Business Objective

Purpose of the Analysis:

The primary objective of this exploratory data analysis (EDA) is to lay a strong analytical foundation for the development of a predictive model aimed at identifying individuals at high risk of developing coronary heart disease (CHD) within the next 10 years. This predictive endeavor is crucial for early intervention strategies and enhancing patient outcomes by targeting preventive measures to those most at risk.

Data Overview:

The dataset comprises medical and demographic information from an ongoing health study, including age, gender, smoking status, blood pressure, cholesterol levels, and more. This rich dataset allows for a nuanced analysis of factors contributing to CHD, providing a holistic view of the patient profiles.

Dataset summary

- The training dataset contains 18 potential variables (expect patientID) and 3816 observations
- The response variables is TenYearCHD
- Discrete variables contain:
    - male
    - education
    - currentSmoker
    - BPMeds
    - prevalentStroke
    - diabetes
    - prevalentHyp
    - TenYearCHD

- Continuous variables contain:
    - age
    - cigsPerDay
    - totChol
    - sysBP
    - diaBP
    - BMI
    - heartRate
    - glucose
    - a1c
    - income

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)

Data quality summary

```
df.isnull().sum()
```

```
patientID            0
male                 0
age                  0
education           93
currentSmoker        0
cigsPerDay        1975
BPMeds              45
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             47
sysBP                0
diaBP                0
BMI                 19
heartRate            1
glucose            361
TenYearCHD           0
a1c                361
income               0
dtype: int64
```

Missing Values:

- 7 variables have missing values

I will decide to impute glucose and BPMeds based on the later feature selection.

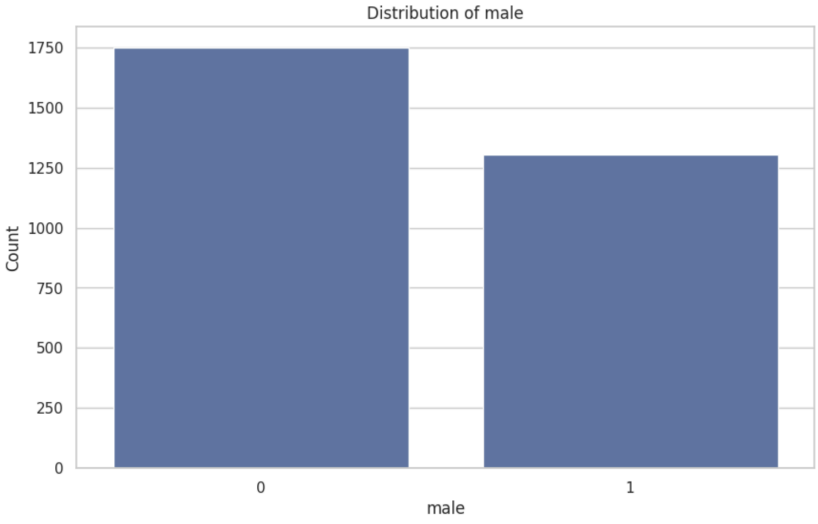
```
df.dtypes
```

```
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
```

Data type:

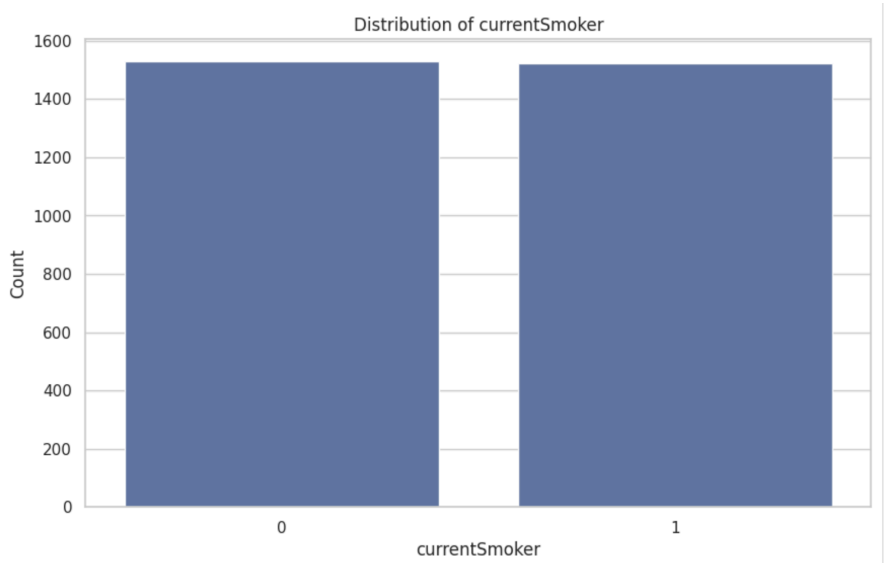
- All are numerical, but some are discrete, and some are continuous

