USC Viterbi
School of Engineering

FINAL REPORT

FINAL REPORT

OUTLINE

- Exploratory Data Analysis Report
- Data Preparation Plan
- Model Pipeline
- Summary Discussion

EDA REPORT

OUTLINE

- Business objective
- Dataset summary
- Data quality summary
- Univariate analysis
- Bivariate analysis

EDA REPORT

BUSINESS OBJECTIVE

- To develop a predictive model capable of accurately forecasting whether participants will develop coronary heart disease (CHD) within a 10-year timeframe. This model aims to leverage available data and advanced analytical techniques to enhance early identification and intervention strategies for individuals at risk of CHD.
- The data represents information on Patient demographics , medical history , biometric measurements and Lifestyle patterns(smoking)

USC Viterbi School of Engineering

EDA REPORT

DATASET SUMMARY

- “TenYeatCHD.csv” – dataset with 19 variables and 3,816 observations
- Dataset contains one potential response variables(TenYearCHD)

Variable	Description
Age	age of the participant at the time of examination
Male	gender of the participant (male =1, female = 0)
Education	Educational level of the patient (1 = less than high school, 2 = completed high school or equivalent, 3 = some college, 4= completed college or higher)
Income	Income of the patient
Current Smoker	whether the participant is currently a smoker (yes or no)
Cigarettes per Day	the average number of cigarettes smoked per day by current smokers
BP Meds	whether the participant is taking blood pressure medication (yes or no)
Prevalent Stroke	whether the participant has a history of stroke (yes or no)
Prevalent Hyp	whether the participant has a history of hypertension (yes or no)
Diabetes	whether the participant has diabetes (yes or no)
Total Chol	total cholesterol level in milligrams per deciliter
Sys BP	systolic blood pressure in millimeters of mercury
Dia BP	diastolic blood pressure in millimeters of mercury
BMI	body mass index in kilograms per square meter
Heart Rate	resting heart rate in beats per minute
Glucose	Blood glucose level in milligrams per deciliter
A1c	Hemoglobin A1c (%)
Ten Year CHD	whether the participant developed coronary heart disease (CHD) within 10 years of the examination (yes or no)

EDA REPORT

DATA ATTRIBUTE SUMMARY

Data types:

```
1 # getting the dtpes of all columns
2 df.dtypes
```

```
patientID            int64
male                 int64
age                  int64
education          float64
currentSmoker        int64
cigsPerDay         float64
BPMeds             float64
prevalentStroke      int64
prevalentHyp         int64
diabetes             int64
totChol            float64
sysBP              float64
diaBP              float64
BMI                float64
heartRate          float64
glucose            float64
TenYearCHD           int64
a1c                float64
income             float64
dtype: object
```

All the columns in the dataset are numerical values with datatype of either int or float.

Thus , no need to for further change of data types. We could convert the education column into Int as it a categorical variable but that would make any difference.

EDA REPORT

DATA QUALITY SUMMARY

Response variable stats:

```
1 df['TenYearCHD'].value_counts()
```

```
TenYearCHD
0    3237
1     579
Name: count, dtype: int64
```

The response variable is imbalanced as the ratio of number of instances with class 0 is way greater than instances with class 1

EDA REPORT
DATA QUALITY SUMMARY

Missing Values:

```python

# Check for missing values in each column
missing_count = df.isnull().sum()
missing_percentage = df.isnull().mean() * 100
missing_info_df = pd.concat([missing_count, missing_percentage], axis=1)
missing_info_df.columns = ['Missing Count', 'Missing Percentage']
print(missing_info_df)

```

```
                 Missing Count  Missing Percentage
patientID                    0            0.000000
male                         0            0.000000
age                          0            0.000000
education                   93            2.437107
currentSmoker                0            0.000000
cigsPerDay                1975           51.755765
BPMeds                      45            1.179245
prevalentStroke              0            0.000000
prevalentHyp                 0            0.000000
diabetes                     0            0.000000
totChol                     47            1.231656
sysBP                        0            0.000000
diaBP                        0            0.000000
BMI                         19            0.497904
heartRate                    1            0.026205
glucose                    361            9.460168
TenYearCHD                   0            0.000000
a1c                        361            9.460168
income                       0            0.000000
```

Cigs per day has the highest number of missing values.

- Either imputing with the median works
- Binning should work
- Find an underlying relationship between Cigsperday and current smoker

Glucose and a1c can be imputed with median value as there are continuous variables.

Bpmeds has 45 Nan values.

- Either imputing it with 1,0 based on mode becauce its categorical variable.
- Dropping the rows with NaN values

EDA REPORT

DATA QUALITY SUMMARY

unique Values:

```
1 df.nunique()
```

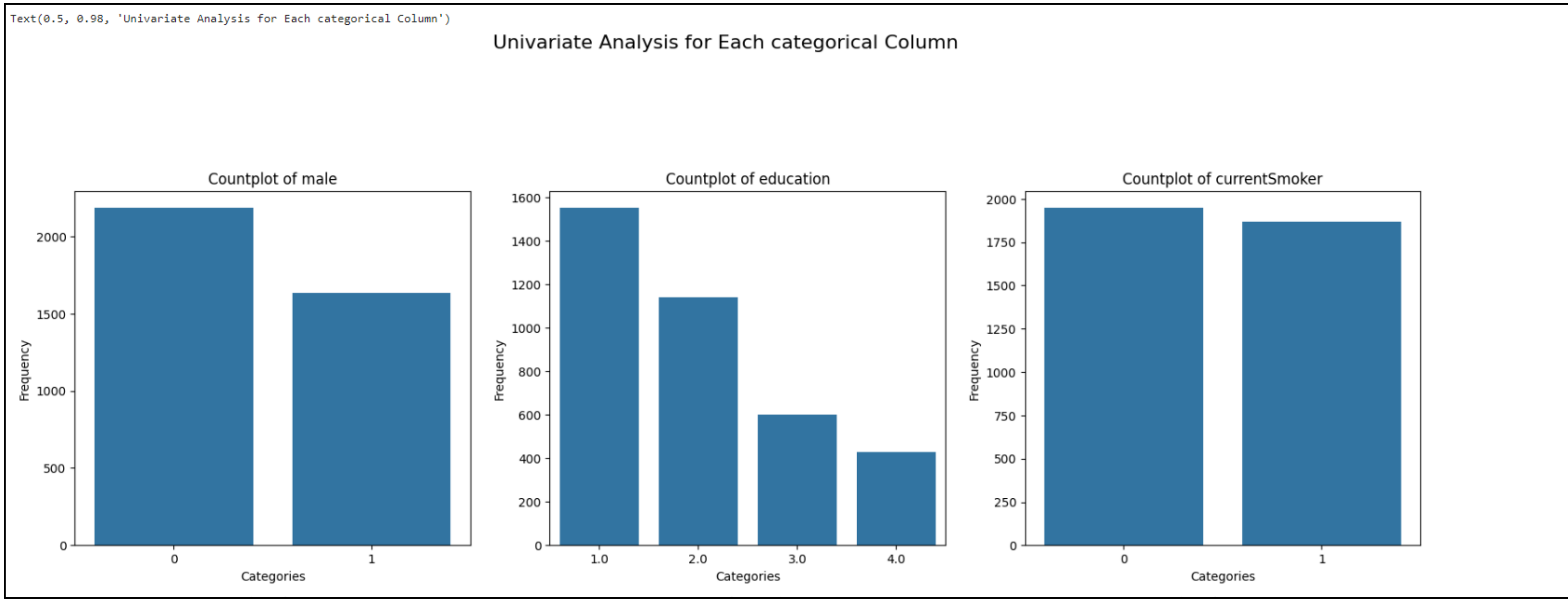
```
patientID          3816
male                  2
age                  39
education             4
currentSmoker         2
cigsPerDay           31
BPMeds                2
prevalentStroke       2
prevalentHyp          2
diabetes              2
totChol             246
sysBP               232
diaBP               142
BMI                1319
heartRate            73
glucose             134
TenYearCHD            2
a1c                3455
income             3282
dtype: int64
```

From this data I can infer that

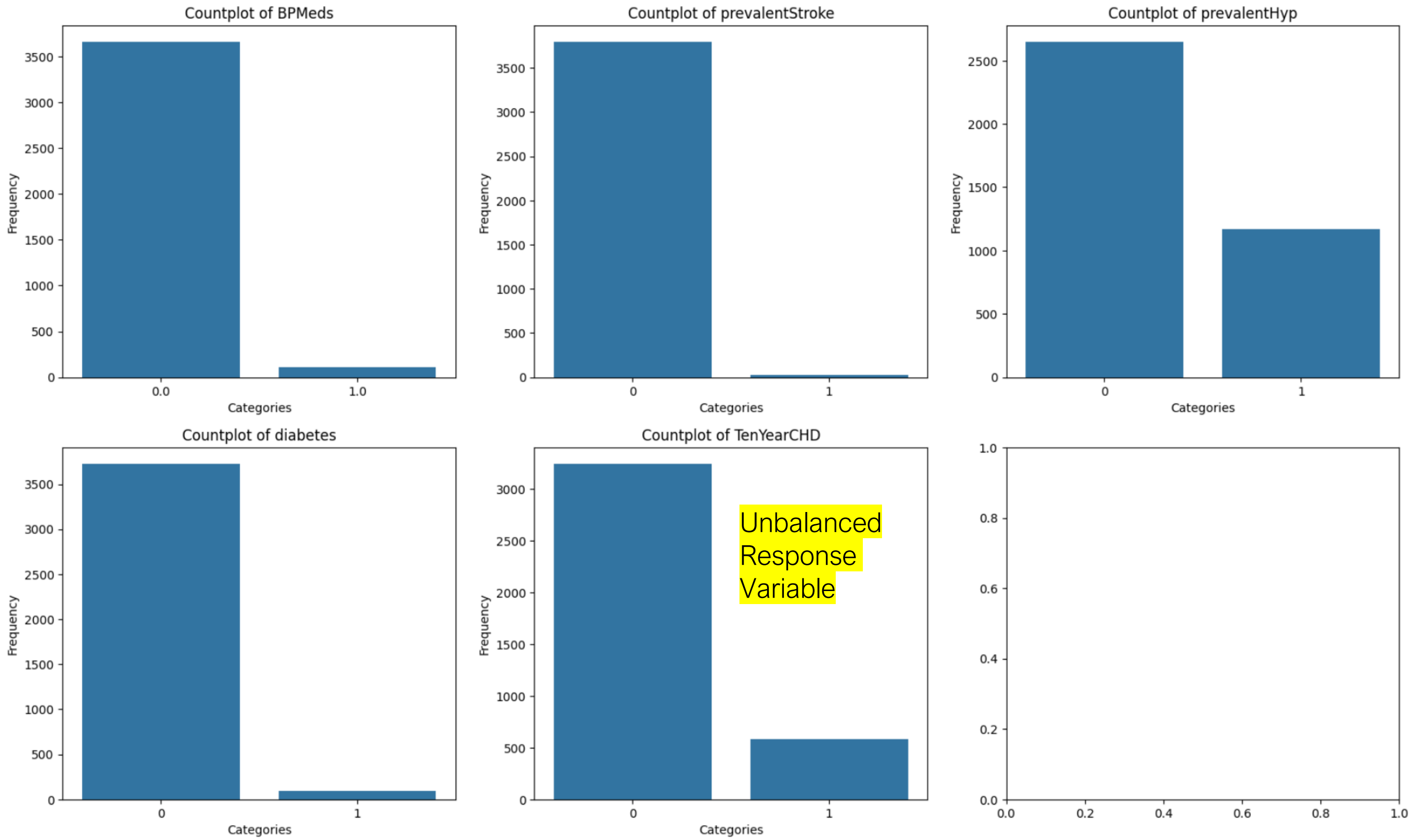
1. All the variable with nunique <=4 are considered categorical variables, and the rest are continuous variables.

2. CigsPerday and age can be considered as categorical but due to high cardinality I would consider it to be continuous variables.

