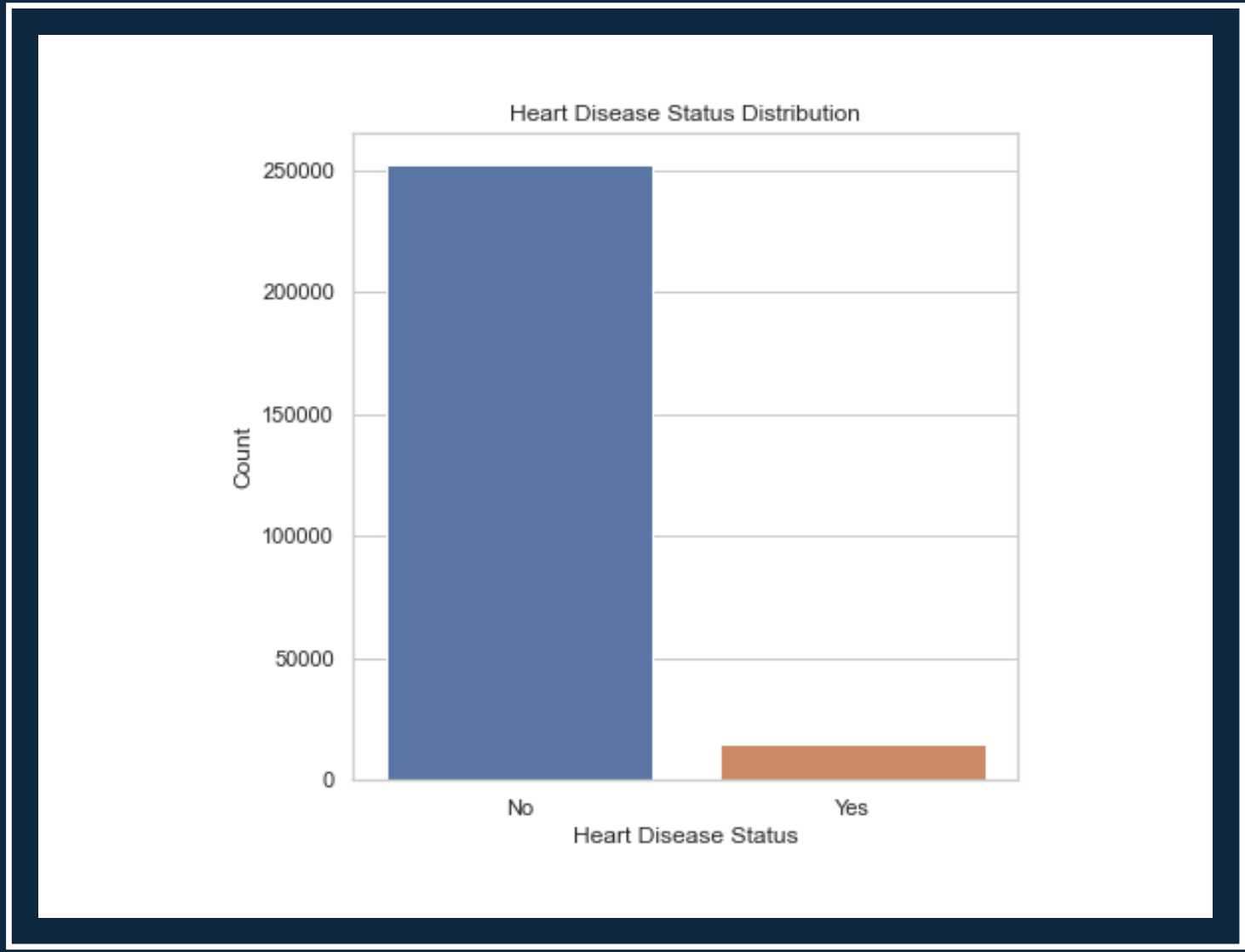
FURTHER FEATURE SELECTION

- Use RandomForestClassifier to find 45 most relevant features
- Manually remove highly correlated, redundant, or (other) survey methodology features
- Remaining features: 27



CVDCRHD4 VALUE DISTRIBUTION

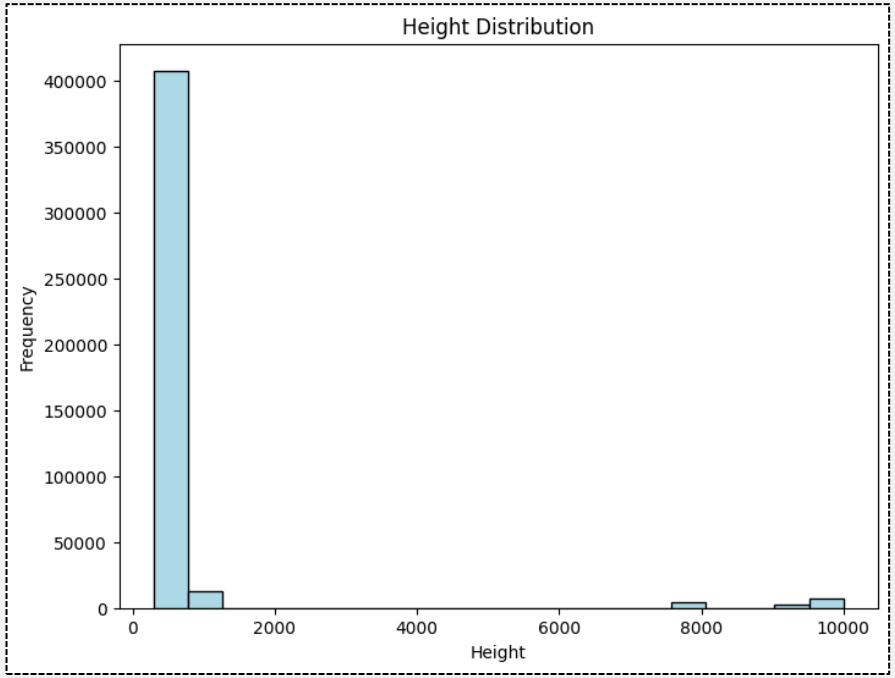
Will employ techniques (SMOTE, stratified) for minority class