Univariate Analysis

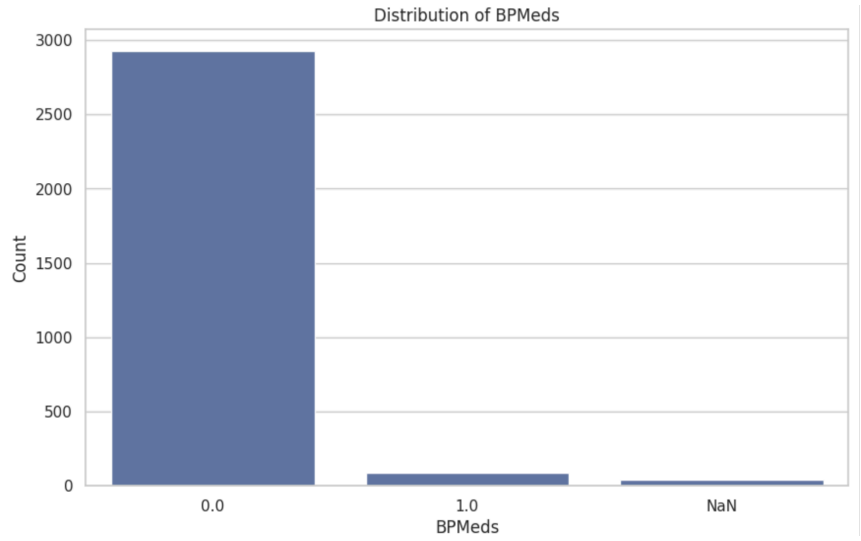


Moderately balanced

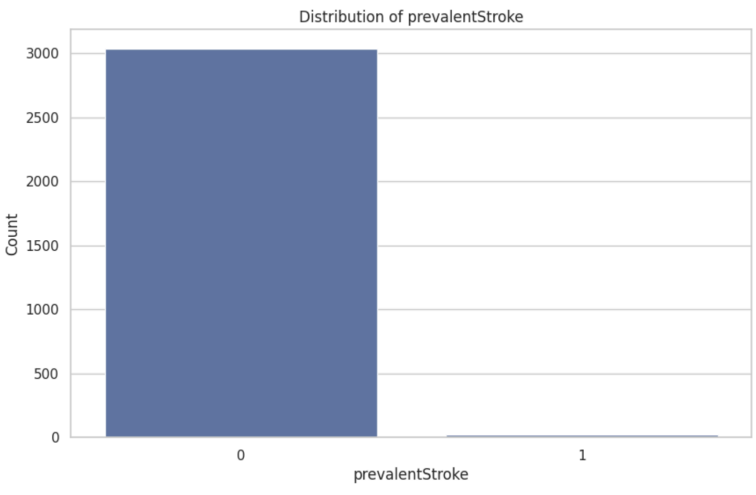
Balanced



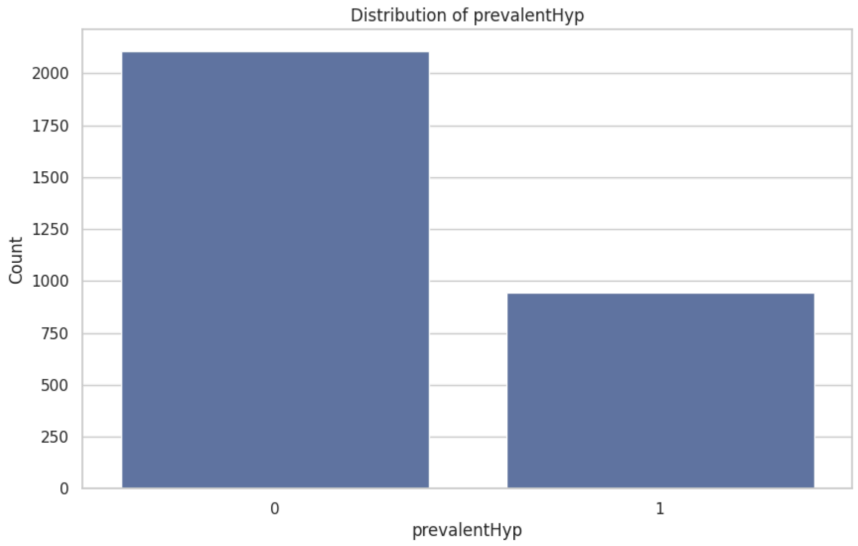
Very imbalanced



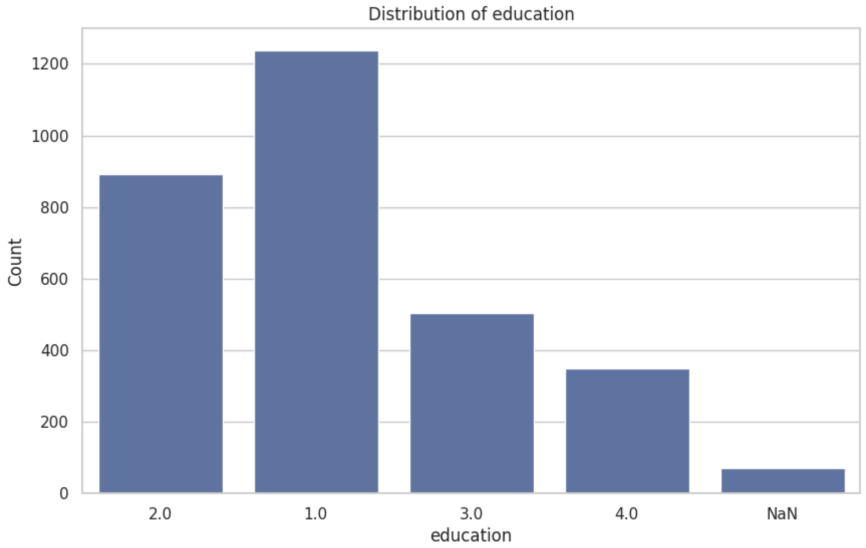
Very imbalanced



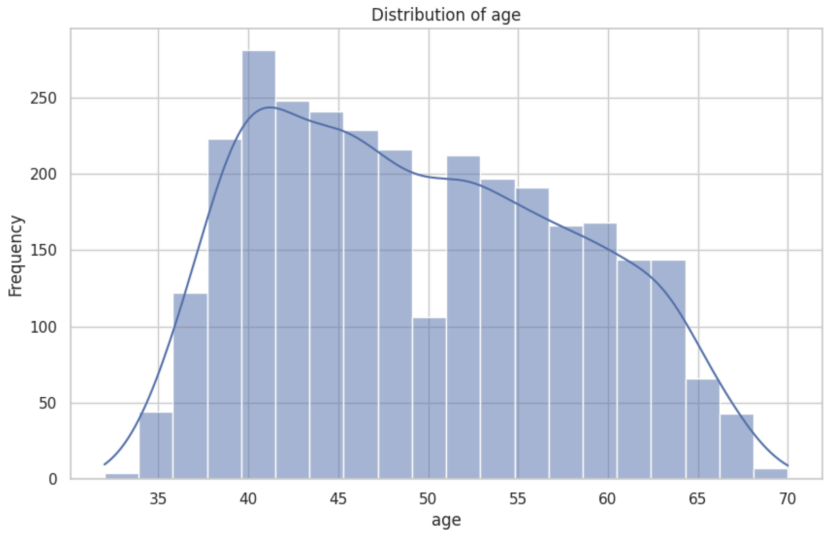
Imbalanced

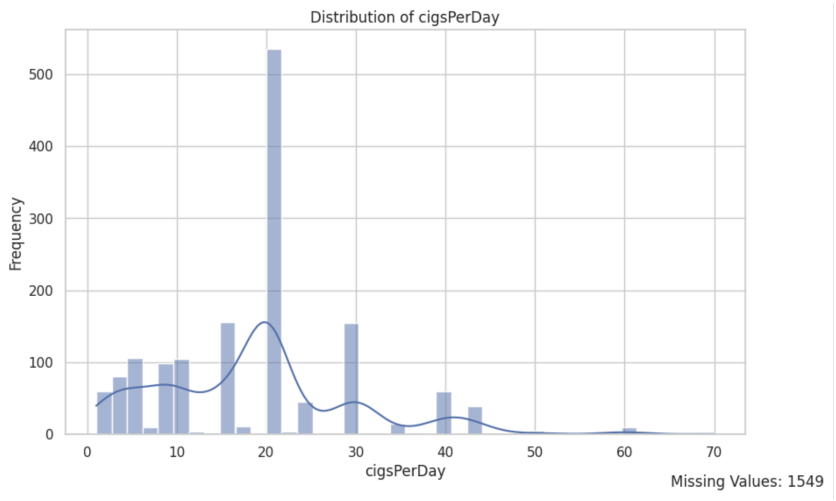


Imbalanced

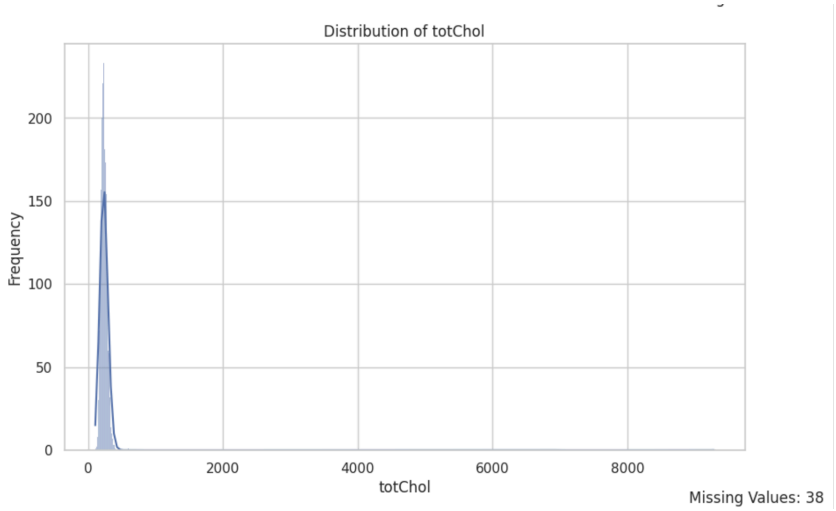


Moderately right-skewed

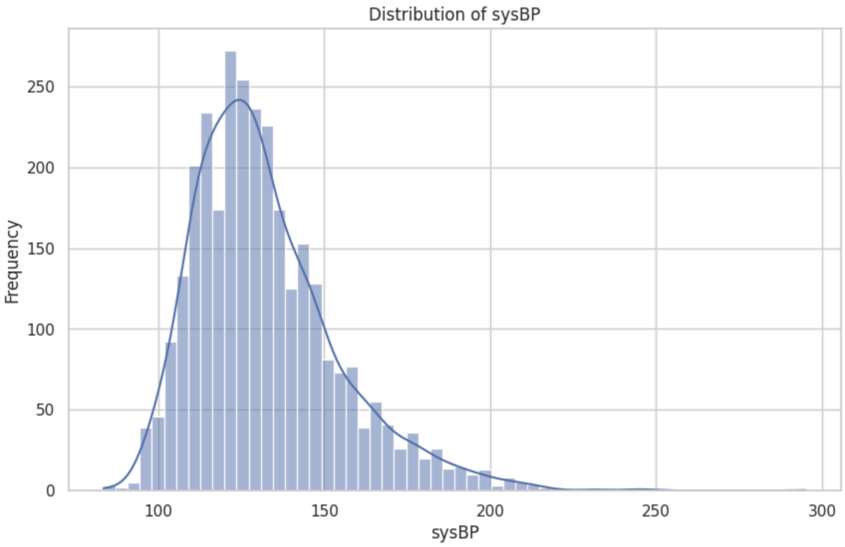




Right-skewed

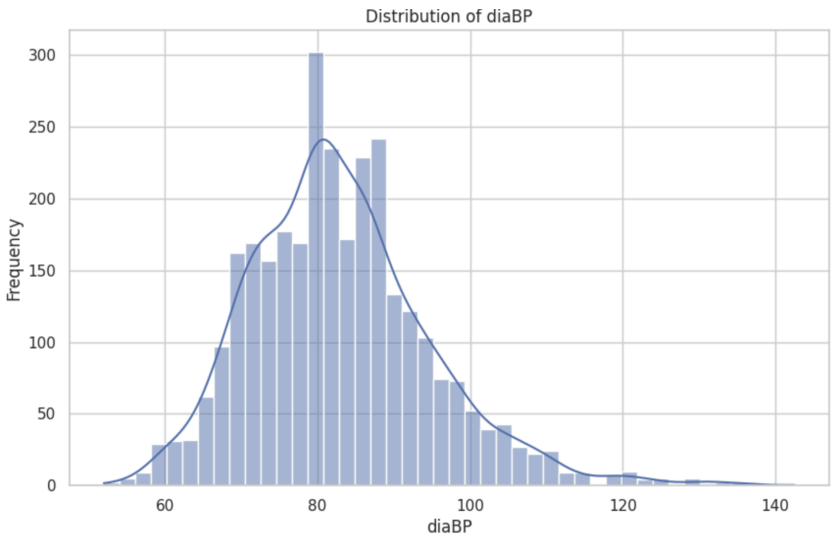


Intensively right-skewed

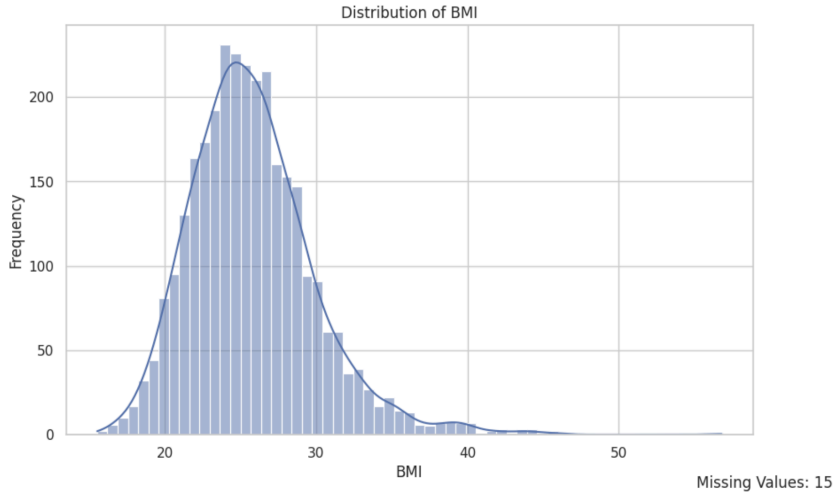


Moderately right-skewed

Slightly right-skewed

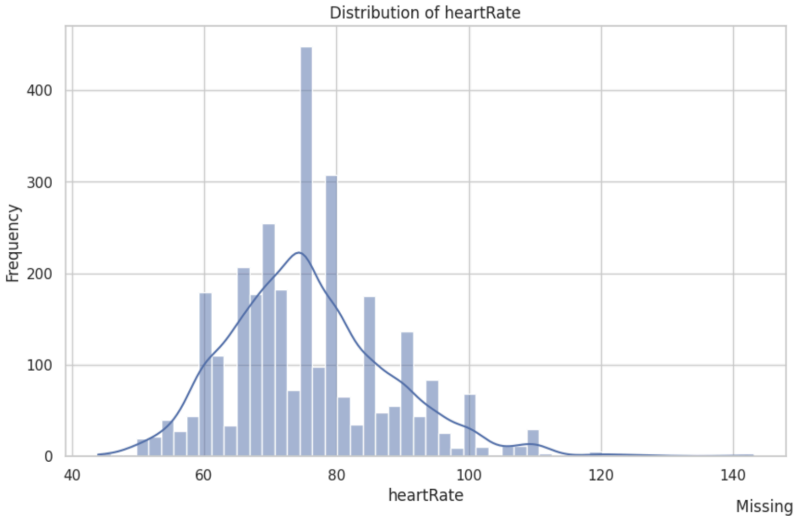


Right-skewed



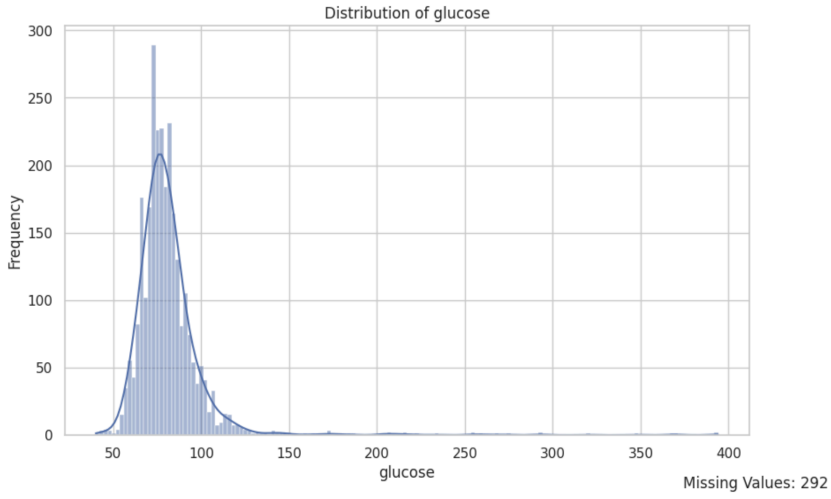
Missing Values: 15

Right-skewed

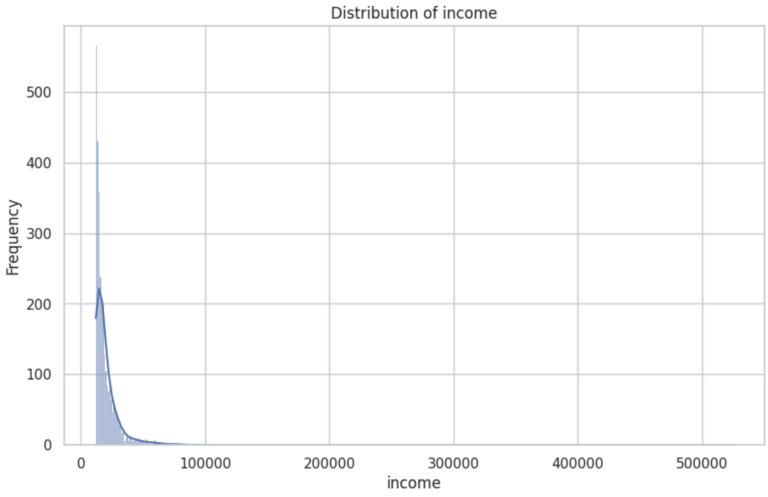