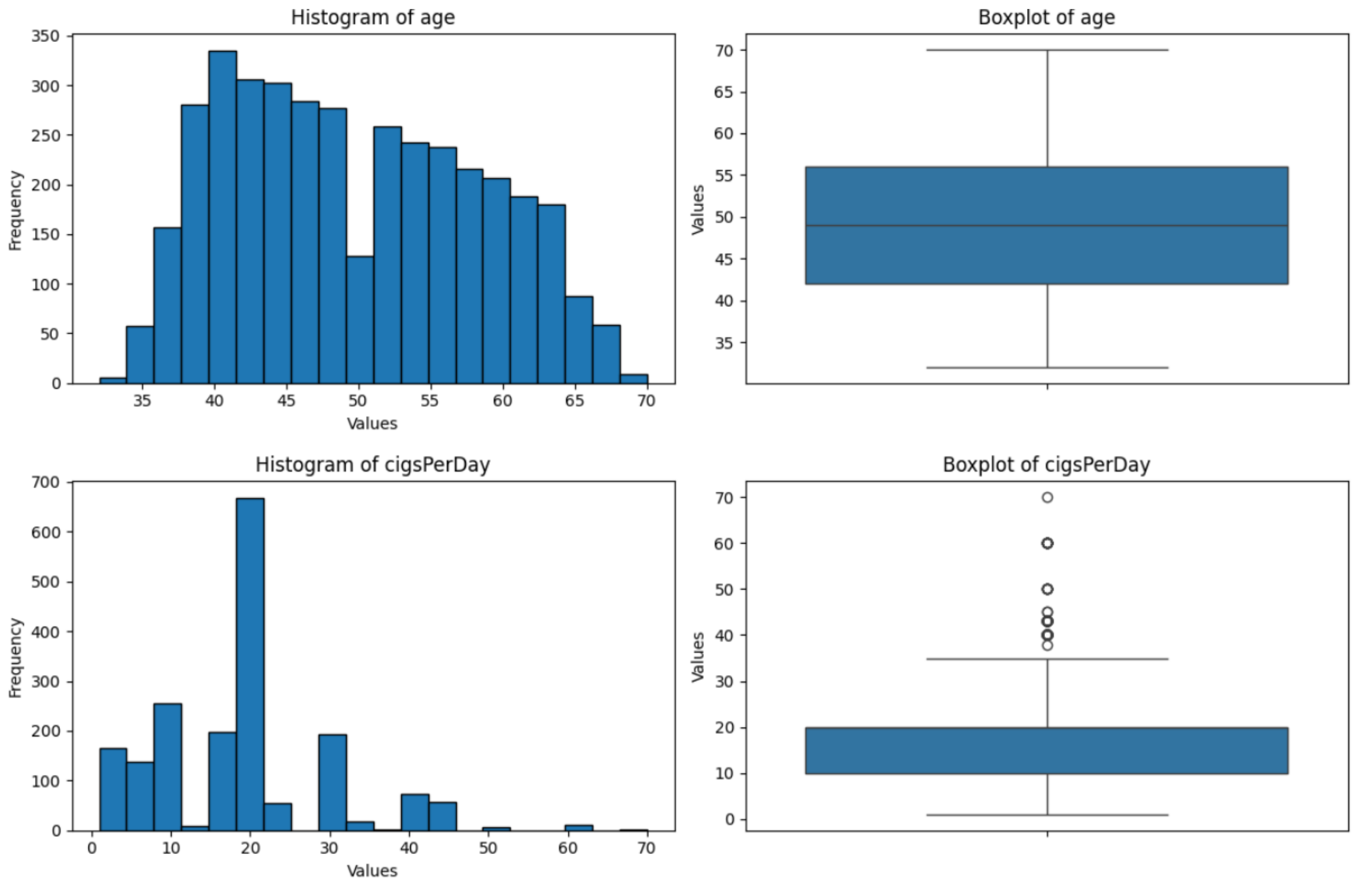
EDA REPORT – UNIVARIATE ANALYSIS CATEGORICAL VARIABLES



EDA REPORT – UNIVARIATE ANALYSIS
CATEGORICAL VARIABLES



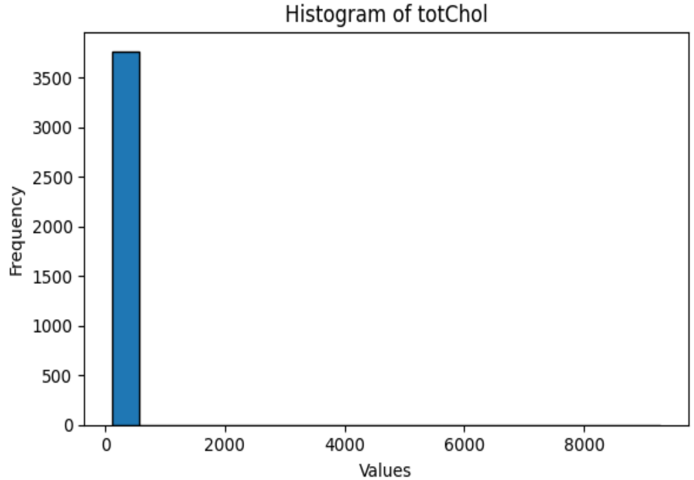
EDA REPORT – UNIVARIATE ANALYSIS
NUMERICAL CONTINUOUS VARIABLES

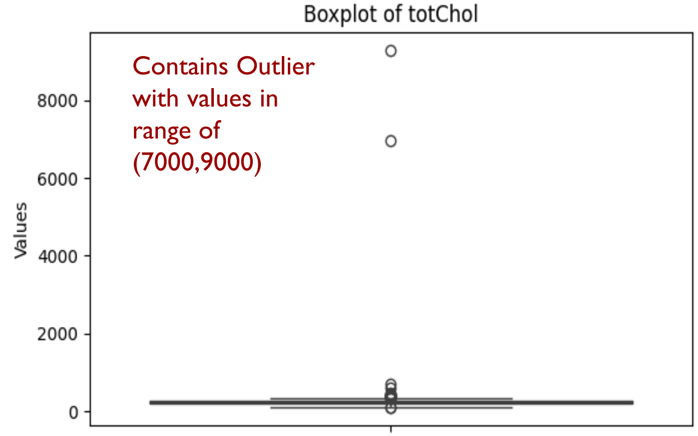


There shows no extreme skewness in bot variables. However , binning of values can be done in this instance for both variables . To do this I will have see how this variable changes with the response variable; by performing bivariate analysis

EDA REPORT – UNIVARIATE ANALYSIS

NUMERICAL CONTINUOUS VARIABLES

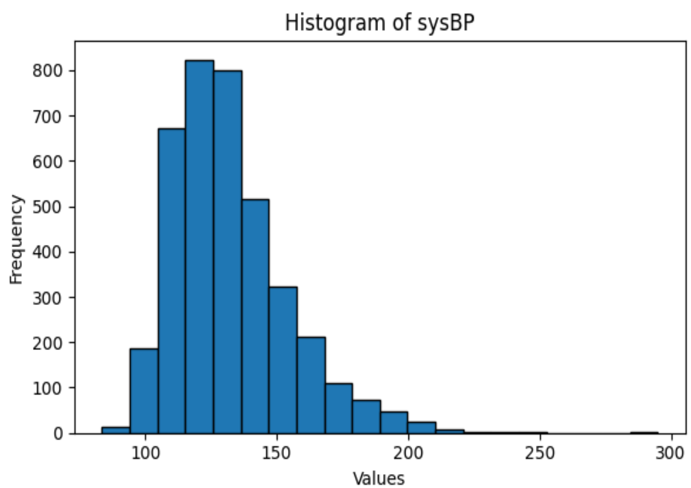


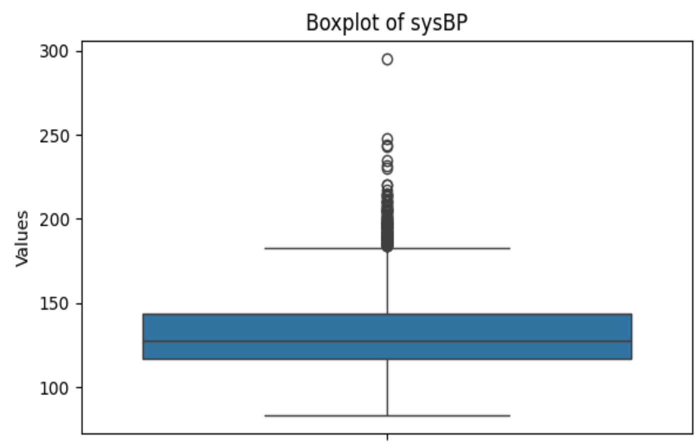


Clipping these or Winsorize at 95th and 5th percentile would resolve this

Skew:

totChol=41.031

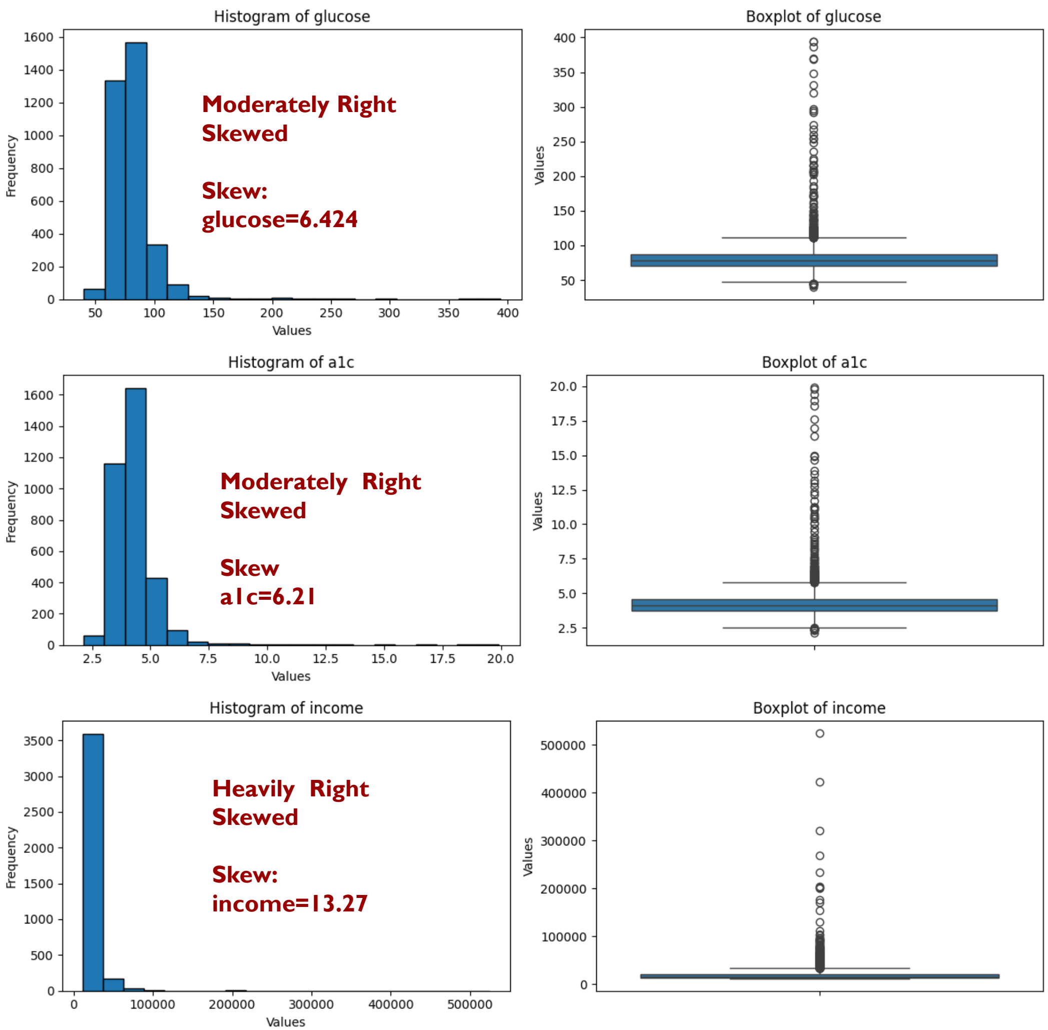




Moderate positive skew

sysBP=1.178

EDA REPORT – UNIVARIATE ANALYSIS
NUMERICAL CONTINUOUS VARIABLES

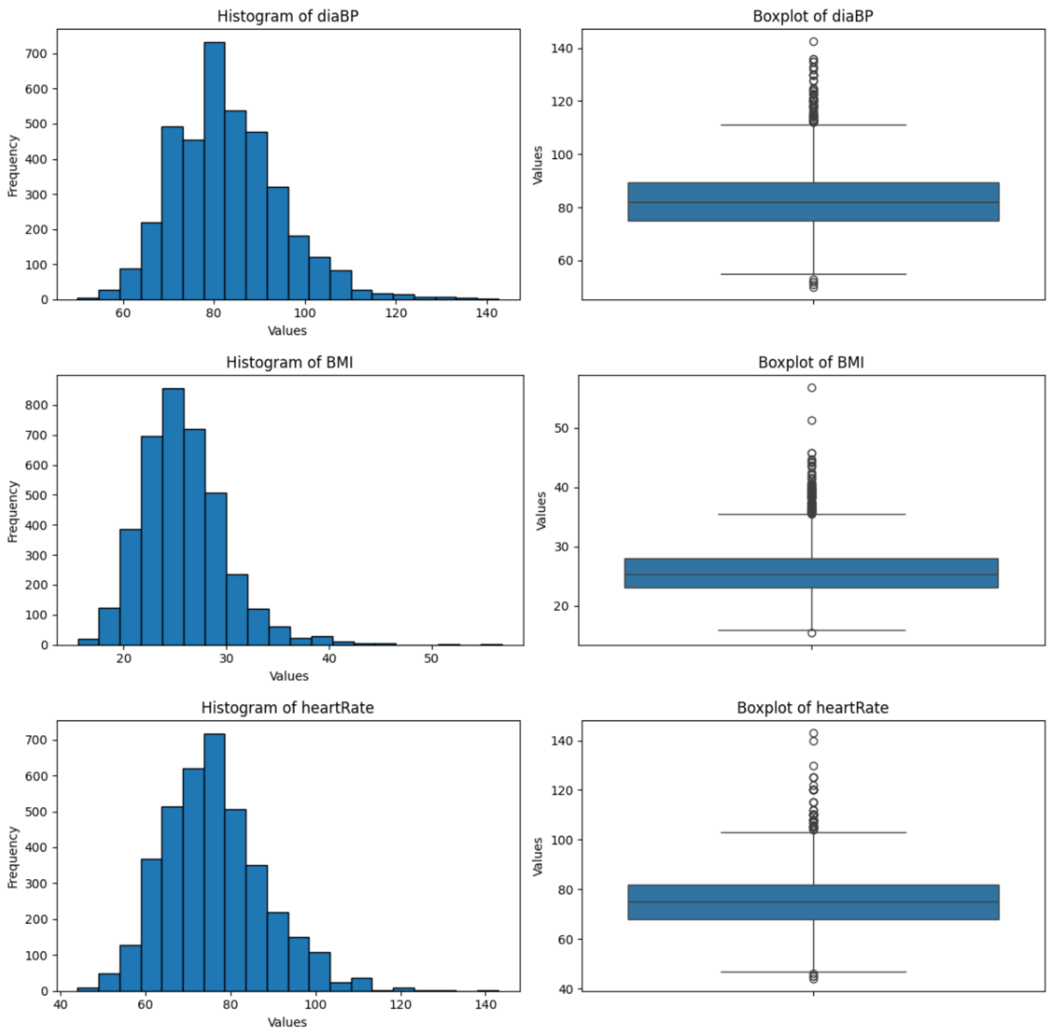


1. Since all these value are possible , instead of clipping and removing the outliers, transforming them using Log function would account to decrease in skewness