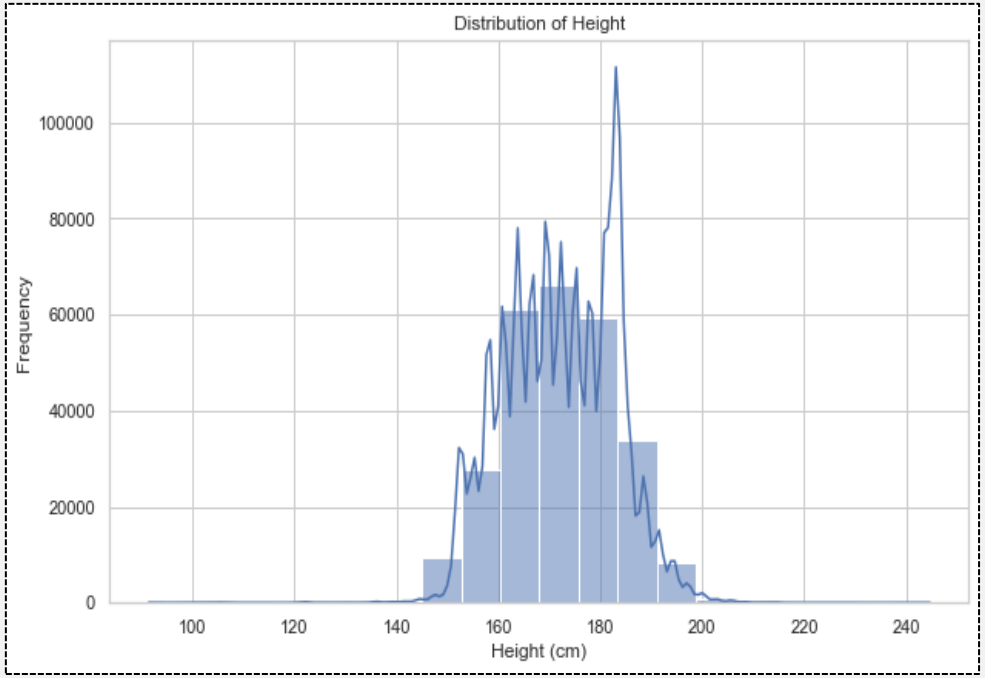
DATA CLEANING EXAMPLE #1

CONVERTING HEIGHT3 VALUES TO CM

BEFORE DATA CLEANING



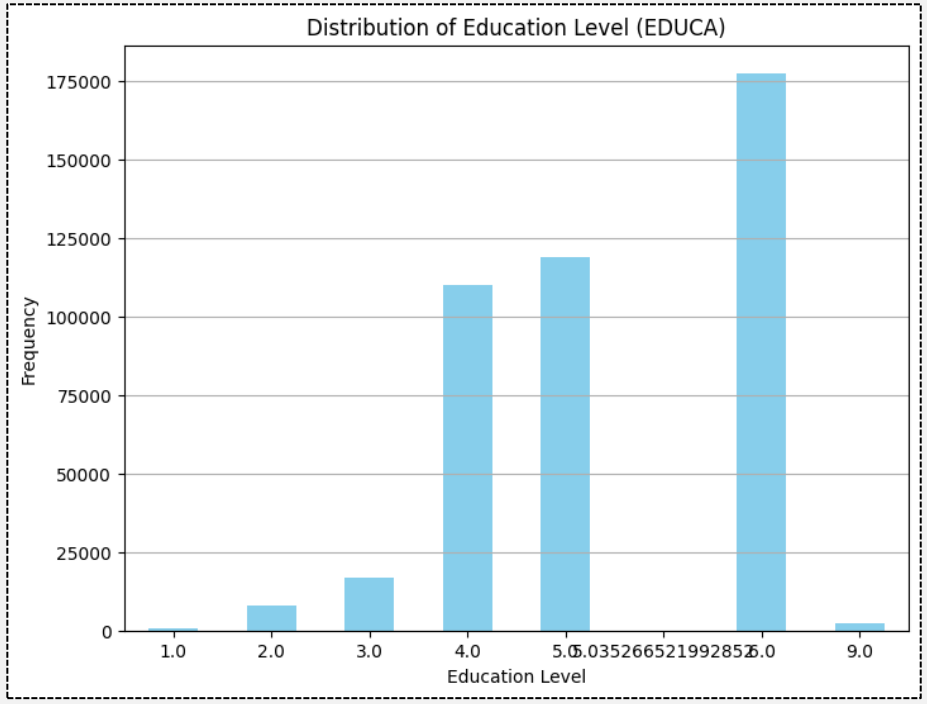
AFTER DATA CLEANING



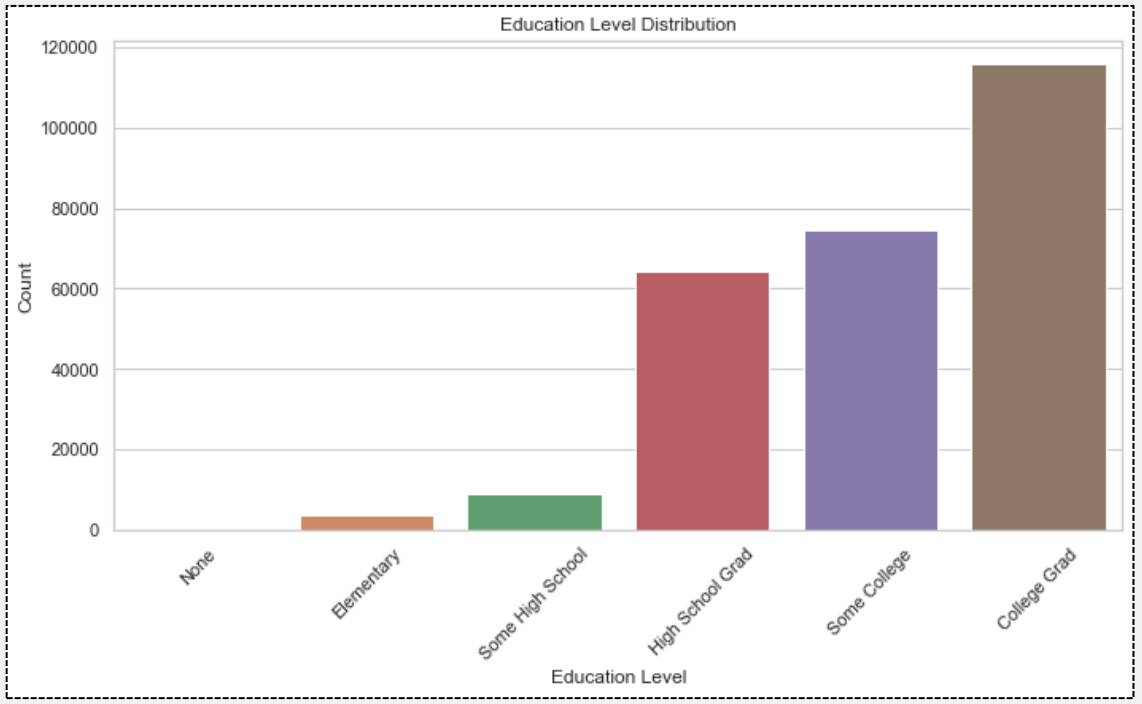
DATA CLEANING EXAMPLE #2

FILTERING & MAPPING EDUCA VALUES

BEFORE DATA CLEANING



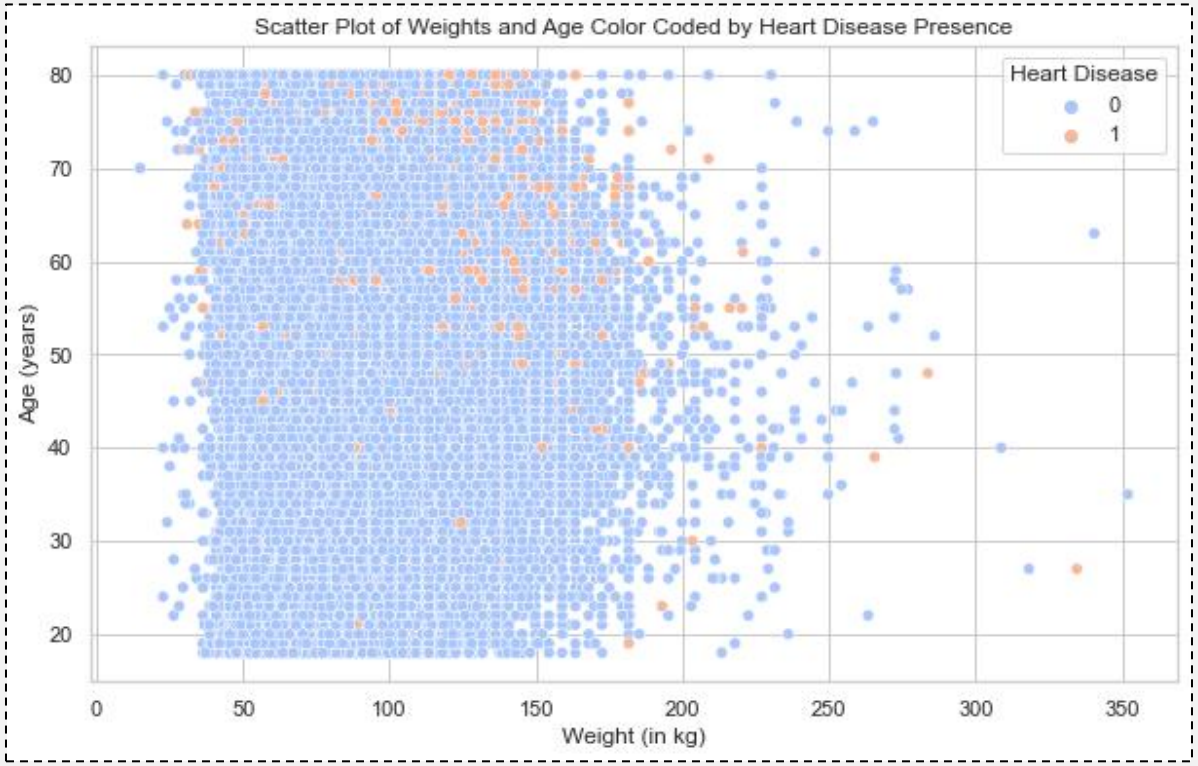
AFTER DATA CLEANING



INSIGHT FROM DATA EXPLORATION #1

Plot displays instances of heart disease compared against weight (kilograms) and age (years)

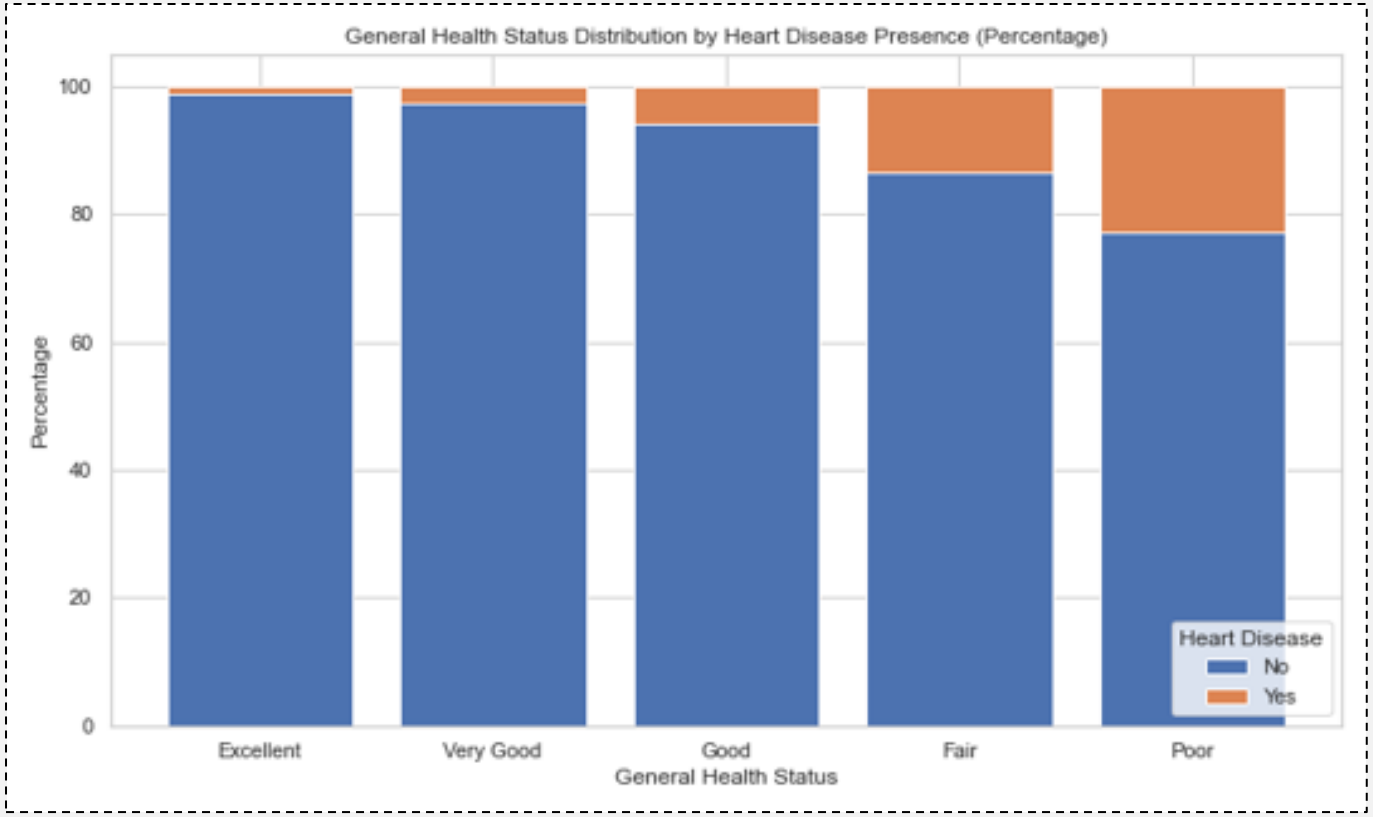
More prevalent red dots in top right corner show that an increase in both age and weight correlates with an increase in heart disease presence



INSIGHT FROM DATA EXPLORATION #2

Plot displays percentage of heart disease presence by reported general health status

Graph shows a correlation between worsening reported general health and prevalence of heart disease



ML TECHNIQUES TO IMPLEMENT

Models: KNN, random forest, gradient boosting, and SVM (ensemble stacking to be used)

Grid search for best combination of models/parameters

Recall will be a key metric

Applied Machine Learning - COMS W4995