Missing Values: 1

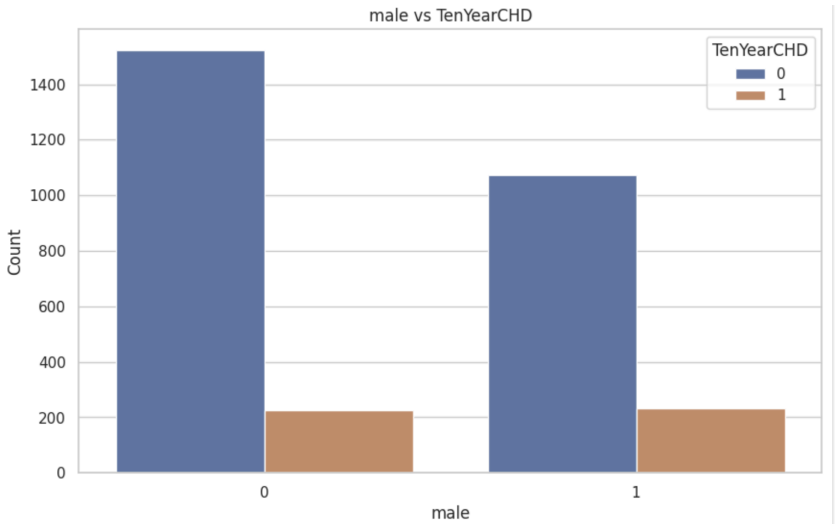
Intensively right-skewed



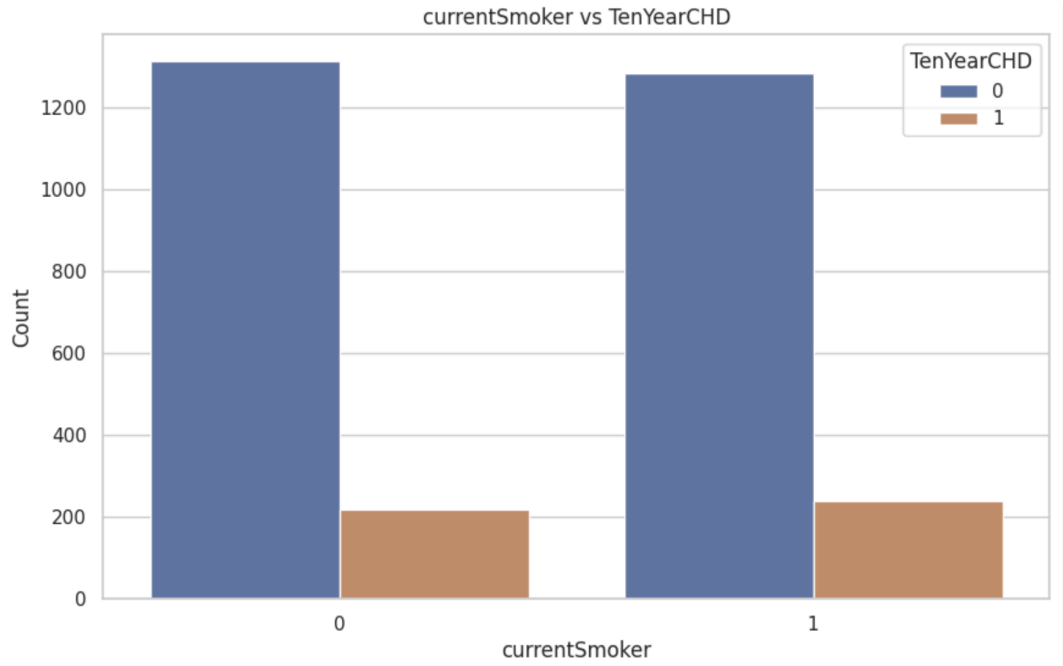
Intensively right-skewed



Bivariate Analysis to Response Varibale

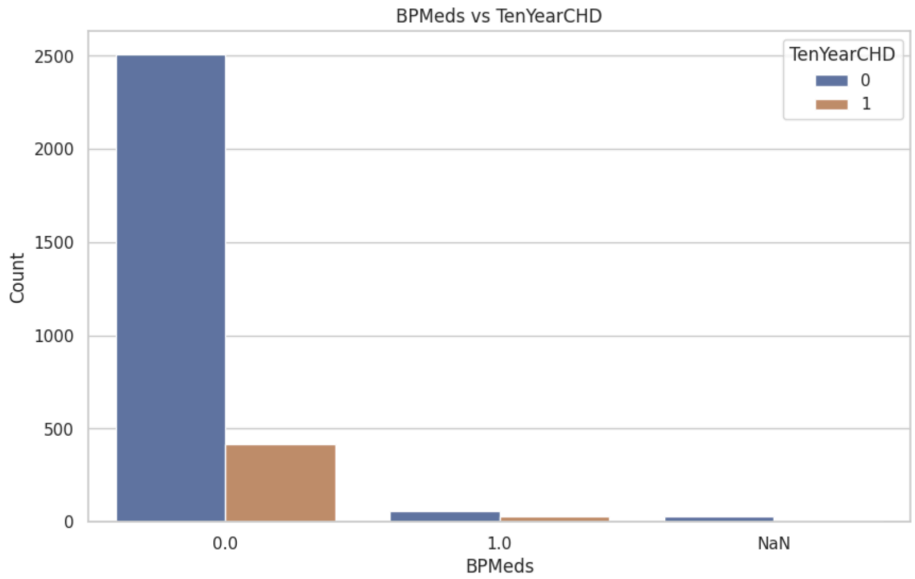


The majority of non-males did not develop CHD (TenYearCHD = 0), represented by the large blue bar.
A smaller number of non-males did develop CHD, represented by the smaller brown bar.

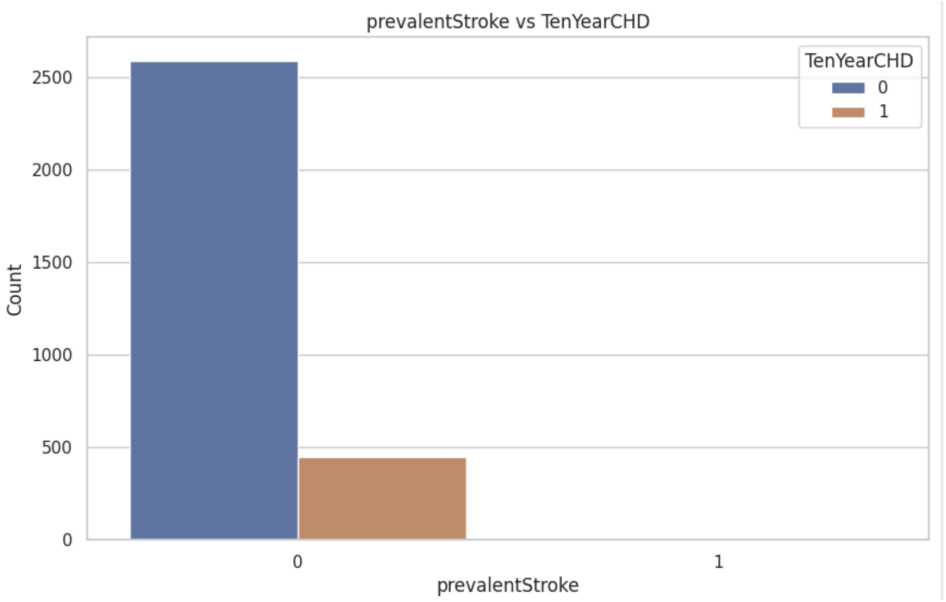


The majority of non-smokers did not develop CHD (TenYearCHD = 0), represented by the large blue bar.
A smaller number of non-smokers did develop CHD, represented by the smaller brown bar.
Among current smokers, the count of those who did not develop CHD is substantial but smaller compared to non-smokers, represented by the blue bar.