2. This would also be preserving the information at the same time.

USC Viterbi School of Engineering

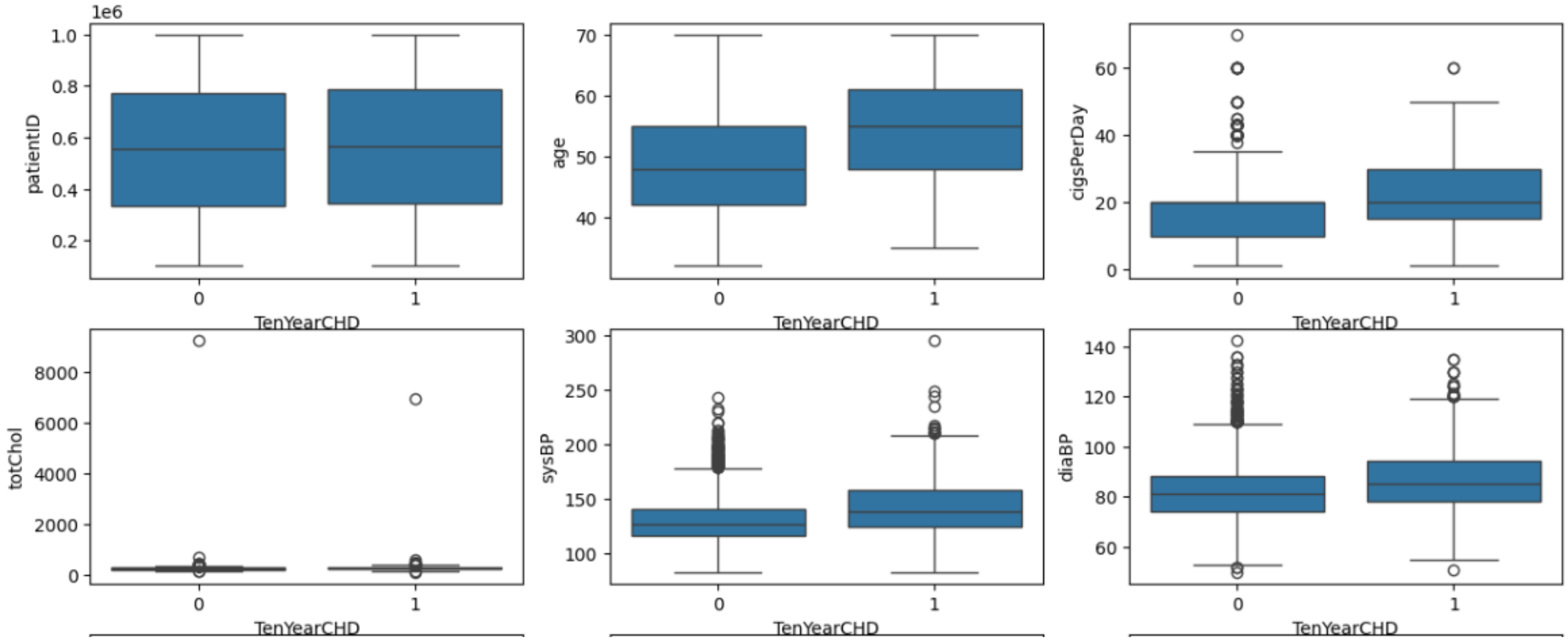
EDA REPORT – UNIVARIATE ANALYSIS
NUMERICAL CONTINUOUS VARIABLES



No Anomalies present with distribution of these variables.

1. All variables does have a few extreme values, but I would consider them to account for more information

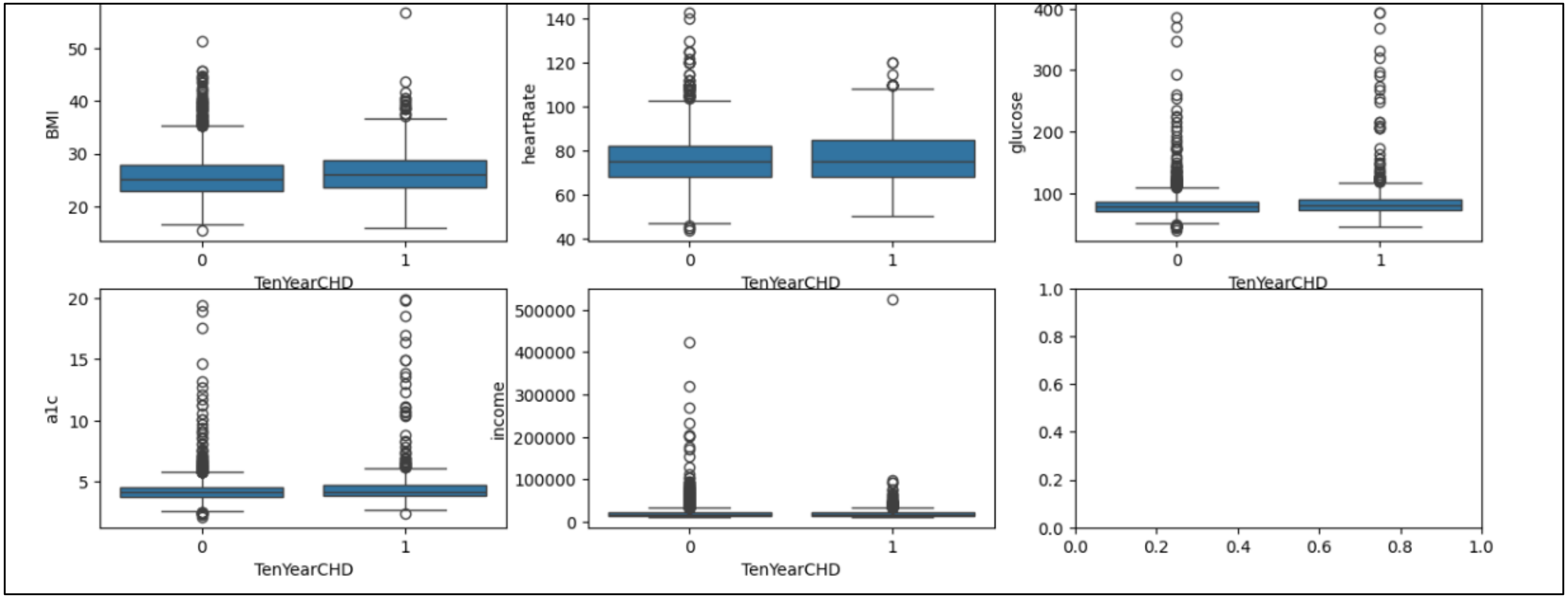
BIVARIATE ANALYSIS
CONTINUOUS VARIABLES VS RESPONSE



In individuals prone to heart disease, median values of age, (CigPerDay), (SysBP), and (DiBP) tend to be higher, indicating a potential correlation between these variables and increased disease susceptibility.

BIVARIATE ANALYSIS

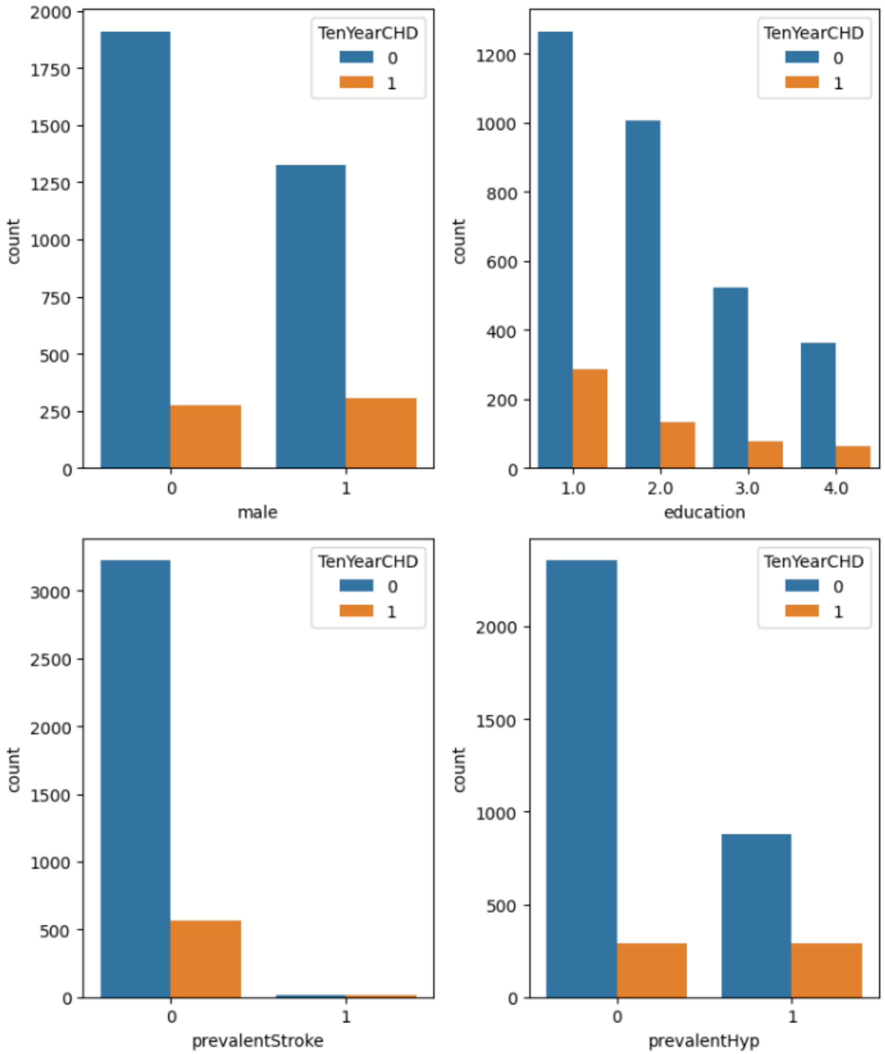
CONTINUOUS VARIABLES VS RESPONSE



From these Box lot , we can say there is definitive correlation between response and these variables as the median values are in the same range for both classes.

BIVARIATE ANALYSIS

CATEGORICAL VARIABLES VS RESPONSE

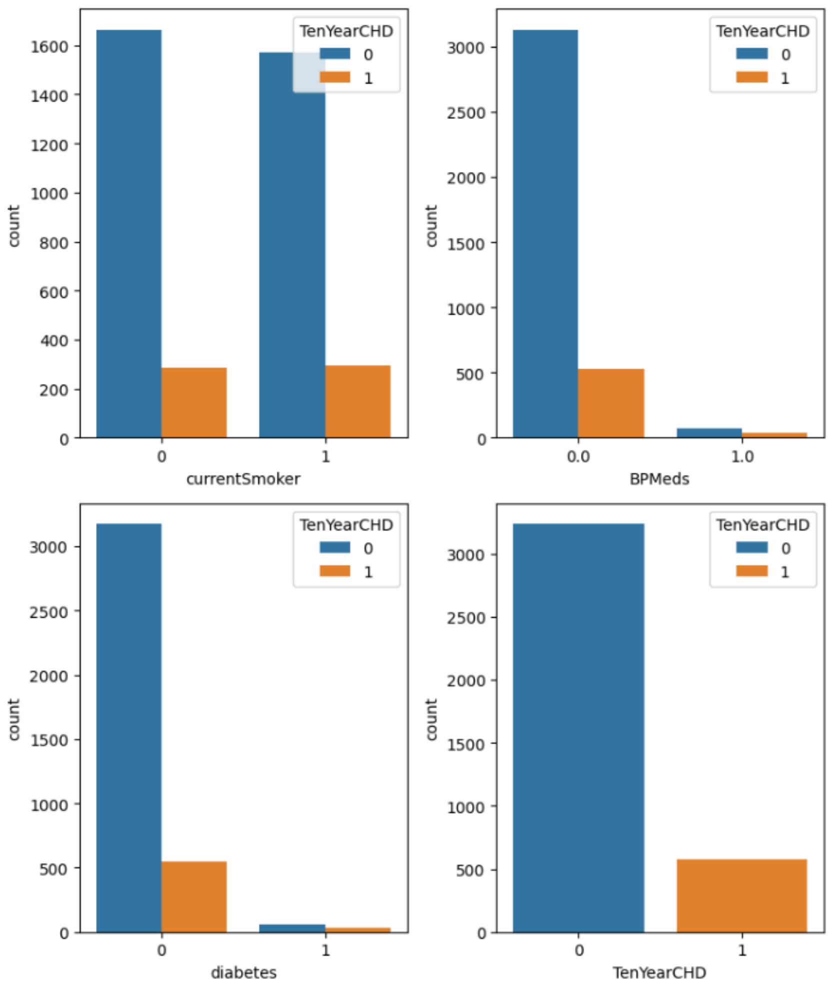


From these Bar charts , we can say that:

1. The percentage of people prone to heart disease is high when that individual is a male or if that individual has prevalentHyp

BIVARIATE ANALYSIS

CATEGORICAL VARIABLES VS RESPONSE

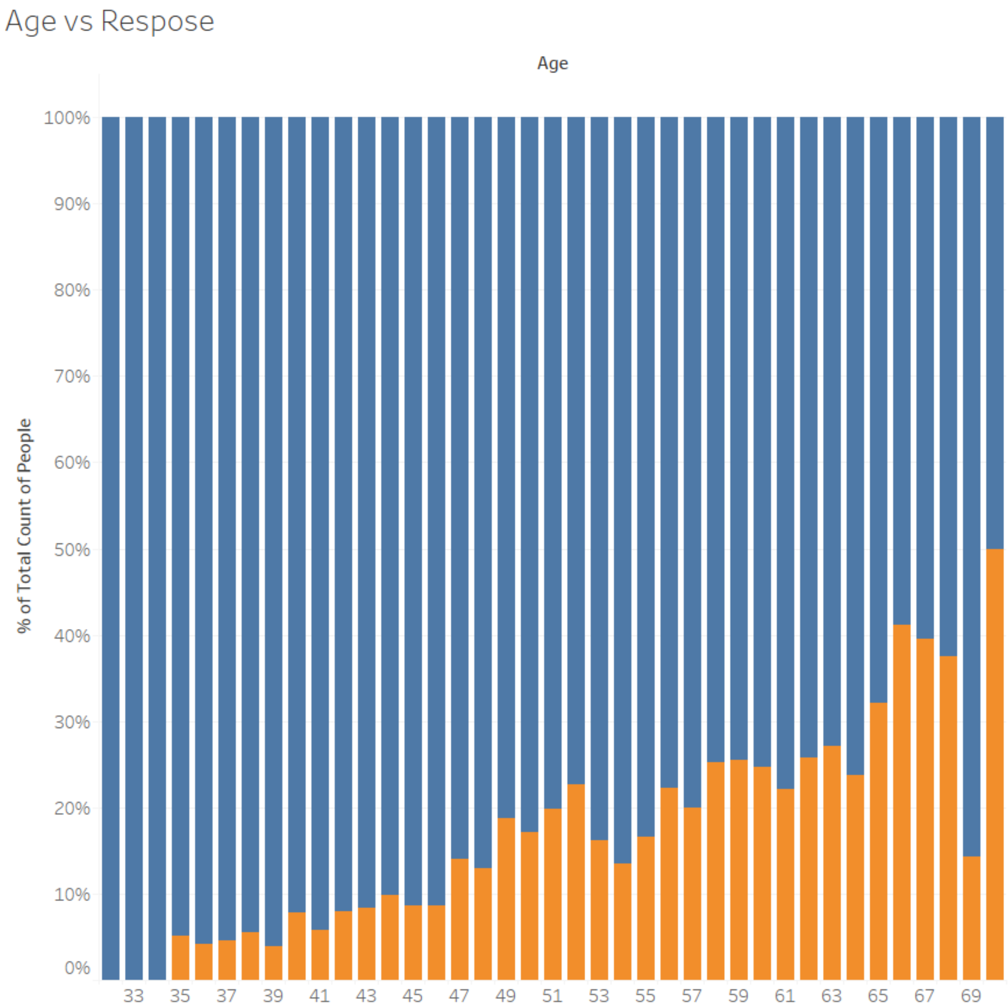


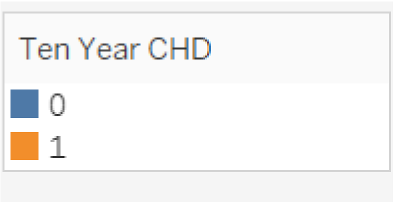
From these Bar charts , we can say that:

1. Current smoker has no direct correlation with response variable as all.

BIVARIATE ANALYSIS

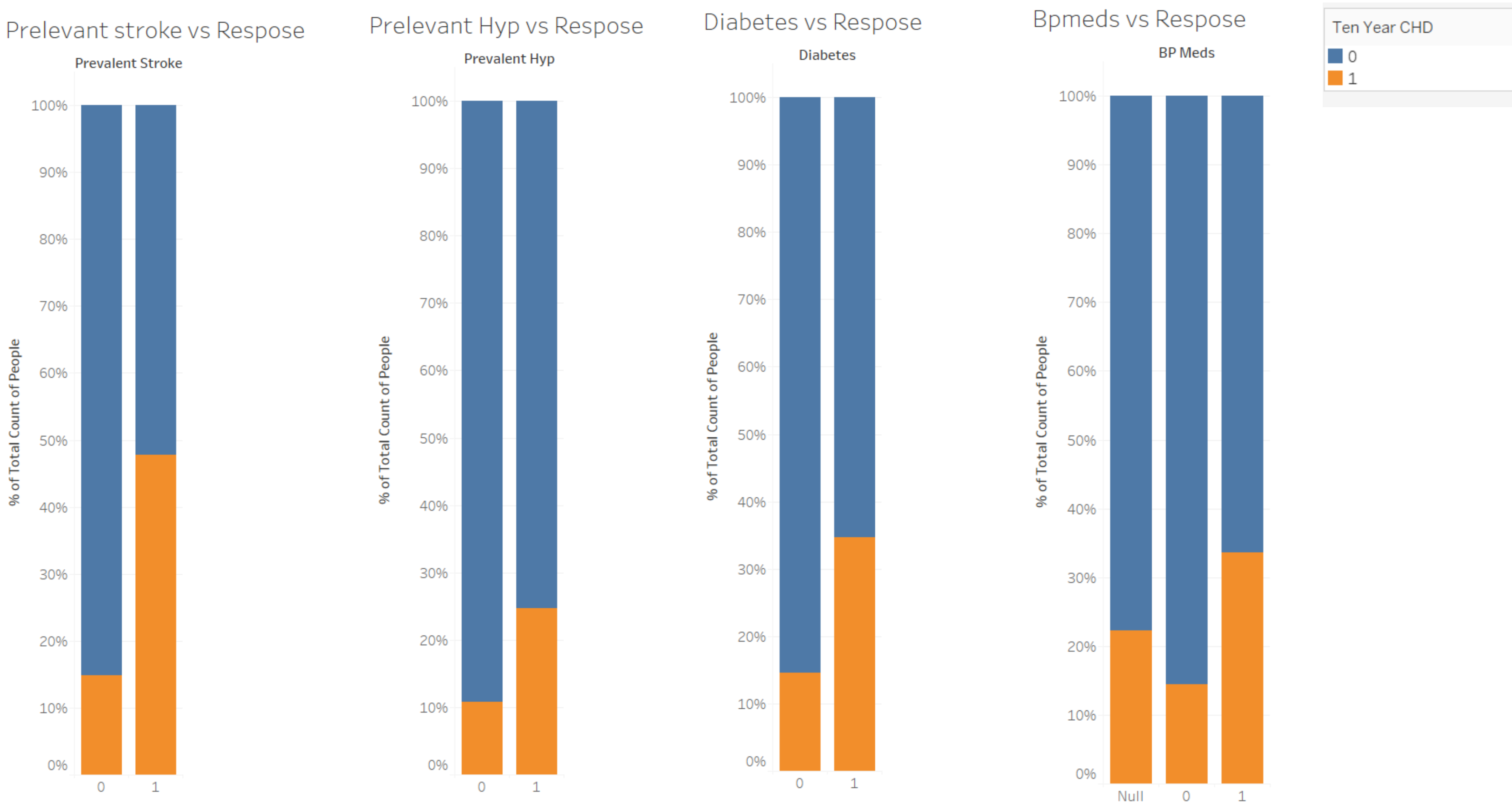
IN-DEPTH ANALYSIS





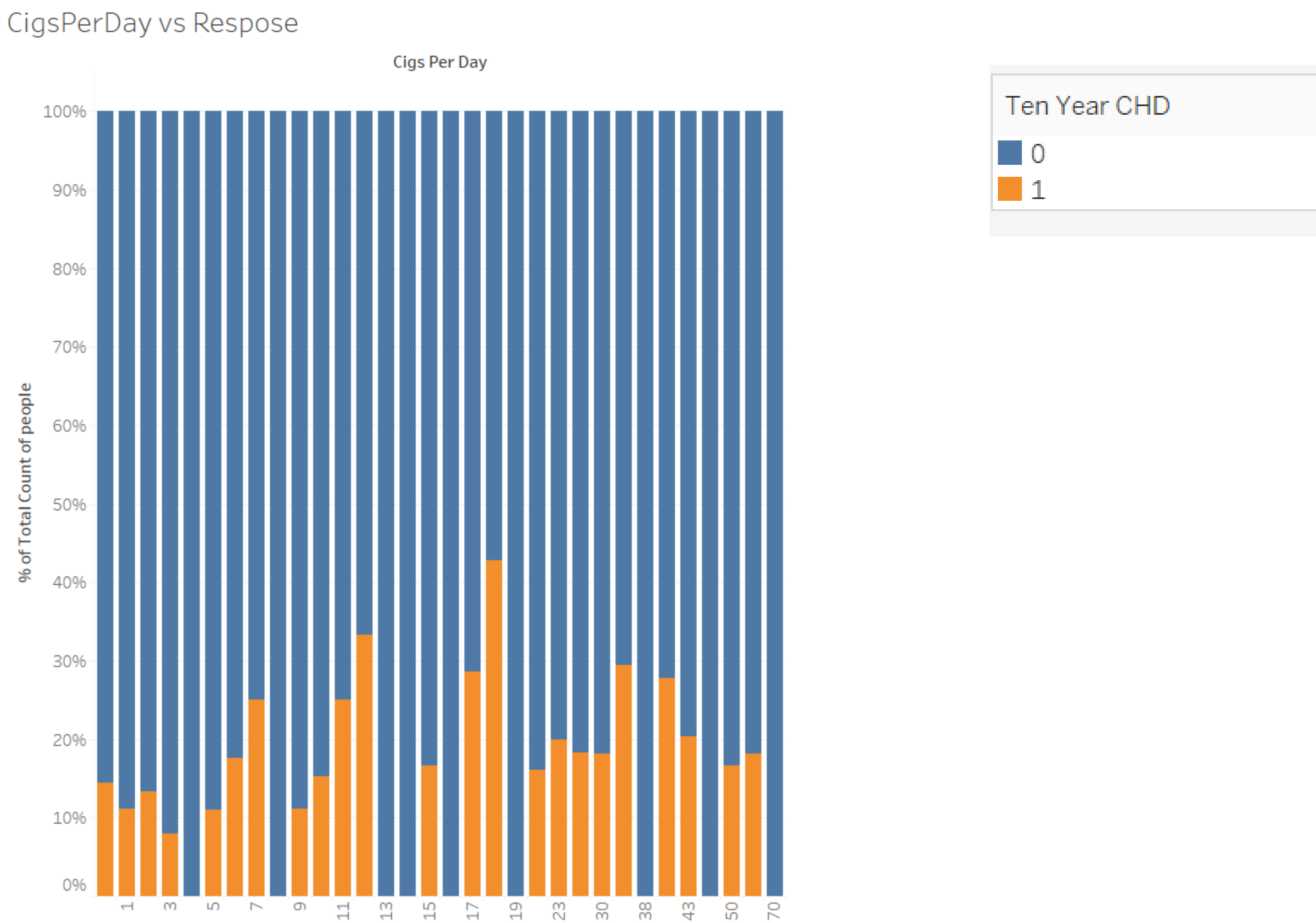
This visual shows clear elation between age and response variable i.e
Older the individual, more are the changes of getting a heart disease.

BIVARIATE ANALYSIS
IN-DEPTH ANALYSIS

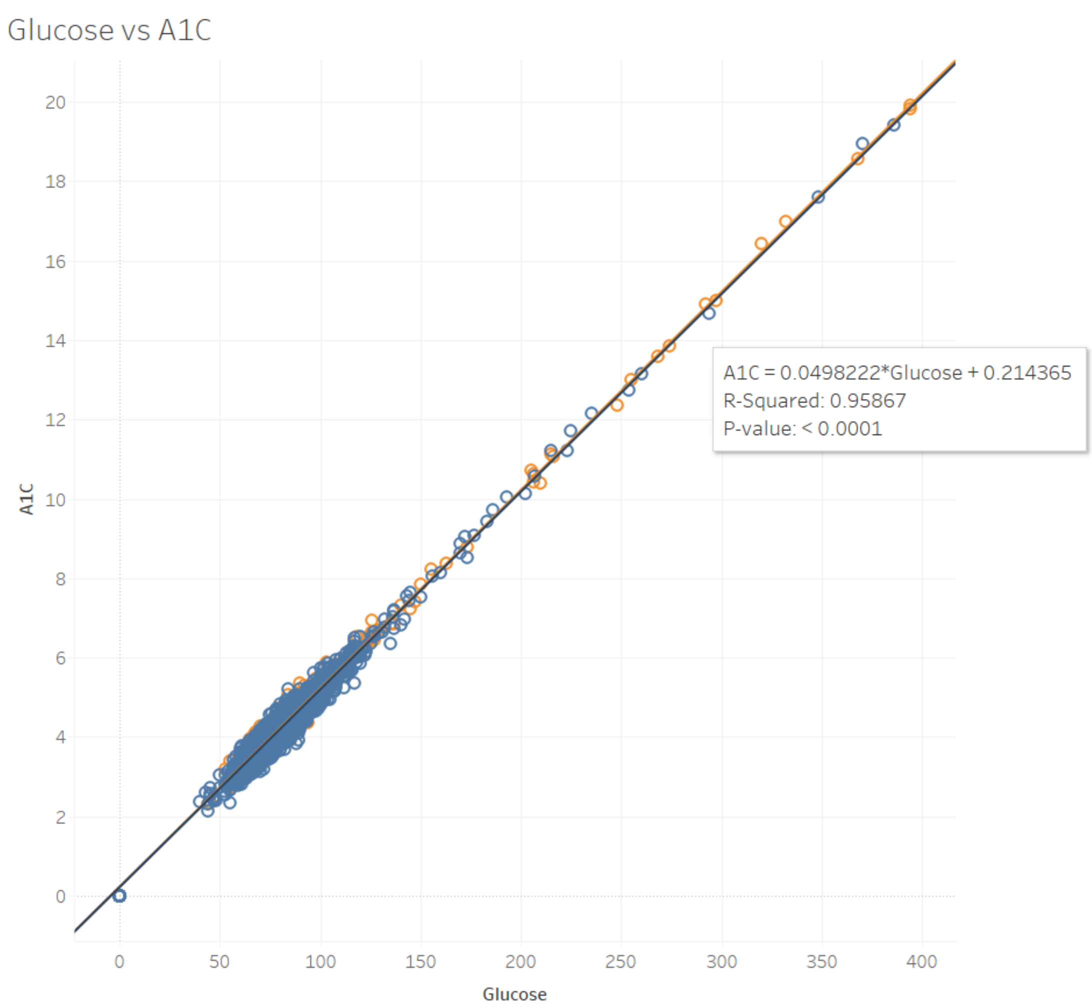


All these variables are more related to the positive outcome (1) of the response variable