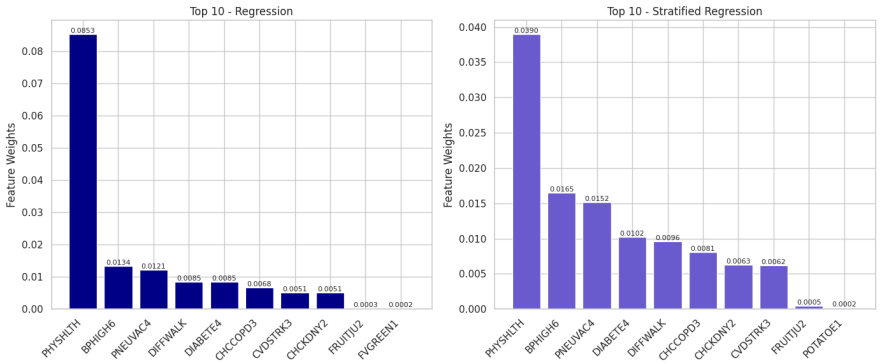
Project Deliverable 3

Group 17: Avraham Kaminker (ark2218), Blanca Herreros de Tejada Lobo (beh2154), Eldar Djangirov (ed3047), Rashmi Chelliah(rc3605), Yuang Fan (yf2676)

Background & Dataset: Machine learning in healthcare research holds significant potential for providing insights into critical questions, such as the factors that increase susceptibility to heart disease. We were motivated to leverage our background to investigate and predict if a person is prone to getting heart disease using data that is easily attainable without the need for professional and expensive equipment. We utilized the 2021 Behavioral Risk Factor Surveillance System (BRFSS) dataset from the CDC, which includes 303 variables and over 438,693 entries. We focused on the target variable CVDCRHD4, which indicates a diagnosis of angina or coronary heart disease. Data preprocessing was detailed in the second deliverable. Ultimately, we reduced the dataset to 27 key features. Throughout our modeling efforts, we prioritized the recall metric, considering the significant consequences of false negatives in medical diagnostics.