The number of current smokers who developed CHD is represented by a brown bar, indicating a similar proportion of disease occurrence among smokers compared to non-smokers.

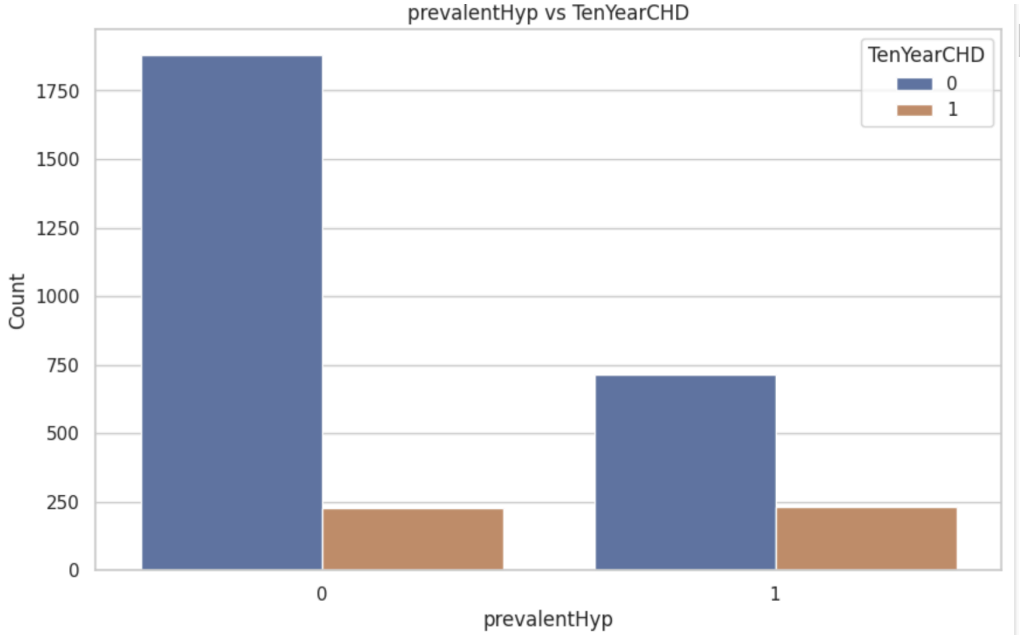


A large number of individuals not on blood pressure medications did not develop CHD (TenYearCHD = 0), represented by the large blue bar.
Among those not on blood pressure medications, a smaller number did develop CHD, shown by the brown bar.
For individuals on blood pressure medications, the number who did not develop CHD is significantly lower but still larger than those who did, as shown by the respective blue and brown bars.

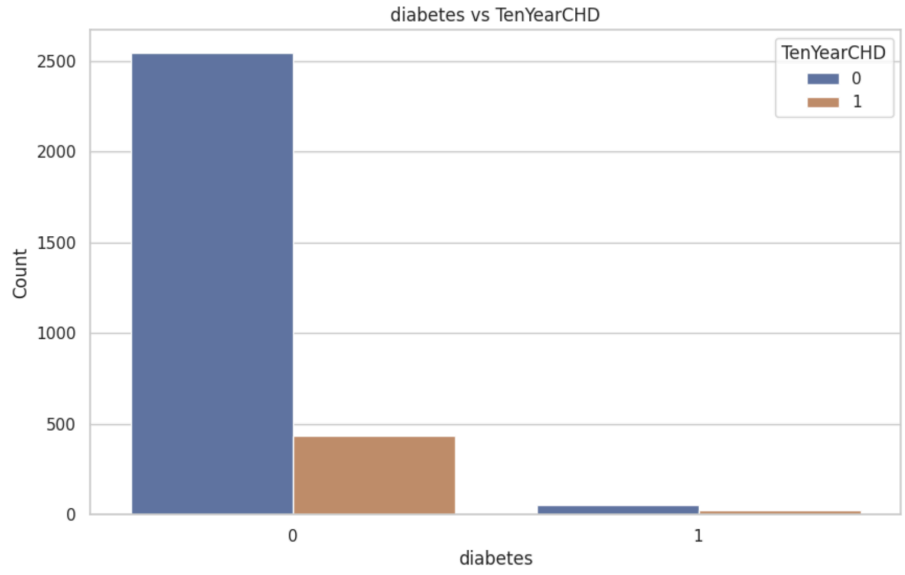


A large number of individuals without a history of stroke did not develop CHD (TenYearCHD = 0), shown by the large blue bar.

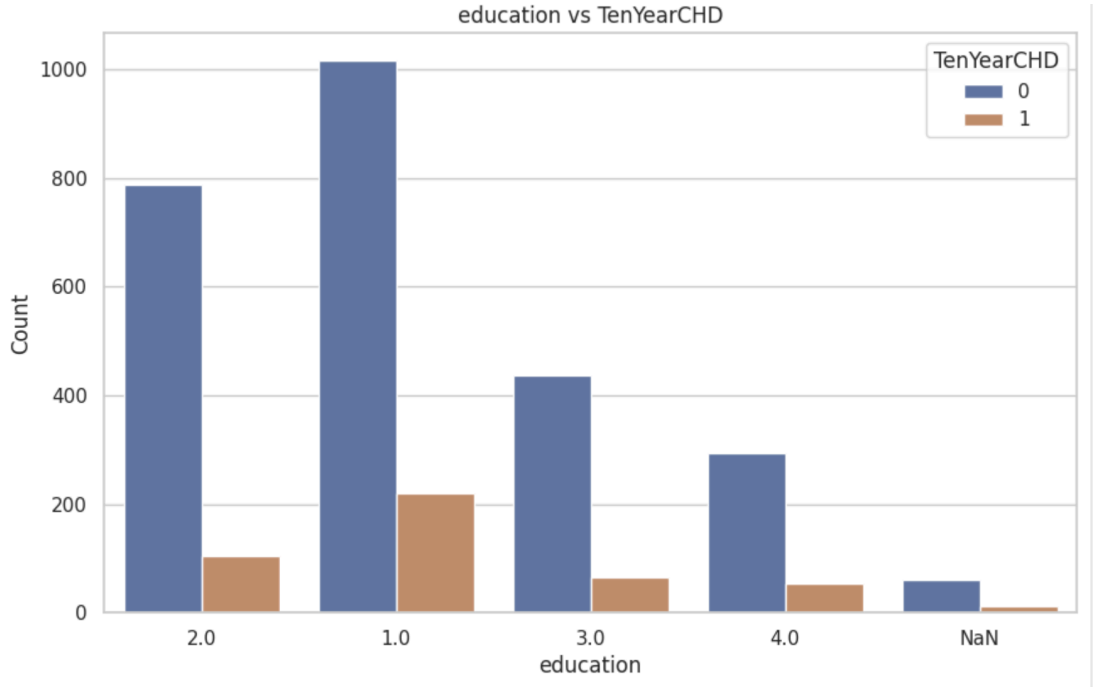
Among those without a history of stroke, a smaller number did develop CHD, as shown by the brown bar.



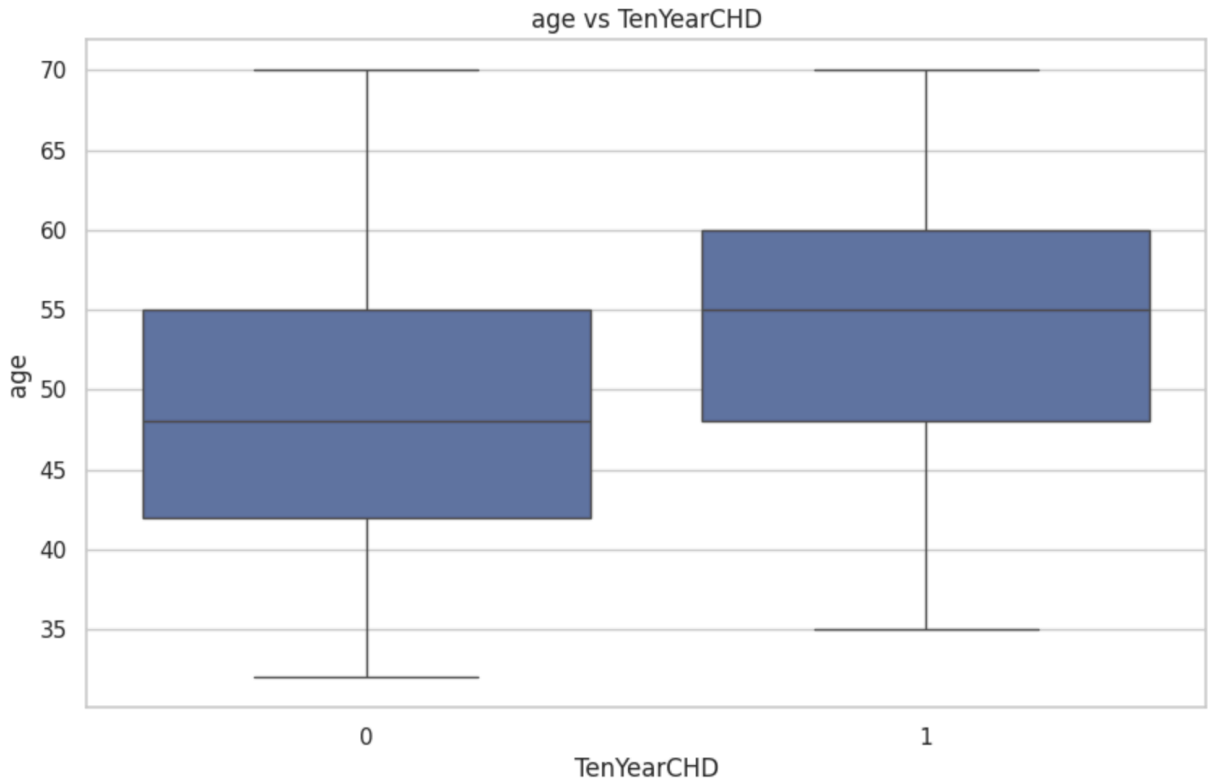
A large number of individuals without hypertension did not develop CHD (TenYearCHD = 0), shown by the large blue bar.
Among those without hypertension, a smaller number did develop CHD, shown by the smaller brown bar.
For individuals with hypertension, there is a significant count of those who did not develop CHD, though the number is less than those without hypertension, as indicated by the blue bar.
The number of individuals with hypertension who developed CHD is also considerable, shown by the brown bar, suggesting a higher incidence of CHD in this group compared to those without hypertension.