BIVARIATE ANALYSIS
IN-DEPTH ANALYSIS

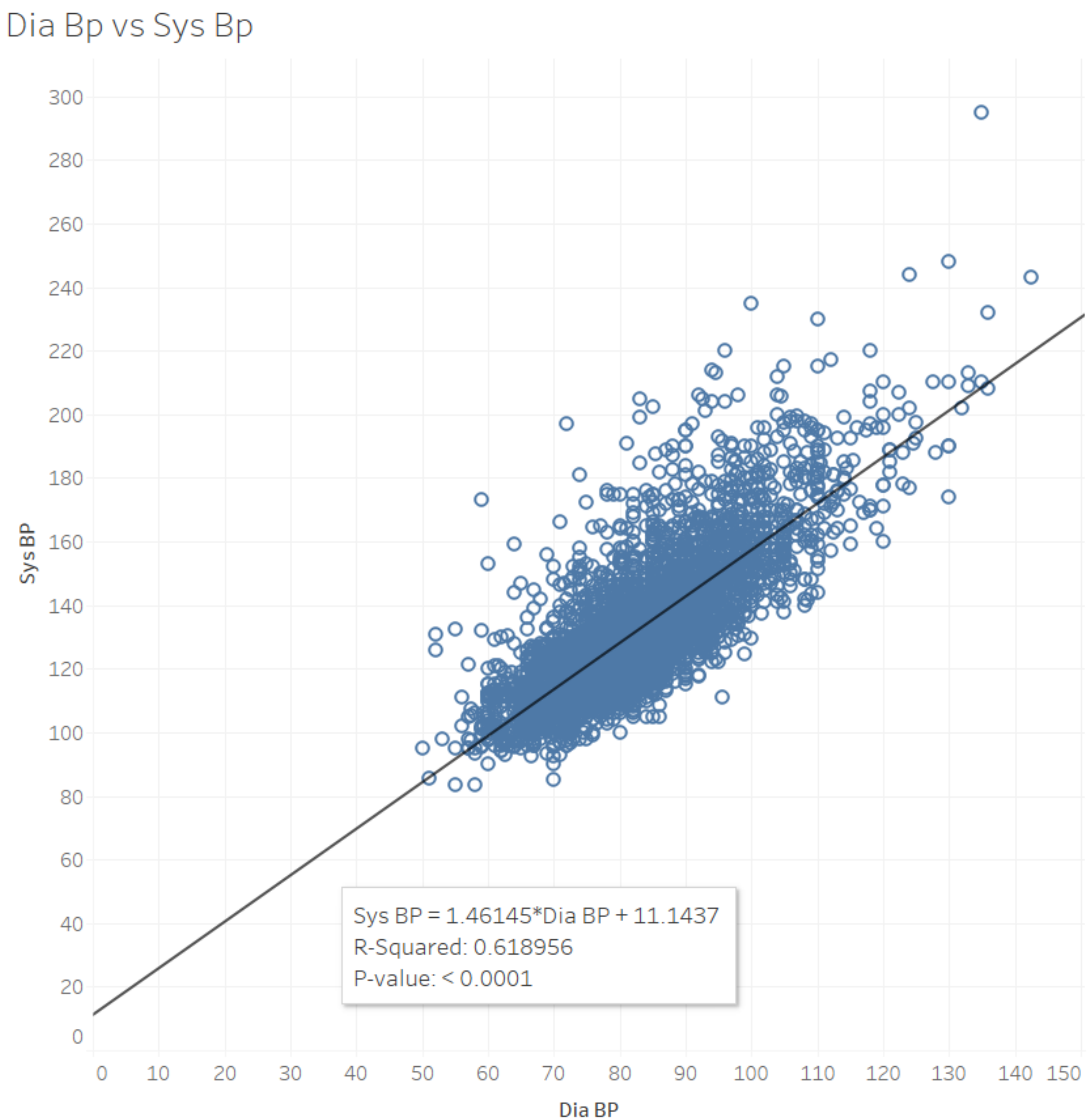


BIVARIATE ANALYSIS
OTHER PAIRS OF VARIABLES



High correlation between both the variables

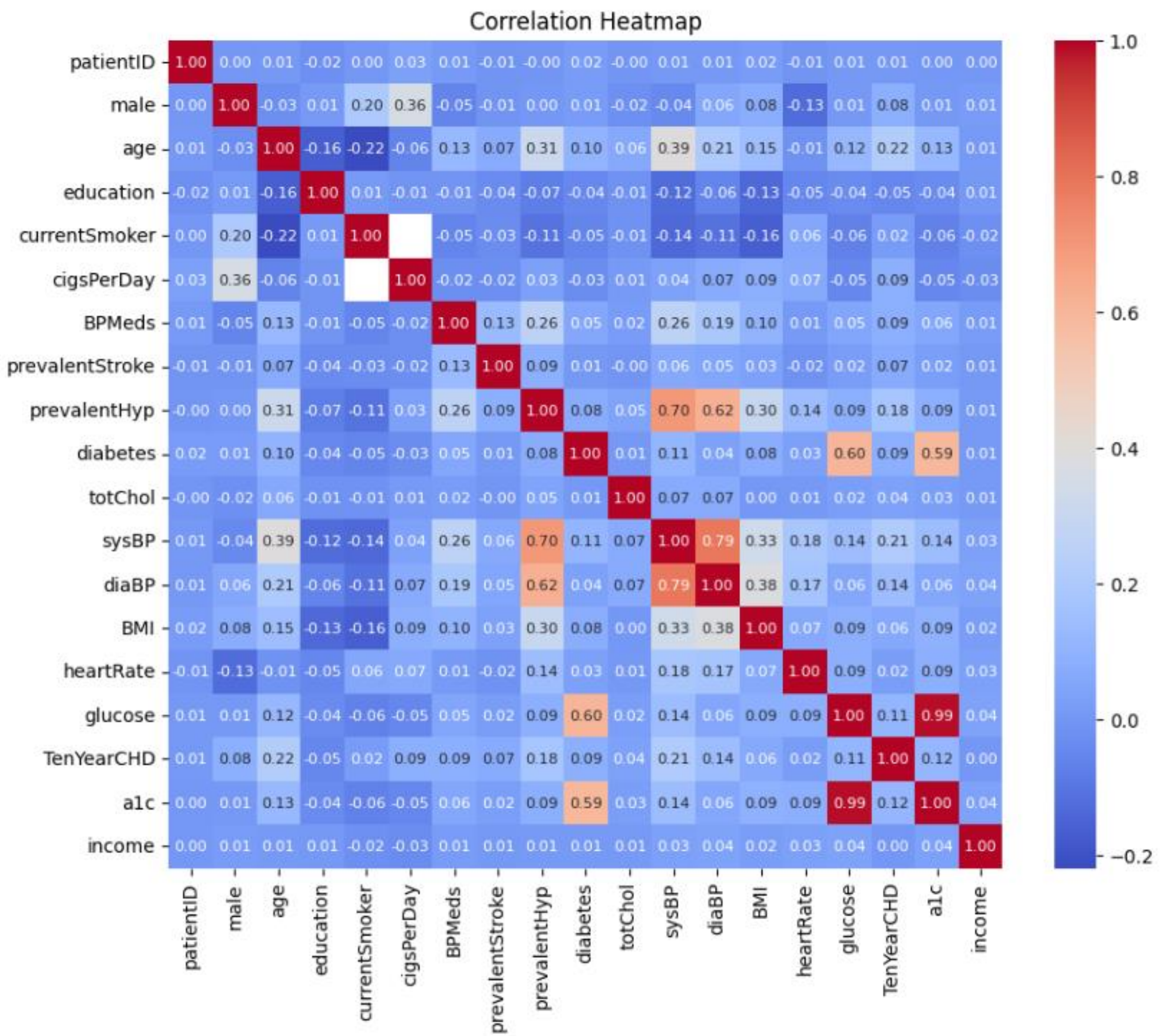
BIVARIATE ANALYSIS
OTHER PAIRS OF VARIABLES



USC Viterbi School of Engineering

BIVARIATE ANALYSIS

CORRELATION MATRIX HEATMAP



DATA PREPARATION PLAN

OVERVIEW

- Data quality issues and actions
- Feature selection decisions
- Feature engineering decisions
- Dataset partitioning decisions

DATA PREPARATION PLAN

DATA QUALITY ISSUES AND ACTIONS

Justification 1- CigsPerDay (2)

Current Smoker	Cigs Per Day	
0	Null	1,948
1	Null	27
	1	54
	2	15
	3	87
	4	9
	5	109
	6	17
	7	12
	8	10
	9	116
	10	124
	11	4
	12	3
	13	3
	14	2

removing rows with NA

Ten Year CHD	Current Smoker	Cigs Per Day	
0	0	Null	1,664
	1	Null	25
1	0	Null	284
	1	Null	2

Missing variables

CigPer day= 1975

1. Interestingly if we compare current smoker and cigs per day we get a relation.
2. I have imputed cigperday=0 for all the rows where current smoker=0
3. For currentsmoker =1; I have dropped all null values i.e 27 rows. This might look like a lot but I am losing on 2 rows of data with contain class 1 in response variable

DATA PREPARATION PLAN

DATA QUALITY ISSUES AND ACTIONS

Missing variables

Education = 93

1. Created a new class with label 0
2. For ['totChol', 'BMI', 'heartRate', 'a1c', 'glucose’]. I have imputed the value with median values.
3. For BpMeds , due to ambiguity of a categorical variable I have dropped all the rows with Na values. However, I am losing only 10 rows of class 1 in response variable

Justification-Bpmeds

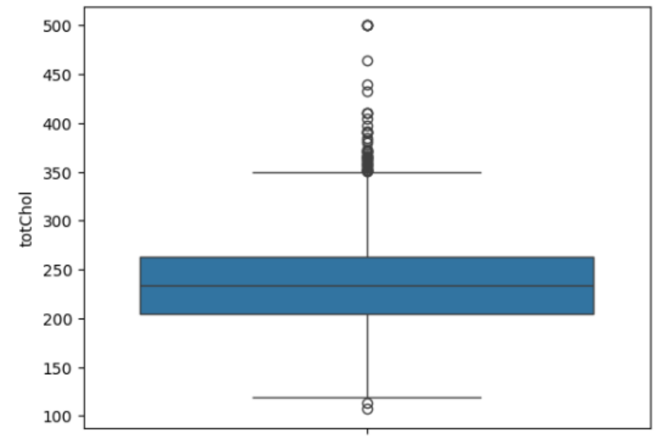
BP Meds	Ten Year ..	
Null	0	35
	1	10
0	0	3,129
	1	532
1	0	73
	1	37

DATA PREPARATION PLAN

DATA QUALITY ISSUES AND ACTIONS

Outliers and heavily skewed

1. Clipping ‘totchol’ to upper limit of 500, I achieved this number by doing some study and taking to professional experts n this field.



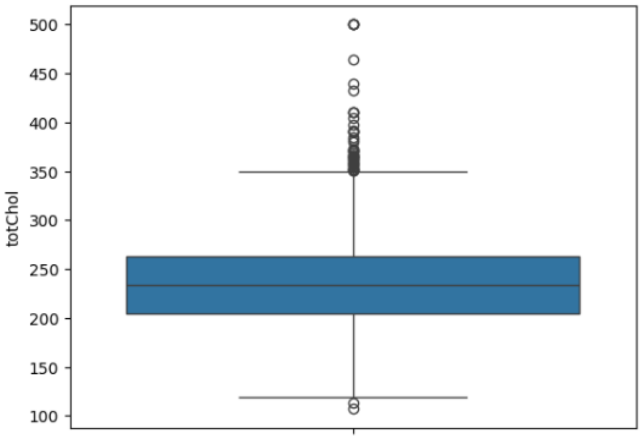

```python
from scipy.stats import skew

for col in df.columns:
  if len(df[col].value_counts())>4:
    skew=df[col].skew()
    if skew>5:
      print(f'Skew before:  {col}={skew}')
      df[col]=np.log(df[col]+1)
      skew1=df[col].skew()
      print(f'Skew aftr log tranformation :  {col}={skew1}')
```

```
Skew before:  glucose=6.424396090382674
Skew aftr log tranformation :  glucose=2.235733475215119
Skew before:  a1c=6.218680763561282
Skew aftr log tranformation :  a1c=2.128896664034904
Skew before:  income=13.27484781197639
Skew aftr log tranformation :  income=2.0332803535759627
```

DATA PREPARATION PLAN

DATA QUALITY ISSUES AND ACTIONS

› Include the graphs before and after transformation



DATA PREPARATION PLAN

SPLIT TRAIN DATASET

Since the dataset is imbalanced I have use stratifies sampling for even distribution of classes in both test and train dataset.

Test split used is 20%

Test Train

```
[340]
# test train split
X = df.drop(columns=['TenYearCHD'])
y = df['TenYearCHD']

from sklearn.model_selection import train_test_split
train,test =train_test_split(df,test_size=0.2,stratify=y)

```

```
[341]
train['TenYearCHD'].value_counts()/len(train['TenYearCHD'])
```

```
TenYearCHD
0    0.848296
1    0.151704
Name: count, dtype: float64
```

```
test['TenYearCHD'].value_counts()/len(test['TenYearCHD'])
```

```
TenYearCHD
0    0.848168
1    0.151832
Name: count, dtype: float64
```

These two values show that split of classes is equal in both test and train

DATA PREPARATION PLAN

DATA PREPROCESSING

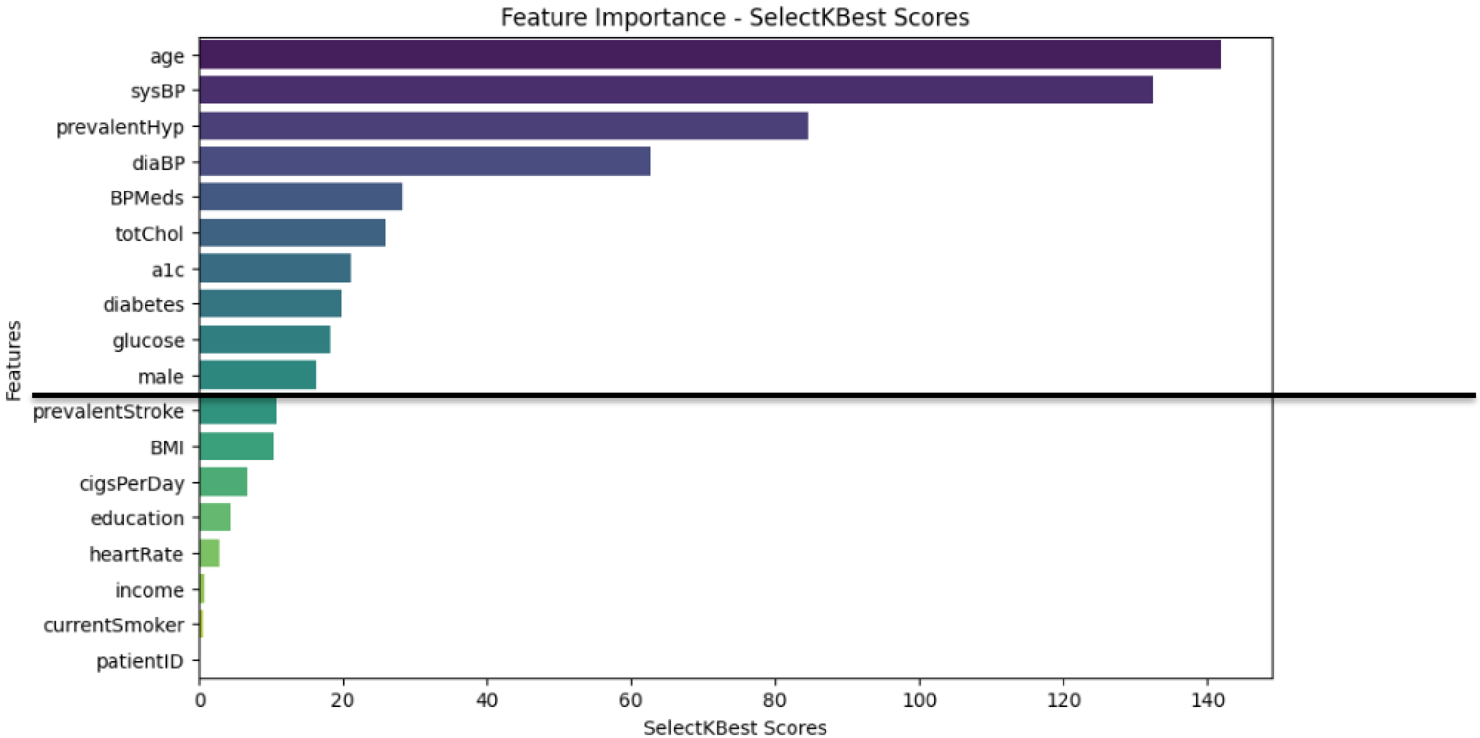
- Performing scaling operating using standard scalar .
- Standard scaling ensures all features have similar scales, promoting better model performance and convergence, particularly with algorithms like Logistic regression and KNN
- However, standardizing the features is not required for tree-based model(RF, Decisions tress and gradient boost).Nevertheless appling scaling on these would make any significant difference to the output

DATA PREPARATION PLAN

FEATURE SELECTION DECISIONS

For feature selection in my analysis, I utilized the SelectKBest method provided by the scikit-learn library. This method selects the top k features based on univariate statistical tests, such as ANOVA F-value, chi-squared, or mutual information.