1. **Logistic Regression Model:**

As a baseline approach, we trained a basic logistic regression model on our dataset using CVDCRHD4 as the target variable and a stratified 80/20 train/test split, ensuring that the ratio of 0/1 in the target variable CVDCRHD4 was uniform in both training and test datasets. Since the dataset is imbalanced, we also applied SMOTE to generate synthetic data for the minority positive class. Upon training the model, the following metrics were returned on the test data: *Accuracy: 63.50% | Recall: 72.76% | F1-Score: 17.78%.* The top ten features in descending order of importance of each model can be found in the figure below.



2. **Neural Network Model:**

An alternative solution for the stated problem was Neural Networks. Initially, we examined the structural performance of the models and determined that models with more than five layers did not effectively perform. Consequently, we proceeded with hyperparameter tuning using two datasets: one processed with SMOTE and another with balanced weights. We employed a Random Search strategy to evaluate 20 different combinations of parameters for each dataset: the number of layers (1-5), nodes per layer (16-128), and dropout rates (0.1-0.5). Each configuration was trained for 20 epochs, and we recorded the two models that demonstrated the highest validation recall for each dataset.

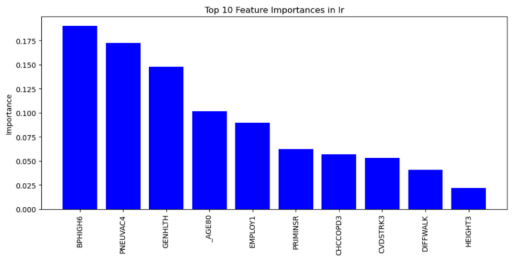
The model utilizing SMOTE achieved a validation recall of 0.812 and an Area Under the Curve (AUC) of 0.81. Meanwhile, the model with balanced weights reached a validation recall of 0.904 and an AUC of 0.84. The AUC is important here because it reflects the ability of the model to distinguish the classes across all classification thresholds. Following these findings, we further analyzed the structure of the balanced weight model and proceeded to train a final model over 30 epochs and stored the best-performing weights. Under the default threshold value of 0.5, this final model yielded the following metrics on the test set: *Accuracy: 73.55% | Recall: 78.67% | F1-Score: 24.39% | AUC: 0.83*.

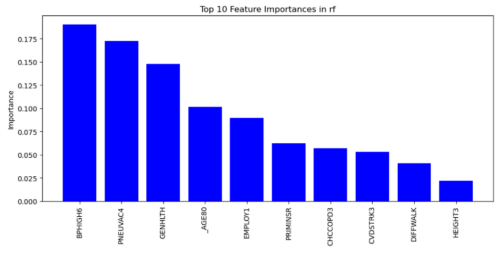
Given our focus on maximizing recall due to the critical nature of false negatives in medical diagnosis, we considered lowering the threshold value to enhance recall at the cost of potential reduction in accuracy. This adjustment emphasizes our commitment to prioritizing sensitivity in detecting heart disease.

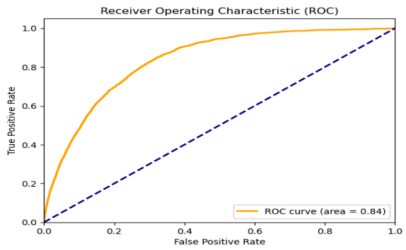
3. **Ensemble Model:**

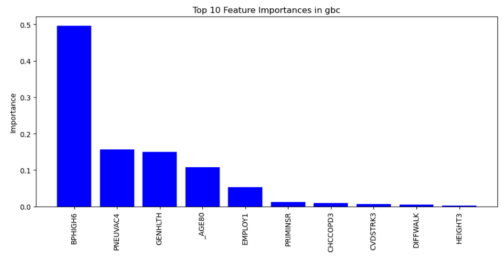
Finally, an ensemble model (logistic regression model, random forest classifier, and gradient boosting classifier) was used on our dataset to try and synthesize multiple models. We first created individual classifiers and then an ensemble of the classifiers using the voting classifier. To balance the dataset, the model was retrained using SMOTE sampling. Due to the relatively low recall score, GridSearchCV (cv=3) was employed to tune the hyperparameters (using a subset of the data to speed up training). The hyperparameters tuned were: Regularization strength for logistic regression, number of trees in the forest, forest minimum depth, GBC number of

boosting stages, and GBC learning rate. After retraining the model, the scores on the test data were: *Accuracy: 75.08% | Recall: 94.04% | Precision: 14.95% | F1-Score: 25.02% | AUC: 0.84.* The improved recall score was promising, but the low precision metric forecasts a large proportion of false positives (people being falsely diagnosed with heart disease).









Conclusion:

In conclusion, we explored three different Applied Machine Learning techniques in an attempt to solve this problem. Given the importance of minimizing False Negatives in diagnosis, we prioritized increasing Recall value throughout our models.

Concluding the modeling phase, the ensemble model has demonstrated promising outcomes, attaining the highest recall and AUC. Considering that the features are collected through easily attainable survey questions instead of professional medical examinations, and despite a high false positive rate, it remains highly useful for preliminary screenings.

For future works, it would be interesting to dedicate efforts toward the hyperparameter tuning of this model. Focusing on this aspect is particularly compelling as fine-tuning the hyperparameters can enhance the model's performance by optimizing its ability to generalize from the training data to unseen data. This approach may lead to more robust and accurate predictions, thus increasing the overall effectiveness of the model.

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
df = pd.read_csv("cleaned_cdc_data_V4.csv")
df
```

Out[5]:

	_AGE80	GENHLTH	WEIGHT2	EMPLOY1	POTATOE1	FVGREEN1	FR
0	70.0	5.0	14.840352	7.0	52.142857	13.035714	1
1	72.0	2.0	35.039720	7.0	52.142857	0.000000	5
2	62.0	2.0	40.192620	7.0	1.520833	4.055556	1
3	80.0	3.0	40.192620	7.0	52.142857	52.142857	(
4	65.0	3.0	108.960000	7.0	17.380952	26.071429	2
...	...	...	...	...	...	...	
267099	66.0	4.0	69.916000	7.0	52.142857	182.500000	5
267100	30.0	2.0	61.290000	1.0	0.811111	26.071429	5
267101	54.0	2.0	99.880000	7.0	52.142857	26.071429	(
267102	67.0	2.0	79.450000	7.0	3.041667	17.380952	(
267103	45.0	1.0	81.266000	1.0	12.166667	17.380952	1

267104 rows × 28 columns

In [9]:
```
X = df.drop(['CVDCRHD4'], axis=1) # Features y = df_raw['isFraud']
y = df.CVDCRHD4
```

In [97]:
```
df.PNEUVAC4.value_counts()
```

Out[97]:
```
0.0    163539
1.0    103565
Name: PNEUVAC4, dtype: int64
```

In [12]:
```
y.value_counts(normalize=True)
# It is imbalanced
```

Out[12]:
```
0    0.945766
1    0.054234
Name: CVDCRHD4, dtype: float64
```

In [17]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Return the shapes of the splits to confirm
(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Out[17]:
```
((213683, 27), (53421, 27), (213683,), (53421,))
```

In [19]:
```
y_train.value_counts(normalize=True)
```

Out[19]:
```
0    0.945765
1    0.054235
Name: CVDCRHD4, dtype: float64
```

In [20]:
```
y_test.value_counts(normalize=True)
```

Out[20]:
```
0    0.94577
1    0.05423
Name: CVDCRHD4, dtype: float64
```

In [25]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report, accuracy_score

# Create the individual classifiers
clf1 = make_pipeline(StandardScaler(), LogisticRegression(random_state
clf2 = RandomForestClassifier(n_estimators=100, random_state=101)
clf3 = GradientBoostingClassifier(n_estimators=100, random_state=101)

# Create an ensemble of the classifiers using the Voting Classifier
# We use 'soft' voting to predict the class label based on the argmax
ensemble = VotingClassifier(estimators=[
    ('lr', clf1), ('rf', clf2), ('gbc', clf3)], voting='soft')

# Fit the ensemble to the training data
ensemble.fit(X_train, y_train)
```