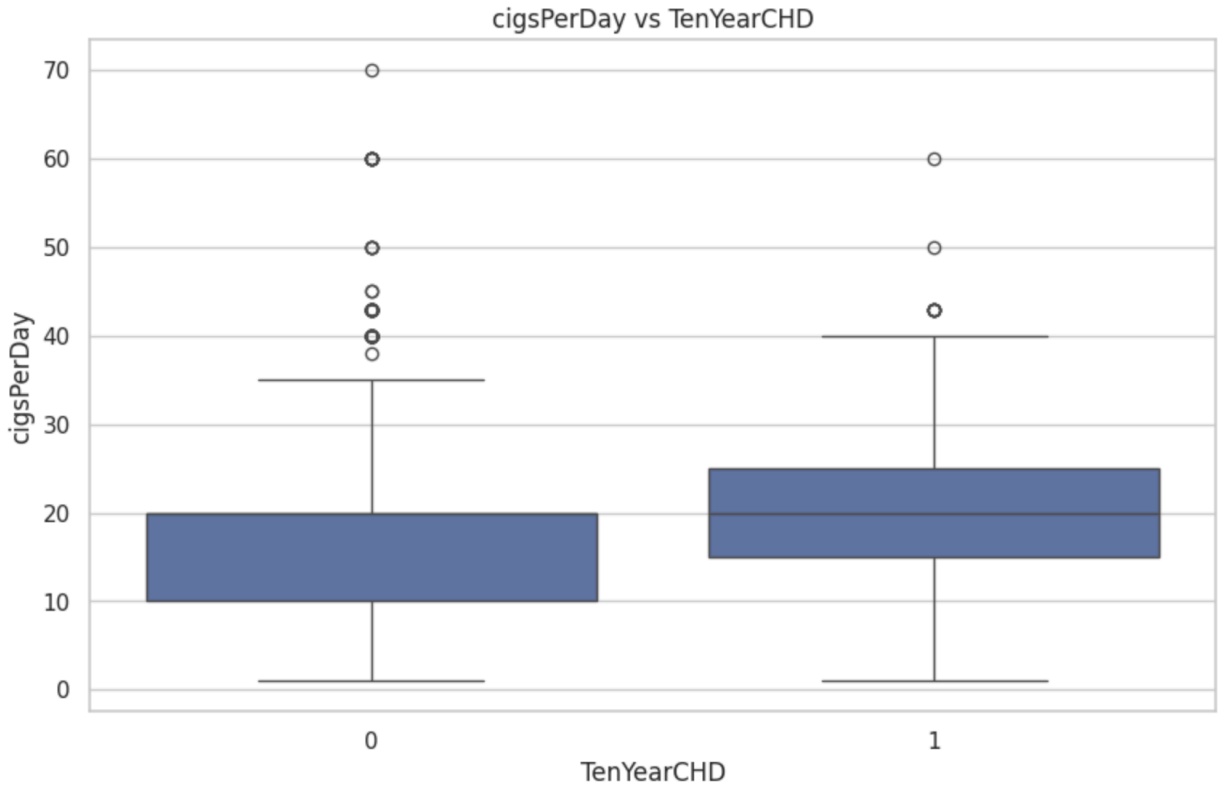
A large number of individuals without diabetes did not develop CHD (TenYearCHD = 0), shown by the large blue bar.
Among those without diabetes, a smaller number did develop CHD, as shown by the brown bar.
The number of individuals with diabetes who did not develop CHD is very low compared to those without diabetes, as indicated by the blue bar.



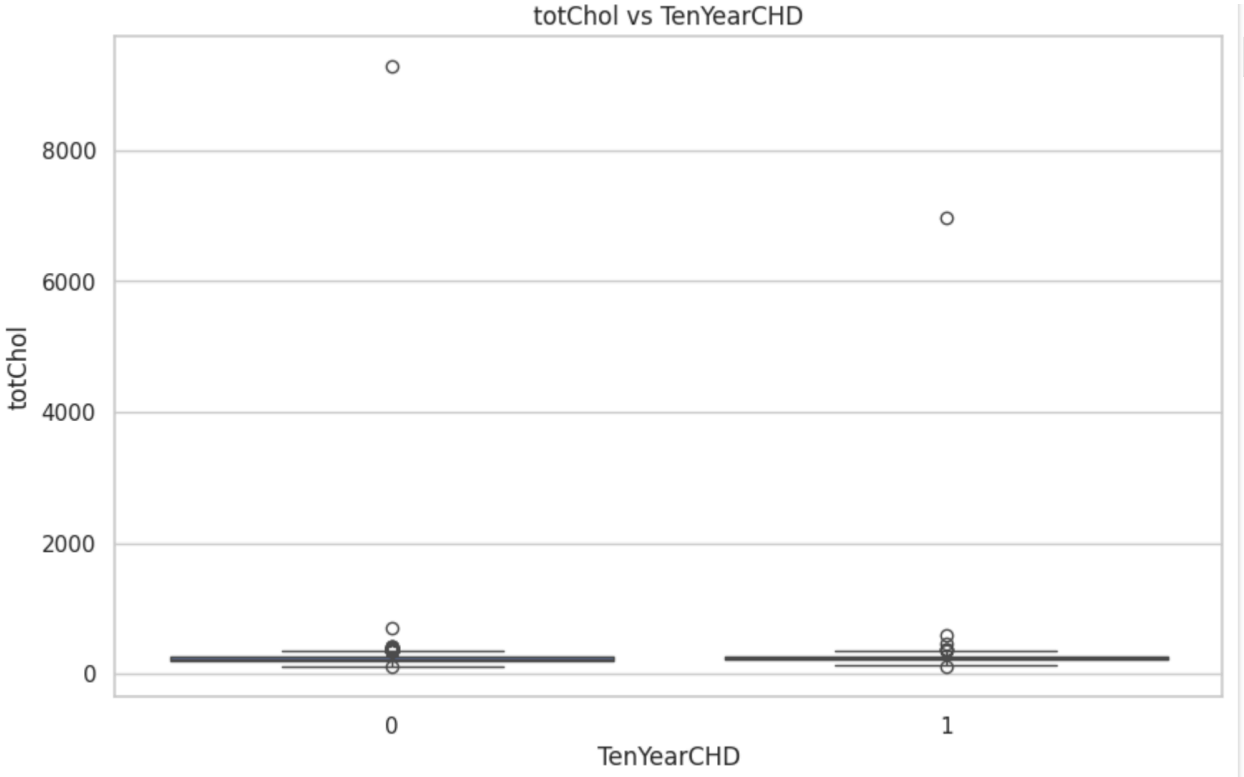
Education Level 1 (Less than high school): A notable number of individuals did not develop CHD, with a relatively higher proportion developing CHD compared to other education levels.
Education Level 2 (Completed high school or equivalent): The largest number of individuals falls into this category, with most not developing CHD and a smaller proportion developing it.
Education Level 3 (Some college): Fewer individuals are represented, with most not developing CHD and an even smaller proportion developing it compared to the first two levels.
Education Level 4 (Completed college or higher): The smallest group, with very few developing CHD.