I used the top 10 contributing features for my analysis.



DATA PREPARATION PLAN

FEATURE ENGINEERING DECISIONS

1. Log transform on income, glucose and a1c , due to positive skew

```
from scipy.stats import skew

for col in df.columns:
  if len(df[col].value_counts())>4:
    skew=df[col].skew()
    if skew>5:
      print(f'Skew before:  {col}={skew}')
      df[col]=np.log(df[col]+1)
      skew1=df[col].skew()
      print(f'Skew aftr log tranformation :  {col}={skew1}')
```

```
Skew before:  glucose=6.424396090382674
Skew aftr log tranformation :  glucose=2.235733475215119
Skew before:  a1c=6.218680763561282
Skew aftr log tranformation :  a1c=2.128896664034904
Skew before:  income=13.27484781197639
Skew aftr log tranformation :  income=2.0332803535759627
```

DATA PREPARATION PLAN

ADDRESSING UNBALANCED DATASET DECISIONS

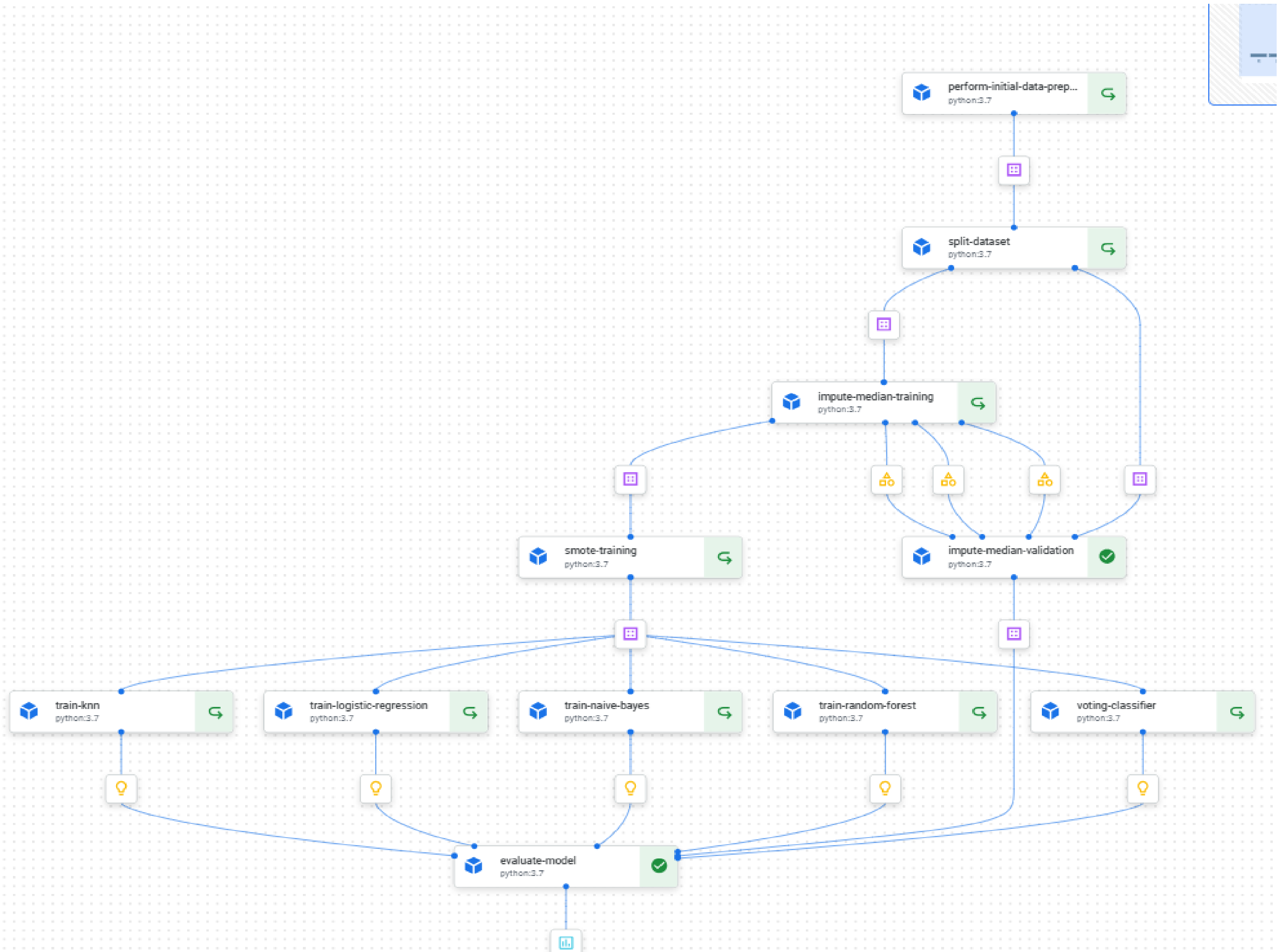
- Perform **Border Line SMOTE** oversampling due to unbalanced dataset.
- I have run my analysis with different combination of Over sampling techniques using

1. Radom under sampling(sampling_strategy=0.2) followed by SMOTE
2. Random Oversampling
3. SMOTE
4. BorderlineSMOTE
5. SVM SMOTE
6. K-Means SMOTE

From my results BorderlineSMOTE beats the other methods by a good margin while SMOTE, SVM SMOTE and Random Oversampling are relatively the same

MODEL PIPELINE

OVERALL PIPELINE



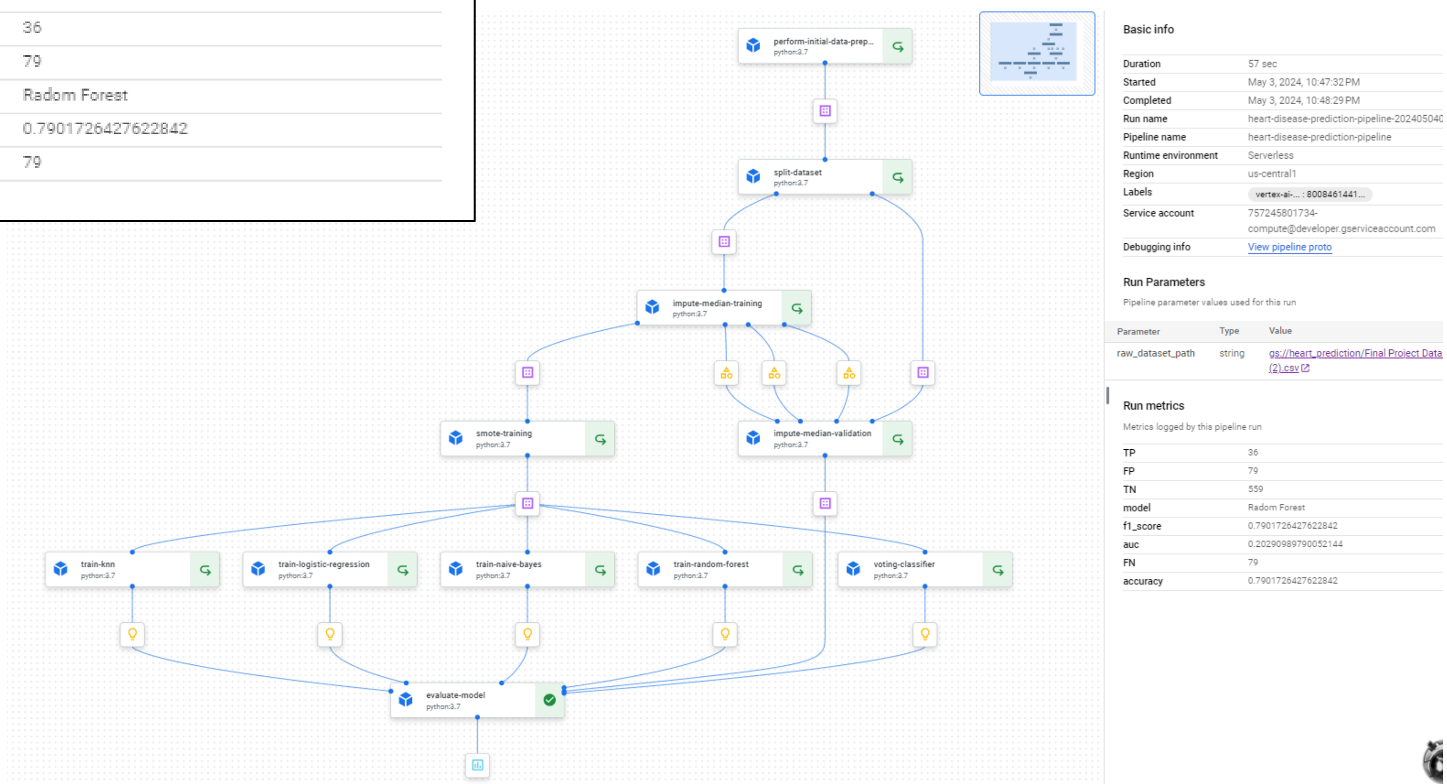
MODEL PIPELINE

MODEL EVALUATION RESULTS

Run metrics

Metrics logged by this pipeline run

f1_score	0.7901726427622842
auc	0.20290989790052144
TN	559
TP	36
FP	79
model	Radom Forest
accuracy	0.7901726427622842
FN	79



MODEL PIPELINE

PIPELINE DEFINITION CODE

```
# Define pipeline
from kfp.v2.dsl import pipeline, Output, Dataset

@pipeline(name="Heart Disease Prediction Pipeline")
def heart_disease_prediction_pipeline(raw_dataset_path: str):

    # Perform initial data preparation
    preprocess_task = perform_initial_data_preparation(input_dataset_path=raw_dataset_path)

    # Split dataset
    split_result = split_dataset(input_dataset_path=preprocess_task.output)

    # Process training dataset - impute median , Features , scaling
    training_data_preparation = impute_median_training(training_dataset_path=split_result.outputs['train_data_path'])

    # Process validation dataset - impute median , Features , scaling
    validation_data_preparation = impute_median_validation(validation_dataset_path=split_result.outputs['validation_data_path'],
                                                           median_path=training_data_preparation.outputs['median'],
                                                           scaler_path=training_data_preparation.outputs['scaler_path'],
                                                           FS_dataset_path=training_data_preparation.outputs['features'])
   # Oversampling-SMOTE
    oversampling_task = Smote_training(training_dataset_path=training_data_preparation.outputs['imputed_dataset_path'])

      # Train models
    train_lr_task = train_logistic_regression(training_dataset_path=oversampling_task.outputs['OS_dataset_path'])
    train_knn_task = train_knn(training_dataset_path=oversampling_task.outputs['OS_dataset_path'])
    train_rf_task = train_random_forest(training_dataset_path=oversampling_task.outputs['OS_dataset_path'])
    train_nb_task = train_naive_bayes(training_dataset_path=oversampling_task.outputs['OS_dataset_path'])
    train_voting_task = voting_classifier(training_dataset_path=oversampling_task.outputs['OS_dataset_path'])

    # Evaluate Models
    evaluate_models_task = evaluate_model(
      test_dataset_path=validation_data_preparation.outputs['imputed_validation_dataset_path'],
      knn_model=train_knn_task.outputs['trained_model_artifact'],
      rf_model=train_rf_task.outputs['trained_model_artifact'],
      nb_model=train_nb_task.outputs['trained_model_artifact'],
      voting_model=train_voting_task.outputs['voting_model_artifact'],
      lr_model=train_lr_task.outputs['trained_model_artifact']
    )
```

COMPONENT DEFINITION PERFORM_INITIAL_DATA_PREPARATION

Common dataset preparation steps

```python
from kfp.v2.dsl import component, InputPath, OutputPath
@component(packages_to_install=["pandas", "numpy", "fsspec", "gcsfs"])
def perform_initial_data_preparation(input_dataset_path: str,
                                      output_dataset_path: OutputPath('Dataset')):
    import pandas as pd
    import numpy as np

    df = pd.read_csv(input_dataset_path)

    # Filling all the Nan value of cigsPerDay with zero for the rows with current Smoker=0
    df.loc[df['currentSmoker']==0,['cigsPerDay']]=df.loc[df['currentSmoker']==0,['cigsPerDay']].fillna(0)


    # create a new label of 0 for all the NA values in education
    df['education']=df['education'].fillna(0)


    # Clip the column to remove outliers
    clipped_column = df['totChol'].clip( upper=500)

    # Replace the original column with the clipped column
    df['totChol']=clipped_column

    #applying log Tranfomation
    col=['glucose','income','a1c']

    for col in col:
      df[col]=np.log(df[col]+1)

    df.to_csv(output_dataset_path, index=False)
```