In [27]:
```
from sklearn.metrics import classification_report, accuracy_score, rec

# Make predictions on the training set

y_train_pred = ensemble.predict(X_train)

# Evaluate the ensemble on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)


print("Training Accuracy:", train_accuracy)
print("Training Recall:", train_recall)
```

```
Training Accuracy: 0.9513999709850574
Training Recall: 0.10648028302700838
```

In [28]:
```
# Make predictions on the test set
y_test_pred = ensemble.predict(X_test)

# Evaluate the ensemble on the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)

print("Training Accuracy:", test_accuracy)
print("Training Recall:", test_recall)
```

```
Training Accuracy: 0.9456955130005055
Training Recall: 0.021746634449430445
```

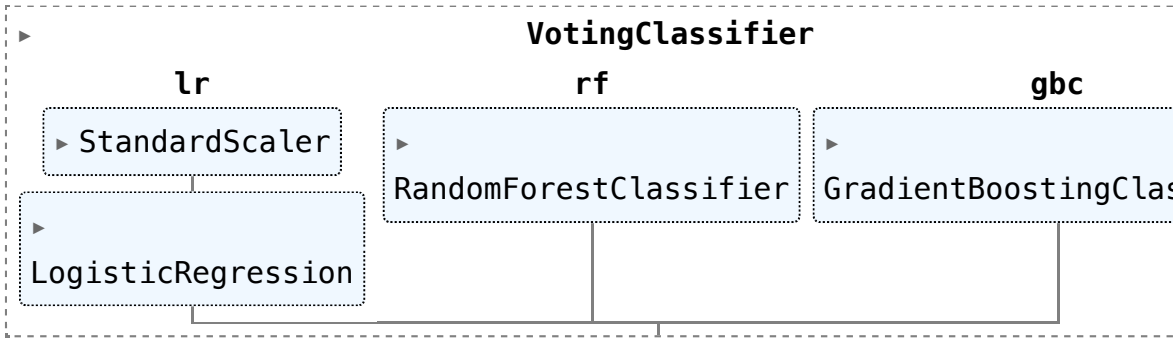
In [ ]:
```
# The results indicate that we achieved a
# high accuracy of approximately 94.57% on the test set,
# which is not surprising given the high class imbalance in favor of t
# However, the recall is too low at approximately 2.17%.

# Let's try SMOTE to make the dataset more balanced.
```

In [29]:
```
from imblearn.over_sampling import SMOTE

# Initialize SMOTE
smote = SMOTE(random_state=101)

# Resample the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check the class distribution after SMOTE
y_train_smote.value_counts()
```

Out[29]:
```
0    202094
1    202094
Name: CVDCRHD4, dtype: int64
```

In [30]:
```
# Train using dataset after SMOTE
ensemble.fit(X_train_smote, y_train_smote)
```

Out[30]:



In [31]:
```
y_train_pred = ensemble.predict(X_train)
```

```
# Evaluate the ensemble on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)


print("Training Accuracy:", train_accuracy)
print("Training Recall:", train_recall)
```

```
Training Accuracy: 0.9690148490988988
Training Recall: 0.6411252049357149
```

In [32]:
```
y_test_pred = ensemble.predict(X_test)

# Evaluate the ensemble on the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)

print("Training Accuracy:", test_accuracy)
print("Training Recall:", test_recall)
```

```
Training Accuracy: 0.9294285019000019
Training Recall: 0.24646185709354504
```

In [ ]:
```
# We've improved the Recall
# But it still needs work.
```

In [ ]:
```
# Let's run a grid search to identify the best candidates
```

In [68]:
```
# Subset to 20,000 of each class to keep it balanced
indices_0 = y_train_smote[y_train_smote == 0].index[:20000]
indices_1 = y_train_smote[y_train_smote == 1].index[:20000]

# Combine the indices
subset_indices = indices_0.union(indices_1)

# Subset the training data and labels
X_train_subset3 = X_train_smote.loc[subset_indices]
y_train_subset3 = y_train_smote.loc[subset_indices]
```

In [69]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
import numpy as np

# Create the individual classifiers with pipeline for logistic regress
clf1 = make_pipeline(StandardScaler(), LogisticRegression(random_state
clf2 = RandomForestClassifier(random_state=101)
clf3 = GradientBoostingClassifier(random_state=101)
```

```
# Ensemble setup
ensemble = VotingClassifier(estimators=[
    ('lr', clf1), ('rf', clf2), ('gbc', clf3)],
    voting='soft')

# Simplified parameters grid
params = {
    'lr__logisticregression__C': [0.001, 0.1],  # Regularization stren
    'rf__n_estimators': [50, 100],  # Number of trees in the forest
    'rf__max_depth': [3, 10, None],  # Minimum depth of 3, plus 10 and
    'gbc__n_estimators': [50, 100],  # Number of boosting stages
    'gbc__learning_rate': [0.01, 0.1]  # Learning rate options
}

# Calculate the total number of fits: number of parameter combinations
total_fits = np.prod([len(v) for v in params.values()]) * 3  # 3 is th

# Grid search with cross-validation
grid = GridSearchCV(ensemble, param_grid=params, cv=3, scoring='recall

# Fit grid search on the subset of training data
grid.fit(X_train_subset3, y_train_subset3)

# Print the total number of fits
print(f"Total fits planned: {total_fits}")

# Best parameters and best score
print("Best parameters:", grid.best_params_)
print("Best recall:", grid.best_score_)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
Total fits planned: 144
Best parameters: {'gbc__learning_rate': 0.01, 'gbc__n_estimators': 100,
'lr__logisticregression__C': 0.1, 'rf__max_depth': 3, 'rf__n_estimator
s': 50}
Best recall: 0.8338989966193259
```

In [87]:

```
from sklearn.metrics import classification_report, accuracy_score, rec

y_test_pred = grid.predict(X_test)

# Calculate the recall for the training data

test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)


# Print the training recall
print("Test Accuracy:", test_accuracy)
print("Test Precision:", test_precision)
print("Test Recall:", training_recall)
```