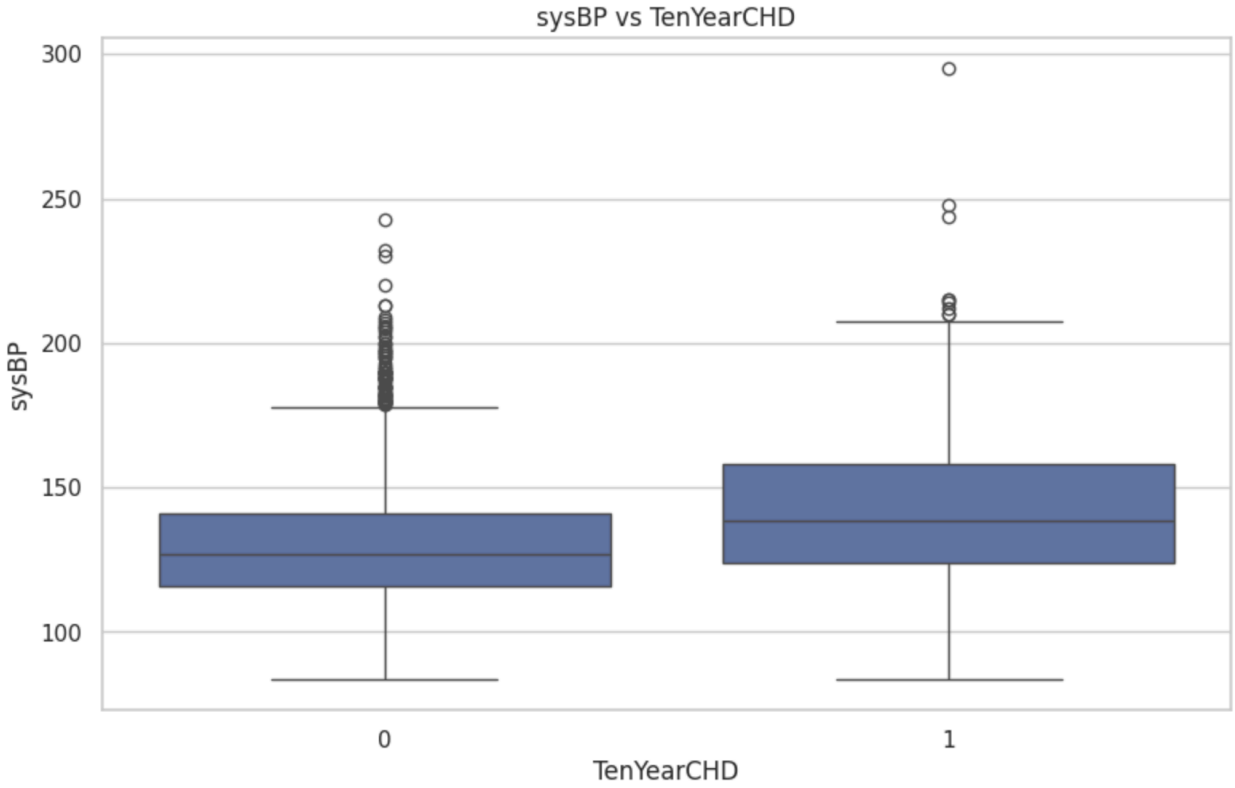


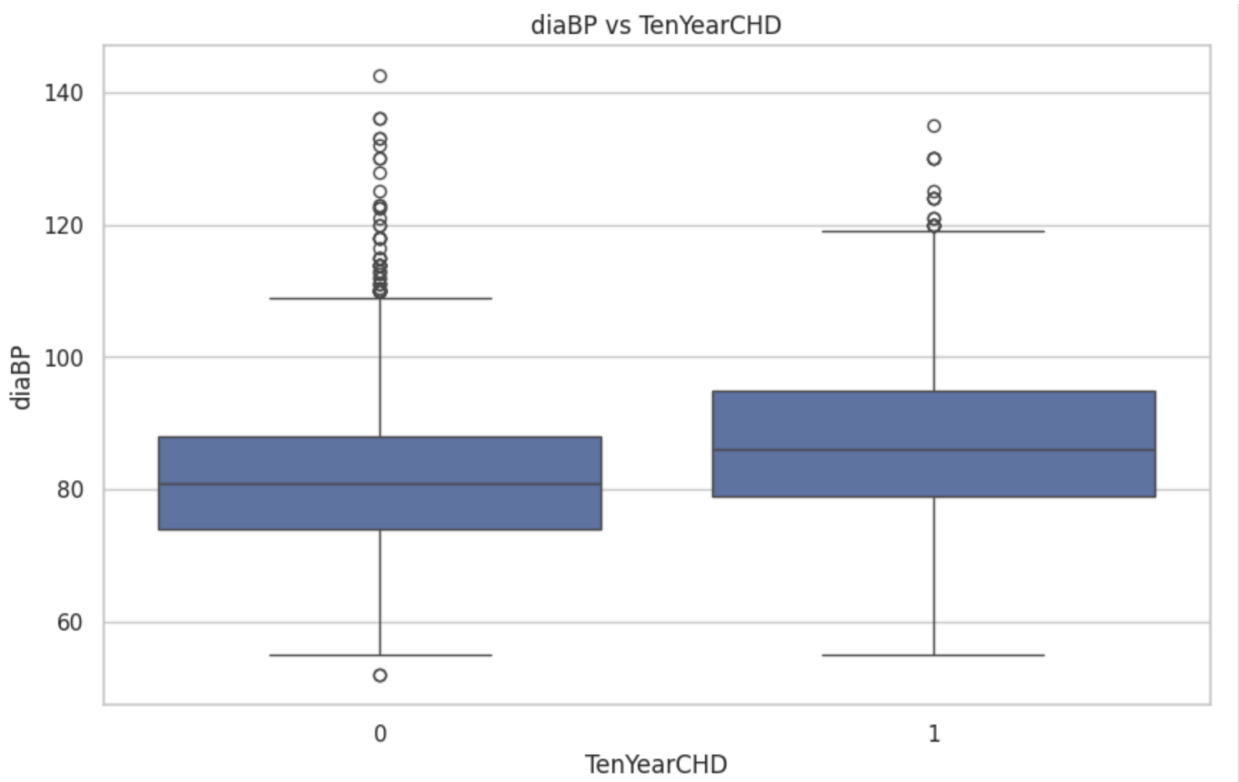
The median age of individuals who did not develop CHD (TenYearCHD = 0) is around 50 years, as indicated by the line within the blue box.
The age range (interquartile range, IQR) for those who did not develop CHD spans from approximately 45 to 55 years, with the box representing the middle 50% of the data.
The median age of individuals who developed CHD (TenYearCHD = 1) is higher, around 55 years.
The age range for those who developed CHD spans from about 50 to 60 years, showing a higher age distribution compared to those who did not develop CHD.

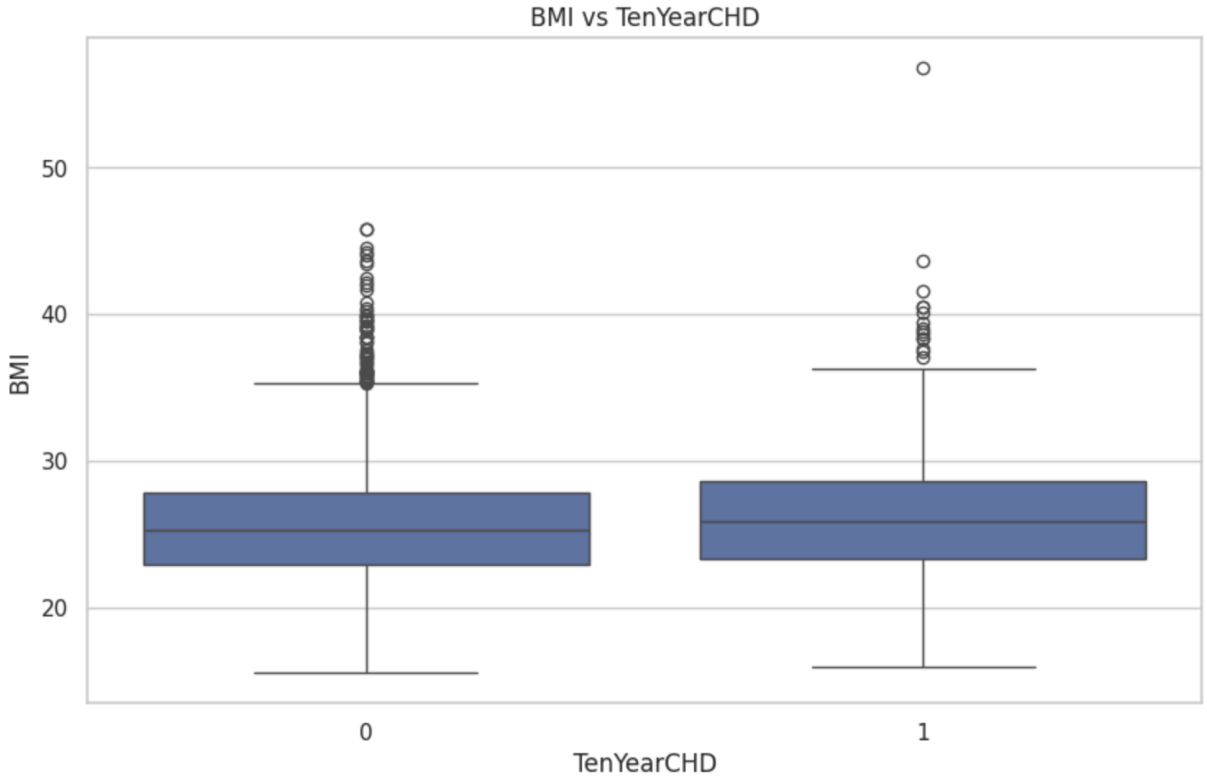


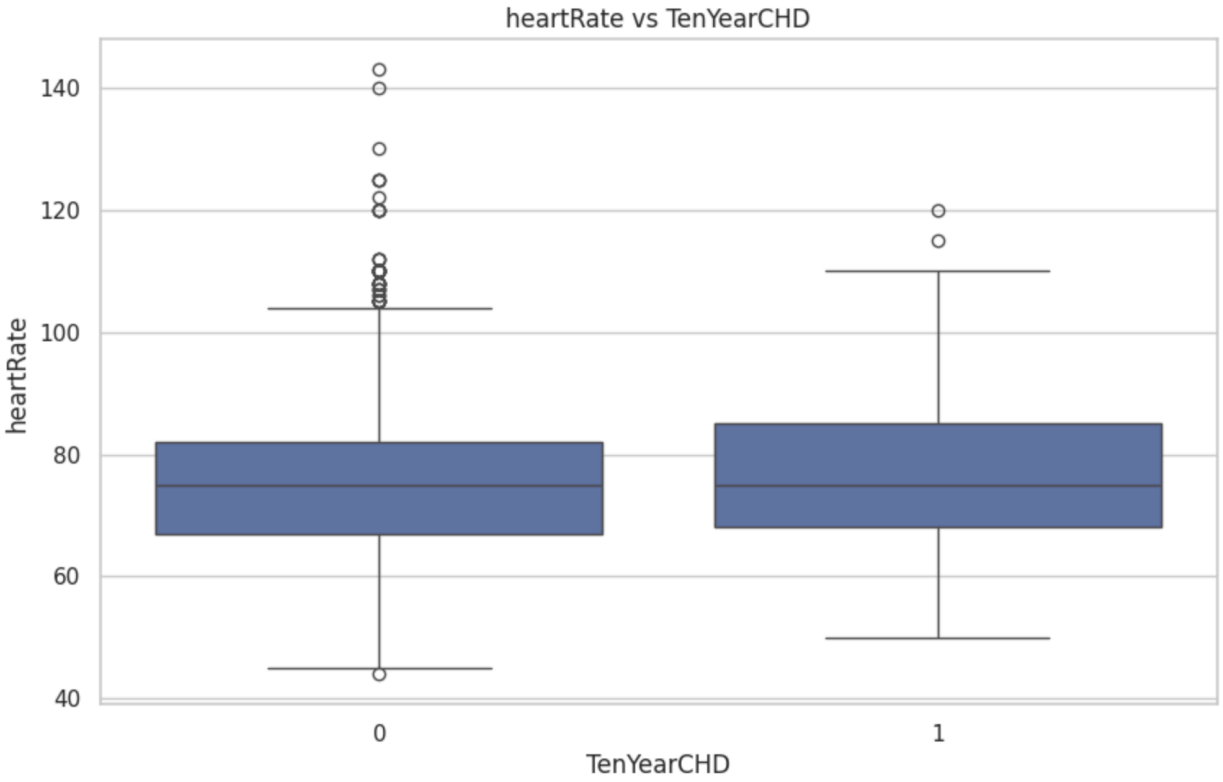
For individuals who did not develop CHD (TenYearCHD = 0):
The median number of cigarettes smoked per day is lower, close to around 10.
The interquartile range (IQR), represented by the height of the box, shows a moderate spread of daily smoking amounts, spanning from near 0 to about 20 cigarettes per day.
There are several outliers who smoke significantly more, up to around 40 or more cigarettes per day.
For individuals who developed CHD (TenYearCHD = 1):
The median cigarettes per day is slightly higher, close to 15.
The IQR is broader, indicating more variability in the number of cigarettes smoked, ranging from near 0 to about 30 cigarettes per day.
There are also outliers in this group, with some smoking upwards of 30 cigarettes per day.