COMPONENT DEFINITION IMPUTE_VALUES_TRAINING

In this Component I am preforming three data prepressing steps:

1. Imputing the Na values with Median values of that columns

Imputing values

```
[ ] from kfp.v2.dsl import Output
    from kfp.v2.dsl import Artifact

    @component(packages_to_install=["pandas", "joblib","scikit-learn","imbalanced-learn==0.11.0"])
    def impute_median_training(training_dataset_path: InputPath('Dataset'),
                       imputed_dataset_path: OutputPath('Dataset'),
                       scaler_path: OutputPath('Artifact'),
                       median: OutputPath('Artifact'),
                               features: OutputPath('Artifact')):

        from sklearn.preprocessing import StandardScaler, MinMaxScaler
        import joblib
        import pandas as pd
        import numpy as np
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import f_regression


        # Load the training dataset
        df = pd.read_csv(training_dataset_path)

        median_values = {}

        for column in ['totChol', 'BMI', 'heartRate', 'a1c', 'glucose']:
            med = df[column].median()
            df[column] = df[column].fillna(med)
            median_values[column] = med

        median_df = pd.DataFrame(median_values.items(), columns=['Column', 'Median'])

        # drop the remaining Na values

        df.dropna(inplace=True)
```

COMPONENT DEFINITION
IMPUTE_VALUES_TRAINING

2. Performing standardization of columns using Standard scaler

```
# Create a scaler object
target_column = df['TenYearCHD']
features_to_scale = df.drop('TenYearCHD', axis=1)

# Apply StandardScaler to the features
scaler = StandardScaler()
scaled_features_array = scaler.fit_transform(features_to_scale)
scaled_features_df = pd.DataFrame(scaled_features_array, columns=features_to_scale.columns)
```

COMPONENT DEFINITION
IMPUTE_VALUES_TRAINING

3. Using SelectKbest algorithm to find the best features

```
# Step 1: Initialize SelectKBest with the desired scoring function
selector = SelectKBest(score_func=f_regression, k=10)

# Step 2: Fit the selector to your data
X_new = selector.fit_transform(scaled_features_df, target_column)

# Step 3: Get the selected feature indices
selected_features_indices = selector.get_support(indices=True)

# Step 4: Get the names of the selected features
selected_features_names = list(scaled_features_df.columns[selected_features_indices])

# Step 5: Save the selected feature names to an artifact
joblib.dump(selected_features_names, features)

# Save the selected features dataset to the output path
X_selected = scaled_features_df[selected_features_names]


# Combine the scaled features with the target column
result_df = pd.concat([X_selected, target_column], axis=1)

# Save the model to the designated output path
joblib.dump(scaler, scaler_path)

# Save the normalized dataframe to the output path
result_df.to_csv(imputed_dataset_path, index=False)

# Save the median dataframe to the output path
median_df.to_csv(median, index=False)
```

COMPONENT DEFINITION IMPUTE_VALUES_VALIDATION

USC Viterbi School of Engineering

imputing Vaidation Dataset

```
@component(packages_to_install=["pandas", "numpy", "scikit-learn", "scipy", "joblib"])
def impute_median_validation(
        validation_dataset_path: InputPath('Dataset'),
        median_path: InputPath('Artifact'),  # medians from training
        scaler_path: InputPath('Artifact'),  # scaler from training
        imputed_validation_dataset_path: OutputPath('Dataset'),
        FS_dataset_path: InputPath('Artifact')):


    import pandas as pd
    import numpy as np
    from sklearn.preprocessing import StandardScaler
    import joblib


    # Load the validation dataset
    df = pd.read_csv(validation_dataset_path)

    (variable) median_df: DataFrame training dataset
    median_df = pd.read_csv(median_path)

    # Fill in the missing values with the median values
    # Iterate over columns in the test dataset
    for column in median_df['Column']:
        # Retrieve the median value for the current column
        median_value = median_df.loc[median_df['Column'] == column, 'Median'].values[0]
        # Fill missing values in the test dataset with the median value
        df[column] = df[column].fillna(median_value)

    # Drop the remaining missing values
    df.dropna(inplace=True)
```

COMPONENT DEFINITION
IMPUTE_VALUES_VALIDATION

```
# Load the scaler
scaler = joblib.load(scaler_path)

y_test = df['TenYearCHD']
X_test = df.drop(columns=['TenYearCHD'])


X_test = X_test.rese (variable) scaler: Any
X_test_scaled_array=scaler.transform(X_test)

X_test_scaled=pd.DataFrame(X_test_scaled_array, columns=X_test.columns)

df=pd.concat([X_test_scaled, y_test], axis=1)


# Load the list of selected feature names from the training dataset
selected_features_names = joblib.load(FS_dataset_path)

# Select the same features in the test dataset as selected in the training dataset
selected_test_df =df[selected_features_names]

# Save the imputed dataframe to the output path
selected_test_df.to_csv(imputed_validation_dataset_path, index=False)
```

COMPONENT DEFINITION
OVERSAMPLING - TRAINING

SMote training

```
@component(packages_to_install=["pandas", "joblib", "scikit-learn","imbalanced-learn==0.11.0"])
def Smote_training(training_dataset_path: InputPath('Dataset'),
                        OS_dataset_path: OutputPath('Dataset')
                       ):

    import pandas as pd
    import joblib

    # Load the training dataset
    df = pd.read_csv(training_dataset_path)

    y_train = df['TenYearCHD']
    X_selected = df.drop(columns=['TenYearCHD'])

    from imblearn.over_sampling import RandomOverSampler, SMOTE, ADASYN
    from imblearn.over_sampling import BorderlineSMOTE
    from imblearn.under_sampling import RandomUnderSampler

    # undersampling using Random Oversampler
    rus = RandomUnderSampler(sampling_strategy=0.5)
    X_rus, y_rus = rus.fit_resample(X_selected, y_train)

    # Oversampling using BorderlineSMOTE
    smote =  BorderlineSMOTE(random_state=42,kind = 'borderline-2')
    X_smote, y_smote = smote.fit_resample(X_selected, y_train)

    oversampled_df = pd.concat([X_smote, y_smote], axis=1)

    oversampled_df.to_csv(OS_dataset_path, index=False)
```

COMPONENT DEFINITION TRAIN_LOGISTIC_REGRESSION

train logistic

```
from kfp.v2.dsl import Output
from kfp.v2.dsl import Artifact
from kfp.v2.dsl import Model
from kfp.v2.dsl import Model
from kfp.v2.dsl import Input
from kfp.v2.dsl import InputPath
from kfp.v2.dsl import OutputPath
from kfp.v2.dsl import component

@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def train_logistic_regression(training_dataset_path: InputPath('Dataset'),
                              trained_model_artifact: Output[Model]):

    import pandas as pd
    from sklearn.linear_model import LogisticRegression
    import joblib
    import os

    # Load the training data
    train_df = pd.read_csv(training_dataset_path)

    X_train = train_df.drop('TenYearCHD', axis=1)
    y_train = train_df['TenYearCHD']

    trained_model = LogisticRegression(max_iter=1000)
    trained_model.fit(X_train, y_train)

    # Save the model to the designated gcs output path
    os.makedirs(trained_model_artifact.path, exist_ok=True)
    joblib.dump(trained_model, os.path.join(trained_model_artifact.path, "model.joblib"))
```

COMPONENT DEFINITION

TRAIN_KNN

```
@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def train_knn(training_dataset_path: InputPath('Dataset'),
              trained_model_artifact: Output[Model]):

    import pandas as pd
    from sklearn.neighbors import KNeighborsClassifier
    import joblib
    import os

    # Load the training data
    train_df = pd.read_csv(training_dataset_path)

    X_train = train_df.drop('TenYearCHD', axis=1)
    y_train = train_df['TenYearCHD']

    trained_model = KNeighborsClassifier()
    trained_model.fit(X_train, y_train)

    # Save the model to the designated gcs output path
    os.makedirs(trained_model_artifact.path, exist_ok=True)
    joblib.dump(trained_model, os.path.join(trained_model_artifact.path, "model.joblib"))
```

COMPONENT DEFINITION

TRAIN_RANDOM FOREST

```
@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def train_random_forest(training_dataset_path: InputPath('Dataset'),
                        trained_model_artifact: Output[Model]):

    import pandas as pd
    from sklearn.ensemble import RandomForestClassifier
    import joblib
    import os

    # Load the training data
    train_df = pd.read_csv(training_dataset_path)

    X_train = train_df.drop('TenYearCHD', axis=1)
    y_train = train_df['TenYearCHD']

    trained_model = RandomForestClassifier()
    trained_model.fit(X_train, y_train)

    # Save the model to the designated gcs output path
    os.makedirs(trained_model_artifact.path, exist_ok=True)
    joblib.dump(trained_model, os.path.join(trained_model_artifact.path, "model.joblib"))
```

COMPONENT DEFINITION
TRAIN_NAIVE BAYES

```
@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def train_naive_bayes(training_dataset_path: InputPath('Dataset'),
                      trained_model_artifact: Output[Model]):

    import pandas as pd
    from sklearn.naive_bayes import GaussianNB
    import joblib
    import os

    # Load the training data
    train_df = pd.read_csv(training_dataset_path)

    X_train = train_df.drop('TenYearCHD', axis=1)
    y_train = train_df['TenYearCHD']

    trained_model = GaussianNB()
    trained_model.fit(X_train, y_train)

    # Save the model to the designated gcs output path
    os.makedirs(trained_model_artifact.path, exist_ok=True)
    joblib.dump(trained_model, os.path.join(trained_model_artifact.path, "model.joblib"))
```

COMPONENT DEFINITION

TRAIN_VOTING CLASSIFIER USING GRIDSEARCH

```
@component(packages_to_install=["scikit-learn", "joblib"])
def voting_classifier(training_dataset_path: InputPath('Dataset'),
                       voting_model_artifact: Output[Model]):

    import pandas as pd
    from sklearn.naive_bayes import GaussianNB
    from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
    from sklearn.naive_bayes import GaussianNB
    from sklearn.model_selection import train_test_split, GridSearchCV
    import joblib
    import os

    # Load the training data
    train_df = pd.read_csv(training_dataset_path)

    X_train = train_df.drop('TenYearCHD', axis=1)
    y_train = train_df['TenYearCHD']



    # Define base models with their respective hyperparameter grids
    rf_model = RandomForestClassifier()
    rf_param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [None, 5, 10]}
    gb_model = GradientBoostingClassifier()
    gb_param_grid = {'n_estimators': [50, 100, 200], 'learning_rate': [0.01, 0.1, 0.5]}
    nb_model = GaussianNB()

    # Perform hyperparameter tuning for each base model using GridSearchCV
    rf_grid_search = GridSearchCV(rf_model, rf_param_grid, cv=3, scoring='f1', n_jobs=-1)
    rf_grid_search.fit(X_train, y_train)
    rf_best_model = rf_grid_search.best_estimator_

    gb_grid_search = GridSearchCV(gb_model, gb_param_grid, cv=3, scoring='f1', n_jobs=-1)
    gb_grid_search.fit(X_train, y_train)
    gb_best_model = gb_grid_search.best_estimator_

    # Create VotingClassifier with tuned models
    voting_clf = VotingClassifier(estimators=[('rf', rf_best_model), ('gb', gb_best_model), ('nb', nb_model)], voting='soft')

    # Train VotingClassifier
    voting_clf.fit(X_train, y_train)

    # Save the voting classifier to the designated gcs output path
    os.makedirs(voting_model_artifact.path, exist_ok=True)
    joblib.dump(voting_clf, os.path.join(voting_model_artifact.path, "model.joblib"))
```

Training a voting classier on RF, NB and gradient boosting.

I am further using grid search to find the best hypermeter combinations

USC Viterbi School of Engineering

COMPONENT DEFINITION
EVALUATE_MODEL

```python
from kfp.v2.dsl import Metrics

@component(packages_to_install=["pandas", "scikit-learn", "joblib"])
def evaluate_model(test_dataset_path: InputPath('Dataset'),
                   knn_model: Input[Model],
                   rf_model: Input[Model],
                   nb_model: Input[Model],
                   voting_model: Input[Model],
                   lr_model:  Input[Model],
                   #svm_model: Input[Model],
                  # gb_model:  Input[Model],
                   #xgb_model: Input[Model],
                   #cat_model: Input[Model],
                   best_model_metrics: Output[Metrics]):

    import pandas as pd
    import joblib
    from sklearn.metrics import accuracy_score, f1_score

    # Load the test dataset
    test_df = pd.read_csv(test_dataset_path)
    y_test = test_df['TenYearCHD']
    X_test = test_df.drop(columns=['TenYearCHD'])


    # Load the trained models
    knn_model_loaded = joblib.load(knn_model.path + "/model.joblib")
    rf_model_loaded = joblib.load(rf_model.path + "/model.joblib")
    nb_model_loaded = joblib.load(nb_model.path + "/model.joblib")
    voting_model_loaded = joblib.load(voting_model.path + "/model.joblib")
    lr_model_loaded = joblib.load(lr_model.path + "/model.joblib")
    #svm_model_loaded = joblib.load(svm_model)
    #gb_model_loaded = joblib.load(gb_model)
    #
```