```
print("Test F1:", test_f1)
```

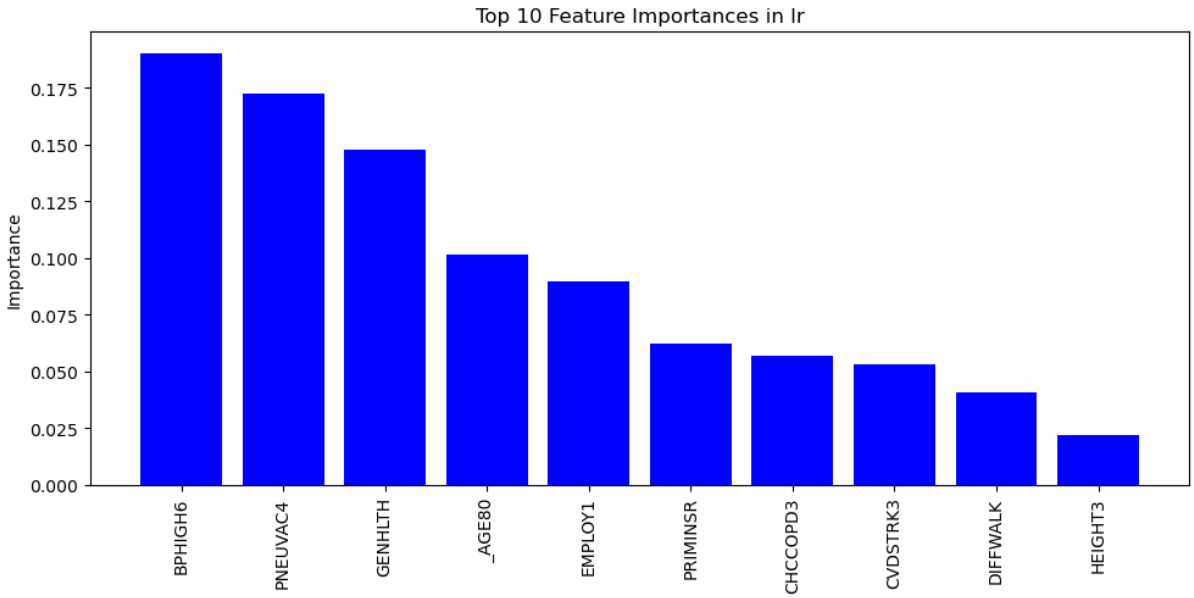
```
Test Accuracy: 0.7507534490181764
Test Precision: 0.14950881442605302
Test Recall: 0.9404336595841539
Test F1: 0.2502393152767611
```

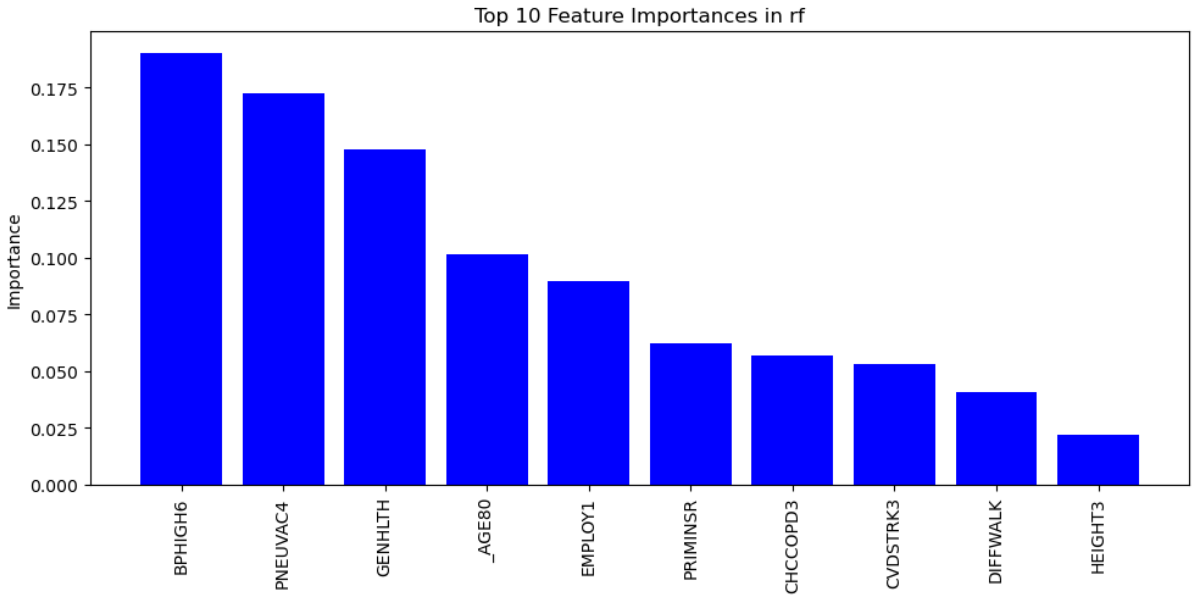
In [ ]:
```
# A high recall is achieved: the model successfully identifies most pa
# minimizing the risk of missing cases.
# This is important for medical diagnostics
# where failing to detect a condition can be dangerous.

# A low Precision: means a large proportion of the positive
# predictions made by the model are false positives.
# Thus, many patients would be incorrectly diagnosed with heart diseas
```

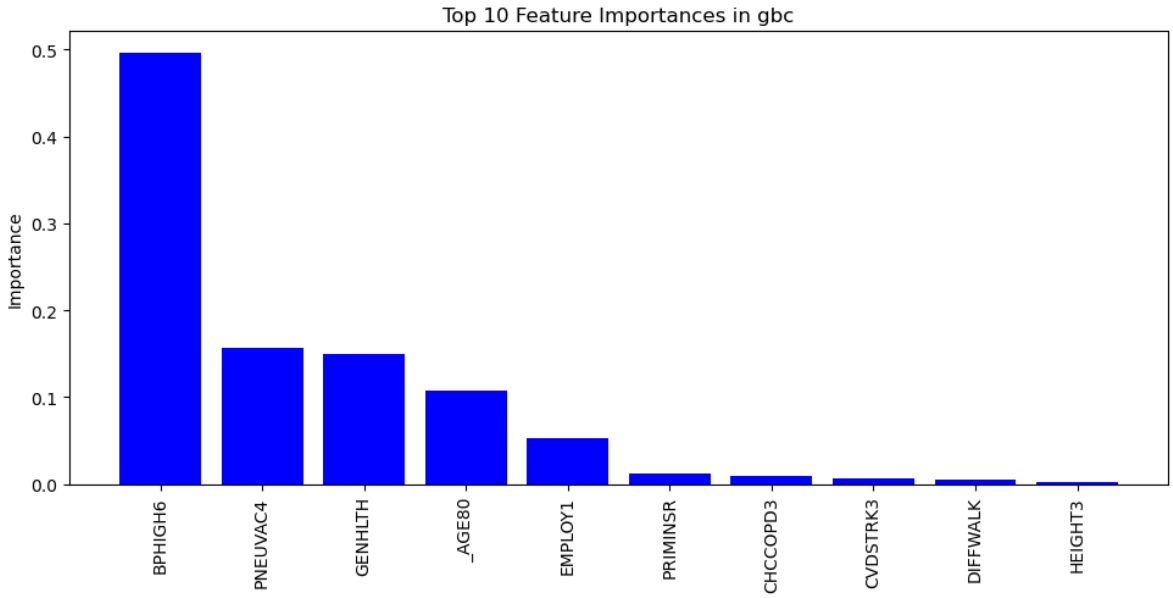
In [82]:
```
import matplotlib.pyplot as plt
import numpy as np

# Assuming 'grid' is your trained GridSearchCV object
best_ensemble = grid.best_estimator_

# Feature names from your dataset
feature_names = X_train.columns

# Initialize a dictionary to hold feature importances or coefficients
feature_importances = {}

# Extract feature importances or coefficients from each model in the e
for name, estimator in best_ensemble.named_estimators_.items():
    if hasattr(estimator, 'feature_importances_'):
        # Models like RandomForest and GradientBoosting
        importances = estimator.feature_importances_
    elif hasattr(estimator, 'coef_'):
        # Logistic Regression (take the absolute values of coefficient
        # Check if the model is wrapped inside a pipeline
        if hasattr(estimator, 'named_steps'):
            lr_model = estimator.named_steps['logisticregression']
        else:
            lr_model = estimator
        importances = np.abs(lr_model.coef_[0])

    # Sort features by importance
    indices = np.argsort(importances)[::-1]
    # Keep top 10 features
    top_indices = indices[:10]
    feature_importances[name] = importances[top_indices]

# Plotting top 10 feature importances from each model
fig, axes = plt.subplots(nrows=len(feature_importances), figsize=(10,
for ax, (name, importances) in zip(np.ravel([axes]), feature_importanc
    top_features = feature_names[indices[:10]]  # Selecting the top 10
    ax.bar(top_features, importances, color='b')
```