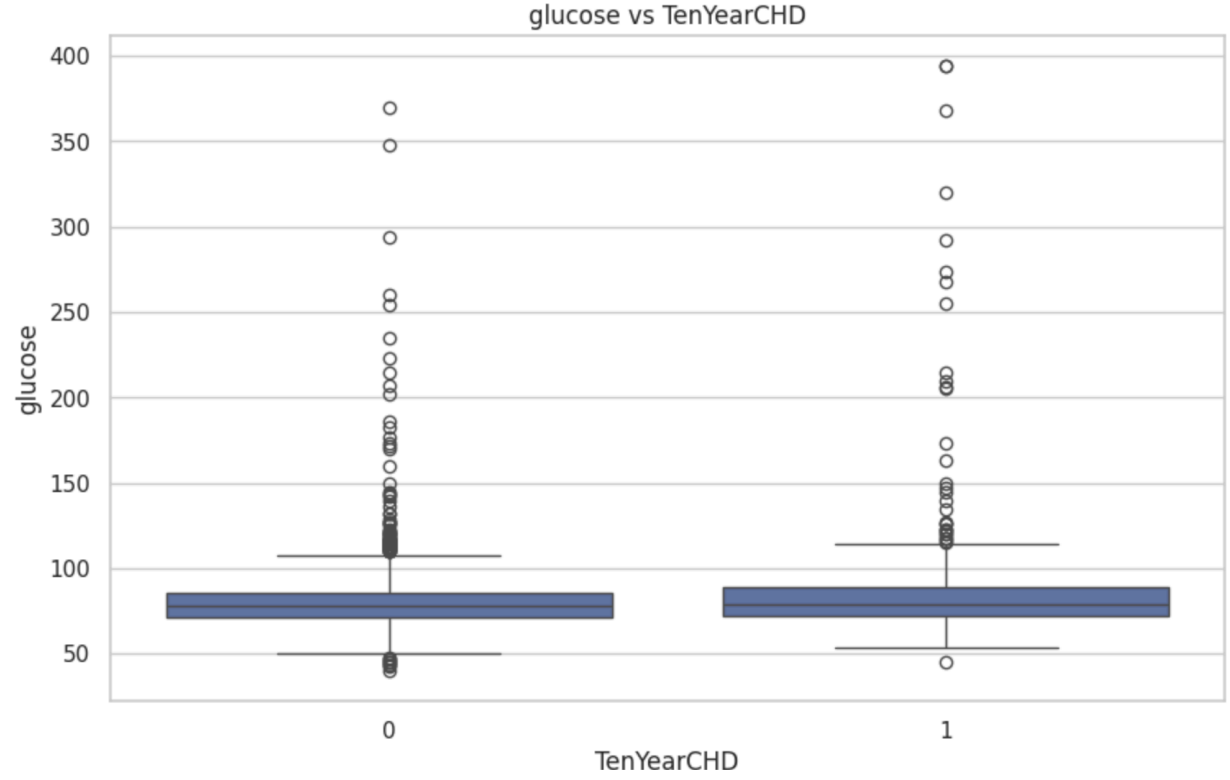

totChol vs TenYearCHD
totChol
8000
6000
4000
2000
0
0
1
TenYearCHD

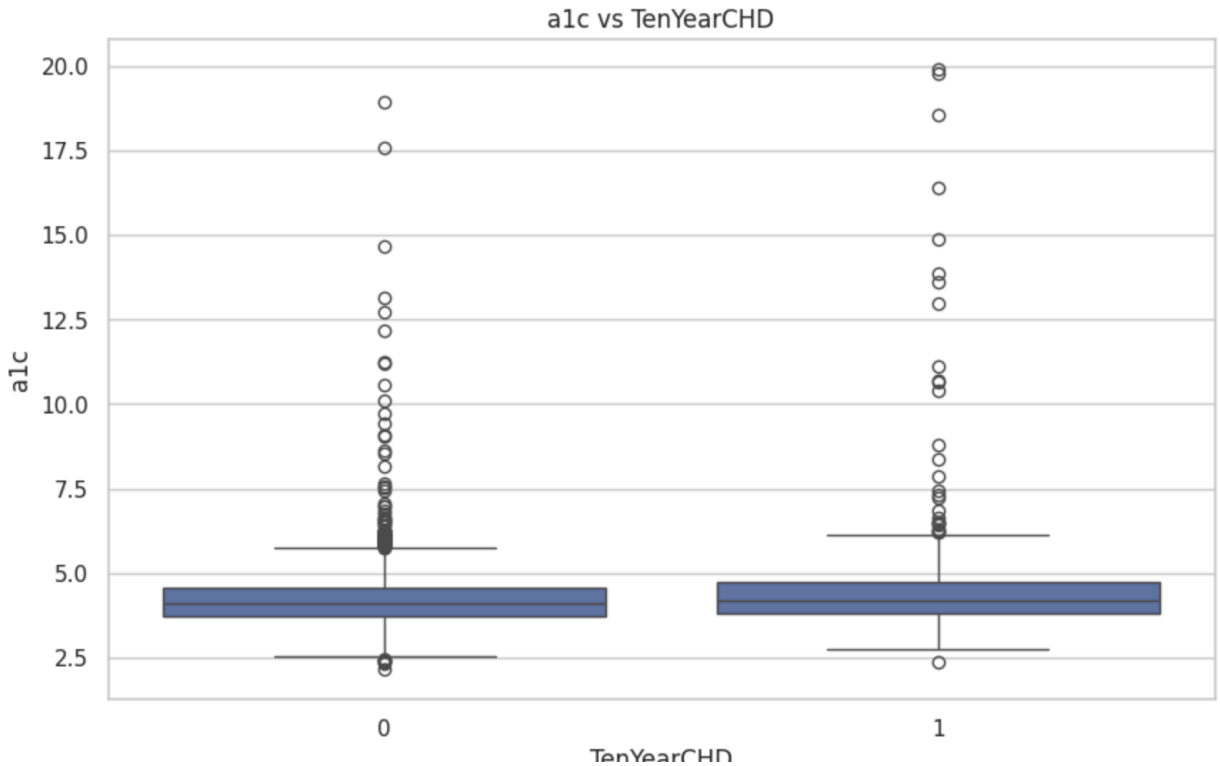

sysBP vs TenYearCHD
sysBP
300
250
200
150
100
0
1
TenYearCHD

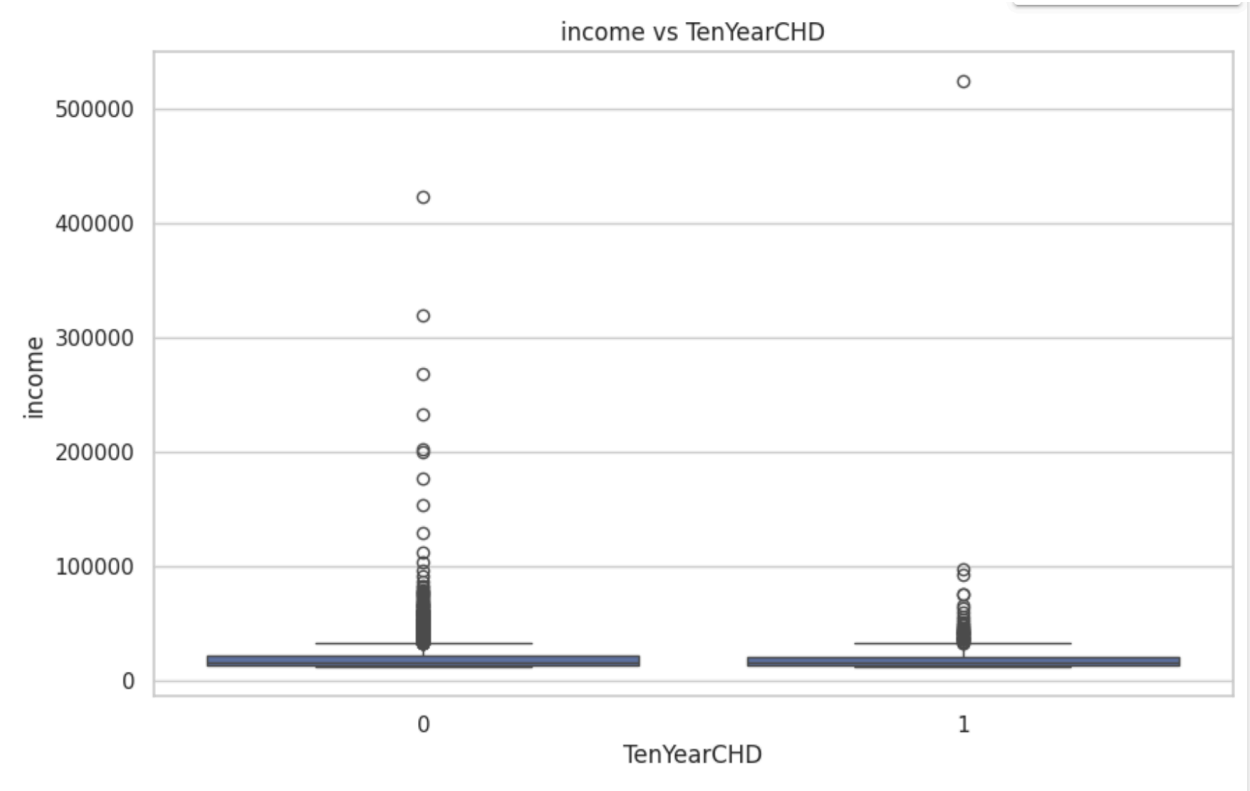

diaBP vs TenYearCHD
140
120
100
80
60
diaBP
0
1
TenYearCHD


BMI vs TenYearCHD
50
40
30
20
BMI
0
1
TenYearCHD

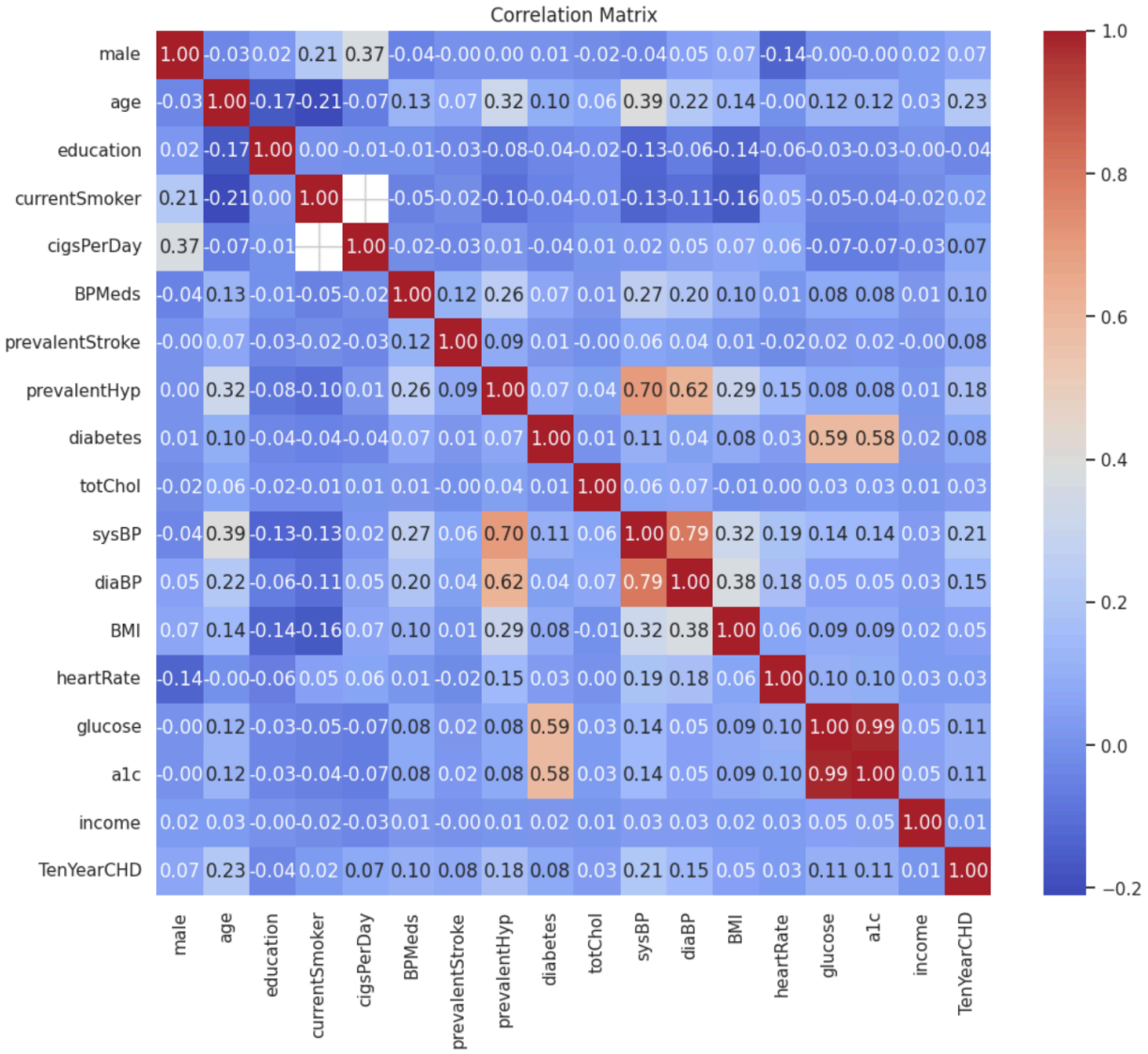

heartRate vs TenYearCHD
heartRate
140
120
100
80
60
40
0
1
TenYearCHD


glucose vs TenYearCHD
glucose
400
350
300
250
200
150
100
50
0
1
TenYearCHD


a1c vs TenYearCHD
a1c
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
1
TenYearCHD


income vs TenYearCHD
income
500000
400000
300000
200000
100000
0
0
1
TenYearCHD

Feature Selection



Features with high correlation to TenYearCHD (above 0.1): ['TenYearCHD', 'age', 'sysBP', 'prevalentHyp', 'diaBP', 'a1c', 'glucose', 'BPMeds']
Collinear feature pairs (above 0.7): {('diaBP', 'sysBP'), ('a1c', 'glucose')}