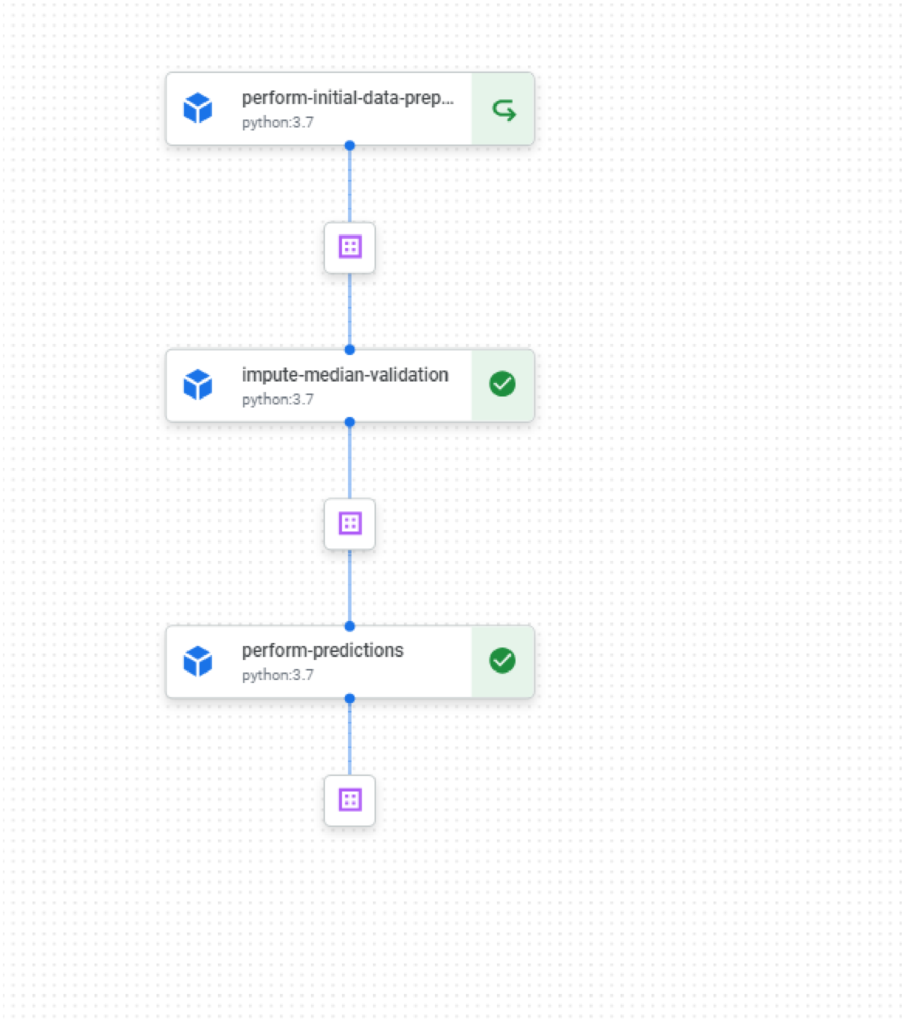
COMPONENT DEFINITION

EVALUATE_MODEL

```
# Make predictions on the test set for each model
knn_pred = knn_model_loaded.predict(X_test)
rf_pred = rf_model_loaded.predict(X_test)
nb_pred = nb_model_loaded.predict(X_test)
voting_pred = voting_model_loaded.predict(X_test)
lr_pred = lr_model_loaded.predict(X_test)
#svm_pred = svm_model_loaded.predict(X_test)
#gb_pred = gb_model_loaded.predict(X_test

# Calculate evaluation metrics for each model
knn_acc = accuracy_score(y_test, knn_pred)
knn_f1 = f1_score(y_test, knn_pred,average='weighted')
rf_acc = accuracy_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred,average='weighted')
nb_acc = accuracy_score(y_test, nb_pred)
nb_f1 = f1_score(y_test, nb_pred,average='weighted')
voting_acc = accuracy_score(y_test, voting_pred)
voting_f1 = f1_score(y_test, voting_pred,average='weighted')
lr_f1=f1_score(y_test,lr_pred,average='weighted')
rf_recall=recall_score(y_test,rf_pred)
rf_precision=precision_score(y_test,rf_pred)
rf_auc1=average_precision_score(y_test,rf_pred)

conf_matrix = confusion_matrix(y_test, rf_pred)

# Extract TP, FP, TN, FN
TN, FP, FN, TP = conf_matrix.ravel()

# Determine the best model based on F1 score
best_model = max([('knn', knn_acc, knn_f1),
                  ('Radom Forest', rf_acc, rf_f1),
                  ('nb', nb_acc, nb_f1),
                  ('voting', voting_acc, voting_f1)],
                 key=lambda x: x[2])

# Log the evaluation metrics of the best model
best_model_metrics.log_metric("accuracy", best_model[1])
best_model_metrics.log_metric("f1_score", best_model[2])
best_model_metrics.log_metric("model", best_model[0])
best_model_metrics.log_metric("auc", rf_auc1)
best_model_metrics.log_metric("TP", int(TP))
best_model_metrics.log_metric("FP",int(FP))
best_model_metrics.log_metric("TN",int(TN))
best_model_metrics.log_metric("FN",int(FN))
```

INFERENCE PIPELINE

PIPELINE VISUALIZATION



INFERENCE PIPELINE

DATA PREPARATION DEFINITION

```
@component(packages_to_install=["pandas", "numpy", "fsspec", "gcsfs"])
def perform_initial_data_preparation(input_dataset_path: str, output_dataset_path: OutputPath(Dataset)):
    import pandas as pd
    import numpy as np

    df = pd.read_csv(input_dataset_path)

    # Filling all the Nan value of cigsPerDay with zero for the rows with current Smoker=0
    df.loc[df['currentSmoker']==0,['cigsPerDay']]=df.loc[df['currentSmoker']==0,['cigsPerDay']].fillna(0)



    # create a new label of 0 for all the NA values in education
    df['education']=df['education'].fillna(0)

    # Clip the column to remove outliers
    clipped_column = df['totChol'].clip( upper=500)

    # Replace the original column with the clipped column
    df['totChol']=clipped_column

    #applying log Tranfomation
    col=['glucose','income','a1c']

    for col in col:
      df[col]=np.log(df[col]+1)


    df.to_csv(output_dataset_path, index=False)
```

INFERENCE PIPELINE

IMPUTING VALUES DEFINITION

Imputing values

```
[ ] @component(packages_to_install=["pandas", "numpy", "scikit-learn", "scipy", "joblib","fsspec", "gcsfs"])
    def impute_median_validation(
            validation_dataset_path: InputPath('Dataset'),
            median_path: str,  # medians from training
            scaler_path: str,  # scaler from training
            imputed_validation_dataset_path: OutputPath('Dataset'),
            FS_dataset_path: str):


        import pandas as pd
        import numpy as np
        from sklearn.preprocessing import StandardScaler
        import joblib
        import gcsfs

        # Load the validation dataset
        df = pd.read_csv(validation_dataset_path)

        # Load the median values from the training dataset
        median_df=pd.read_csv(median_path)


        # Fill in the missing values with the median values
        # Iterate over columns in the test dataset
        for column in median_df['Column']:
            # Retrieve the median value for the current column
            median_value = median_df.loc[median_df['Column'] == column, 'Median'].values[0]
            # Fill missing values in the test dataset with the median value
            df[column] = df[column].fillna(median_value)

        # Drop the remaining missing values
        df.dropna(inplace=True)
    # Load the list of selected feature names from the training dataset
        # Create a GCS file system object
        fs = gcsfs.GCSFileSystem()
```

INFERENCE PIPELINE

IMPUTING VALUES DEFINITION

```
with fs.open(FS_dataset_path, 'rb') as f:
    selected_features_names = joblib.load(f)
#scale_feature=selected_features_names
#selected_features_names.append('patientID')
# Select the same features in the test dataset as selected in the training dataset
selected_test_df=df

# Drop the remaining missing values
selected_test_df.dropna(inplace=True)
# Load the scaler
# Create a GCS file system object
fs = gcsfs.GCSFileSystem()

with fs.open(scaler_path, 'rb') as f:
    scaler = joblib.load(f)

X_test=selected_test_df

X_test_scaled_array=scaler.transform(X_test)

X_test_scaled=pd.DataFrame(X_test_scaled_array, columns=X_test.columns)

selected_test_df.reset_index(drop=True, inplace=True)
df.reset_index(drop=True, inplace=True)

selected_test_df=pd.concat([X_test_scaled[selected_features_names],df['patientID']],axis=1)




# Save the imputed dataframe to the output path
selected_test_df.to_csv(imputed_validation_dataset_path, index=False)
```

INFERENCE PIPELINE
PREDICTIONS DEFINITION

USC Viterbi School of Engineering

Predicitons

```
@component(packages_to_install=["pandas", "numpy", "scikit-learn", "joblib", "fsspec", "gcsfs"])
def perform_predictions(dataset_for_prediction_path: InputPath('Dataset'),
                   model_path: str,
                   predictions_path: OutputPath('Dataset')):

    import pandas as pd
    import joblib
    import gcsfs

    # Create a GCS file system object
    fs = gcsfs.GCSFileSystem()

    # Load the trained model
    with fs.open(model_path, 'rb') as f:
      trained_model = joblib.load(f)

    # Load the test dataset
    pred_df = pd.read_csv(dataset_for_prediction_path)

    final_df=pred_df

    # Drop the patientID column
    pred_df1=pred_df.iloc[:,0:10]


    # Make predictions
    y_pred = trained_model.predict(pred_df1)
    final_df['TenYearCHD'] = y_pred
    final_df=final_df[['patientID','TenYearCHD']]

    # Save the predictions
    final_df.to_csv(predictions_path, index=False)
```

INFERENCE PIPELINE

PIPELINE DEFINITION

Define Pipeline

```
features_path= "gs://heart_prediction/757245801734/heart-disease-prediction-pipeline-20240502233246/impute-median-training_-7069055405124485120/features"
median= "gs://heart_prediction/757245801734/heart-disease-prediction-pipeline-20240502233246/impute-median-training_-7069055405124485120/median"
scaler_path= "gs://heart_prediction/757245801734/heart-disease-prediction-pipeline-20240502233246/impute-median-training_-7069055405124485120/scaler_path"
model= "gs://heart_prediction/757245801734/heart-disease-prediction-pipeline-20240503004309/train-random-forest_8381106066523422720/trained_model_artifact/model.joblib"


@pipeline(name='vs-heart_predictions-inference-pipeline')
def heart_disease_prediction_pipeline(dataset_for_predictions_path: str,
                                features_uri: str = features_path,
                                median_uri:  str = median,
                                scaler_uri:  str = scaler_path,
                                model_uri: str = model):

    # Process dataset - initial data preparation
    initial_prepared_dataset = perform_initial_data_preparation(input_dataset_path=dataset_for_predictions_path)

    # Impute
    imputed_dataset = impute_median_validation(
       validation_dataset_path=initial_prepared_dataset.outputs['output_dataset_path'],
       median_path= median_uri,
       FS_dataset_path=features_uri,
       scaler_path=scaler_uri

    )

    perform_predictions(
        dataset_for_prediction_path=imputed_dataset.outputs['imputed_validation_dataset_path'],
        model_path=model_uri
    )
```

USC Viterbi School of Engineering

INFERENCE PIPELINE

SCREENSHOT OF PREDICTIONS CSV

patientID	TenYearCHD
110399	0
189047	0
957019	0
208967	0
230935	0
216024	0
368834	0
135175	0
294070	0
595710	0
425597	1
650137	0
590019	0
925626	0
276518	0
342284	1
469306	0
197764	0
416488	0
208652	0
562216	0
115448	0
224178	1
271781	0
887721	0
262782	0
583133	0
196173	0
779064	0

SUMMARY DISCUSSION

- Data preprocessing:
  1. Overall, since the data was limited, I wanted impute the nan values as much as possible.
  2. Cigperday contained nearly 2000 NAN. I correlated that with another and imputed most of the values with 0.
  3. Remaining variables were imputed using median as that would be best option oven mean due to extreme values in columns.
- Feature Selection
  1. I used Select bestK and RFE methods to select the best features.
  2. Compared the output with both the features , SeleckbestK gave the best results.
  3. The issue with Select bestK is that it does not take feature interaction into consideration; that is the reason why we have correlating features in our feature set.

SUMMARY DISCUSSION

- Handlining Unbalanced Data
  1. Using SMOTE oversampling is a basic approach. After referring to the SMOTE paper , it was given that it is a better approach to under sample first and then over sample it to desired number.
  2. This method proven to give better results than just SMOTE.
  3. But in my analysis, I used BorederLineSMOTE over the traditional SMOTE due to its focused sampling on borderline instances, leading to improved generalization, reduced overfitting, and better classification performance in imbalanced datasets.

- Model Selection:
  1. From Literature it is proven that tress-based model will give good results for unbalanced datasets. Thus, I have employed the use of ensemble model(Random forests) which is known for robustness, scalability, and resistance to overfitting, **Overall Rf gave the best F1 score of 0.80**
  2. Additionally, I used voting classifier with soft vote to give the best output amongst three different models using grid search
  3. Cat boost and XG boost proven to have better prediction of true Positives which is our actual desires outcome . However, this is at the cost of more False Positives .

USC Viterbi School of Engineering

SUMMARY RESULTS

- From these results I infer that RF has the highest F1 score.
- However, based on the object if we are concerned about predicting the positive outcomes then we have to find the model with best recall.
- In this case LR has the best recall and the cost of high low precision.
- ADA boost also has good recall.

```
Model: Logistic Regression
F1-score weighted by class: 0.7059876351546149
Confusion Matrix:
[[419 211]
 [ 42  72]]
Model: Random Forest
F1-score weighted by class: 0.7982665414583385
Confusion Matrix:
[[562  68]
 [ 79  35]]
Model: Decision Tree
F1-score weighted by class: 0.7178091397849462
Confusion Matrix:
[[486 144]
 [ 84  30]]
Model: Naive Bayes
F1-score weighted by class: 0.7518787728226376
Confusion Matrix:
[[504 126]
 [ 73  41]]
Model: K-Nearest Neighbors
F1-score weighted by class: 0.7496078214759169
Confusion Matrix:
[[480 150]
 [ 58  56]]
Model: AdaBoost
F1-score weighted by class: 0.7561573670444638
Confusion Matrix:
[[492 138]
 [ 62  52]]
```

SUMMARY DISCUSSION

- Further scope of improvement
  1. Further down we can run shapely analysis to find out the feature importance of each variable used.
  2. We should definitely employ using grid search for other models and try find the best hyperparameters that could increase the overall AUC.
  3. Moving forward, it very unlikely to achieve at high Area under precision recall curve. Understanding the trade-off between precision and recall is crucial, as it allows us to customize the outcome according to our specific requirements and constraints.