```
    ax.set_title(f'Top 10 Feature Importances in {name}')
    ax.set_ylabel('Importance')
    ax.set_xticklabels(top_features, rotation=90)

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

```
/var/folders/k2/sqwxh15s60x79d27b61yzgy00000gn/T/ipykernel_11974/327148
1115.py:40: UserWarning: FixedFormatter should only be used together wi
th FixedLocator
  ax.set_xticklabels(top_features, rotation=90)
```



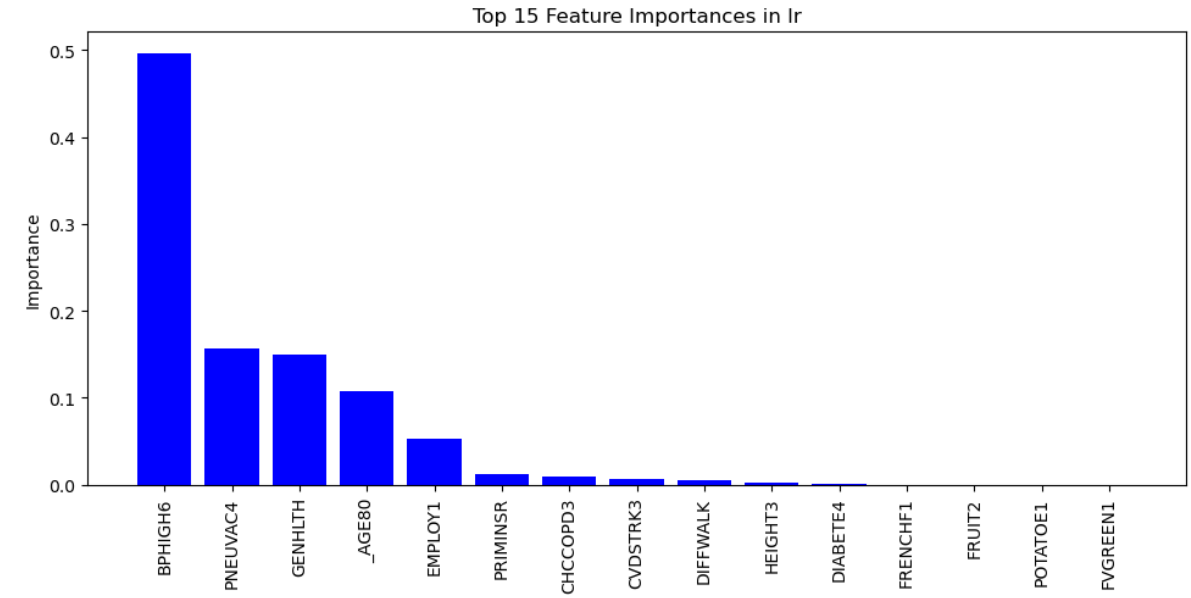


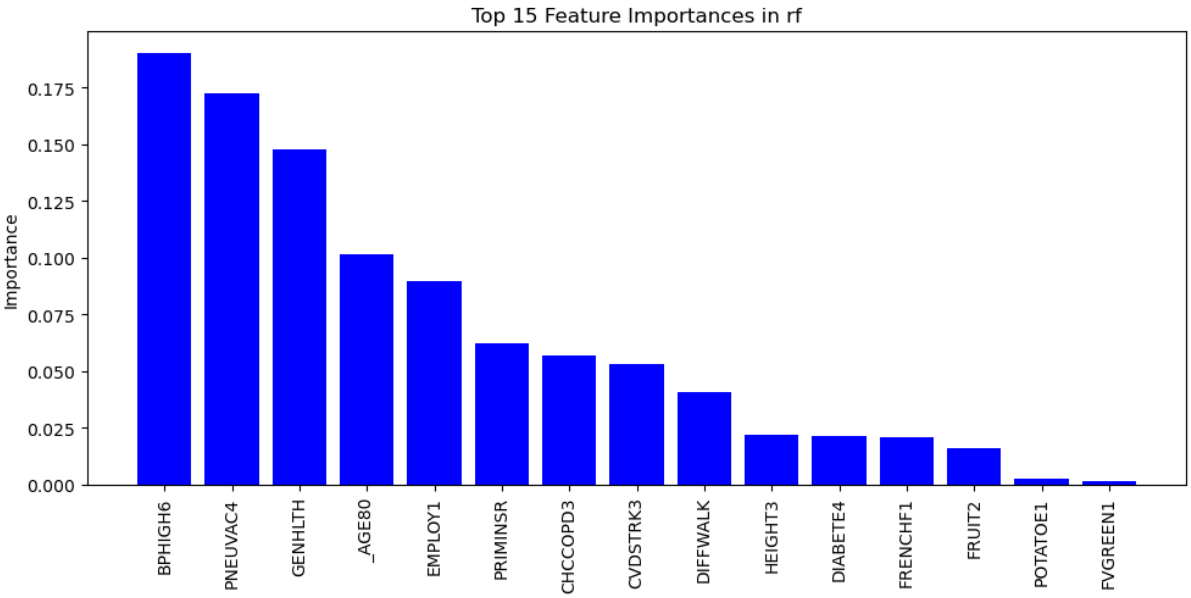


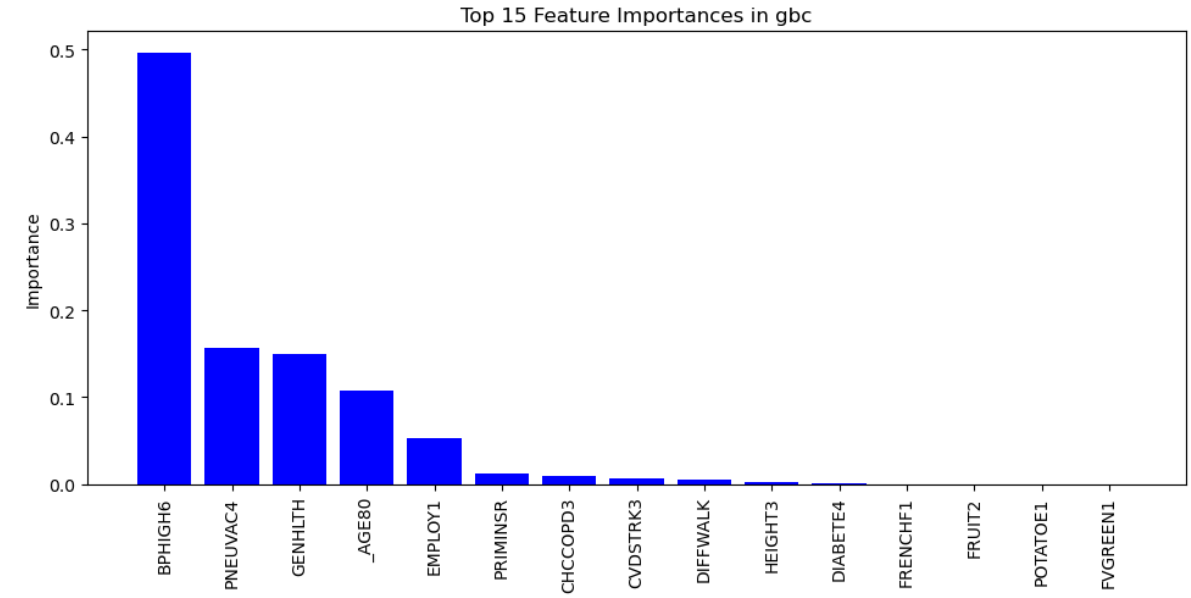
In [86]:
```
import matplotlib.pyplot as plt
```

```
import numpy as np

# Assuming 'grid' is your trained GridSearchCV object
best_ensemble = grid.best_estimator_

# Feature names from your dataset
feature_names = X_train.columns

# Initialize a dictionary to hold feature importances or coefficients
feature_importances = {}

# Extract feature importances or coefficients from each model in the e
for name, estimator in best_ensemble.named_estimators_.items():
    if hasattr(estimator, 'feature_importances_'):
        # Models like RandomForest and GradientBoosting
        importances = estimator.feature_importances_
    elif hasattr(estimator, 'coef_'):
        # Logistic Regression (take the absolute values of coefficient
        # Check if the model is wrapped inside a pipeline
        if hasattr(estimator, 'named_steps'):
            lr_model = estimator.named_steps['logisticregression']
        else:
            lr_model = estimator
        importances = np.abs(lr_model.coef_[0])

    # Sort features by importance and get the top 15
    indices = np.argsort(importances)[::-1][:15]
    feature_importances[name] = importances[indices]

# Plotting top 15 feature importances from each model
fig, axes = plt.subplots(nrows=len(feature_importances), figsize=(10,
for idx, (name, importances) in enumerate(feature_importances.items())
    # Get the correct axis
    ax = axes[idx] if len(feature_importances) > 1 else axes
    top_features = feature_names[indices[:15]]  # Selecting the top 15
    ax.bar(top_features, importances, color='b')
    ax.set_title(f'Top 15 Feature Importances in {name}')
    ax.set_ylabel('Importance')
    ax.set_xticklabels(top_features, rotation=90)

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

```
/var/folders/k2/sqwxh15s60x79d27b61yzgy00000gn/T/ipykernel_11974/335528
0045.py:40: UserWarning: FixedFormatter should only be used together wi
th FixedLocator
  ax.set_xticklabels(top_features, rotation=90)
```







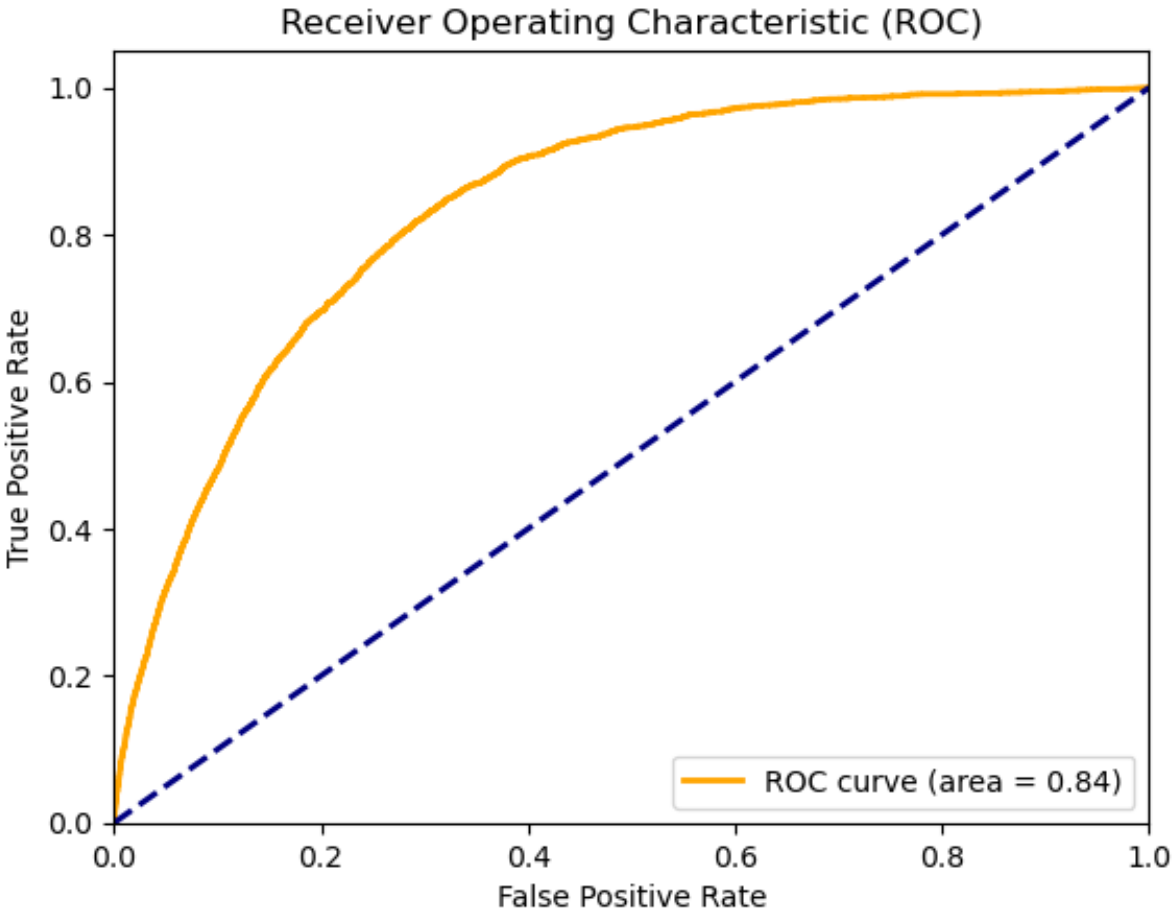
In [ ]:
```
# ORDER OF IMPORTANCE OF FEATURES
```

```
# BPHIGH4: high blood pressure or hypertension. "BP" is commonly used

# PNEUVAC4: related to pneumococcal vaccination, given "PNEU" stands f

# GENHLTH: stands for general health status, a self-reported measure r

# AGE80: represent the age thresholded at 80.

# EMPLOY1: refers to employment status.

# PRIMINSR: represents main source of health insurance coverage.

# CHCCOPD3: stands for Chronic Obstructive Pulmonary Disease status.

# CVDSTRK3: related to stroke, with "CVD" commonly standing for cardio

# DIFFWALK: regarding difficulty walking; a measure of physical disabi

# HEIGHT3: height of the individuals

# DIABETE4: Indicates if the respondent has been told they have diabet

# FRENCHF1: Consumption frequency of french fries.

# FRUIT2: Consumption frequency of fruits.

# POTATOE1: Consumption frequency of potatoes, possibly excluding fren

# FVGREEN1: Consumption frequency of green vegetables.


# Certainly the last 4 are not quite as pronounced of factors.
```

In [95]:

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


y_test_pred_prob = grid.best_estimator_.predict_proba(X_test)[:, 1]

# Calculate the ROC curve points and the AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC
plt.figure()
plt.plot(fpr, tpr, color='orange', lw=2, label='ROC curve (area = %0.2
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



In [ ]:
```
# RECAP:

# Since the dataset was imbalanced, SMOTE was utilized (synthetic data
# An ensemble was trainedconsisting of Logistic Regression,
# Random Forest Classifier, and Gradient Boosting Classifier.

# Then, a GridSearchCV was performed to tune hyperparameters.
# A subset of data was chosen to speed up training.
# The metrics came back as:

# Test Accuracy: 0.7507534490181764
# Test Precision: 0.14950881442605302
# Test Recall: 0.9404336595841539
# Test F1: 0.2502393152767611

# Interpretation of Metrics:
# A high recall is achieved implies that the model successfully
# identifies most patients with heart disease, minimizing the risk
# of missing cases. This is crucial for medical diagnostics where
# failing to detect a condition can be dangerous.
# However, a low precision means that large proportion of the
```

```
# positive predictions made by the model are false positives.
# Thus, many patients would be incorrectly diagnosed with heart diseas

# We have most predictive features and its descriptions based on this
# Additionally, an ROC curved is plotted.
```