So, I decided to choose `'age', 'sysBP', 'prevalentHyp','glucose', 'BPMeds'` as my features for later analysis.

Dealing with Missing Values

```
X_train.isnull().sum()
```

```
age                0
sysBP              0
prevalentHyp       0
glucose          292
BPMeds            39
dtype: int64
```

Impute 'BPMeds', 'glucose' by using thier modes and medians in X_train

```
X_val.isnull().sum()
```

```
age              0
sysBP            0
prevalentHyp     0
glucose         69
BPMeds           6
dtype: int64
```

Data quality steps (dealing with outliers)

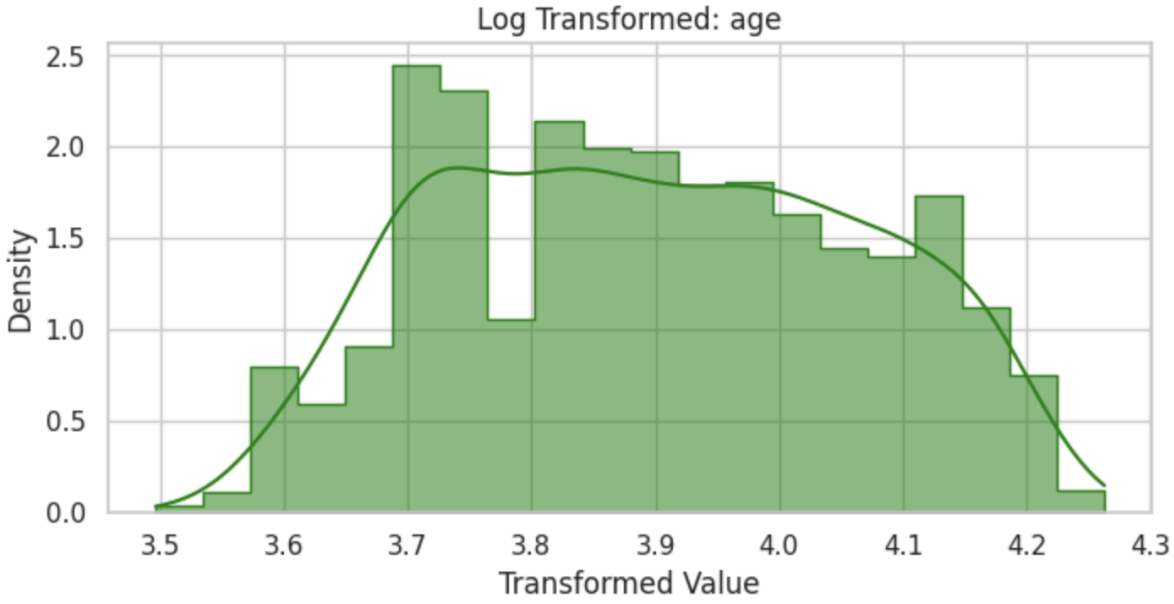
- Choose `continuous_features = ['age', 'sysBP','glucose']`
- Calculate IQR on the training set
- Apply to X_train and y_train, remove rows containing outliers
- Apply to X_val and y_val, delete rows containing outliers

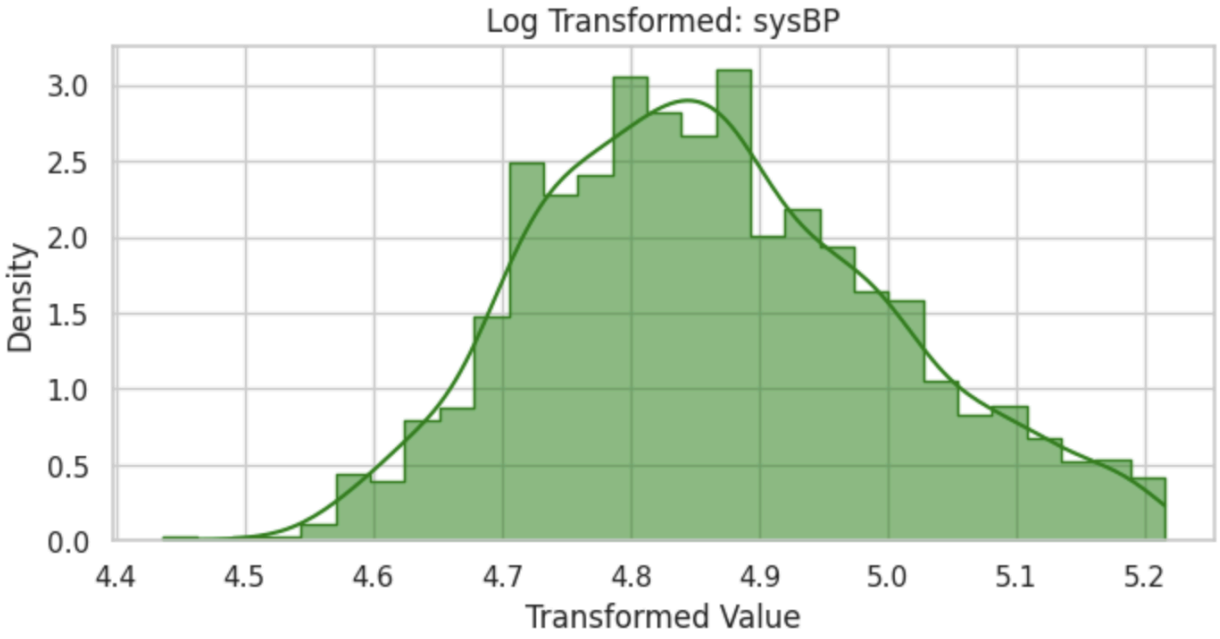
Feature Engineering Decision

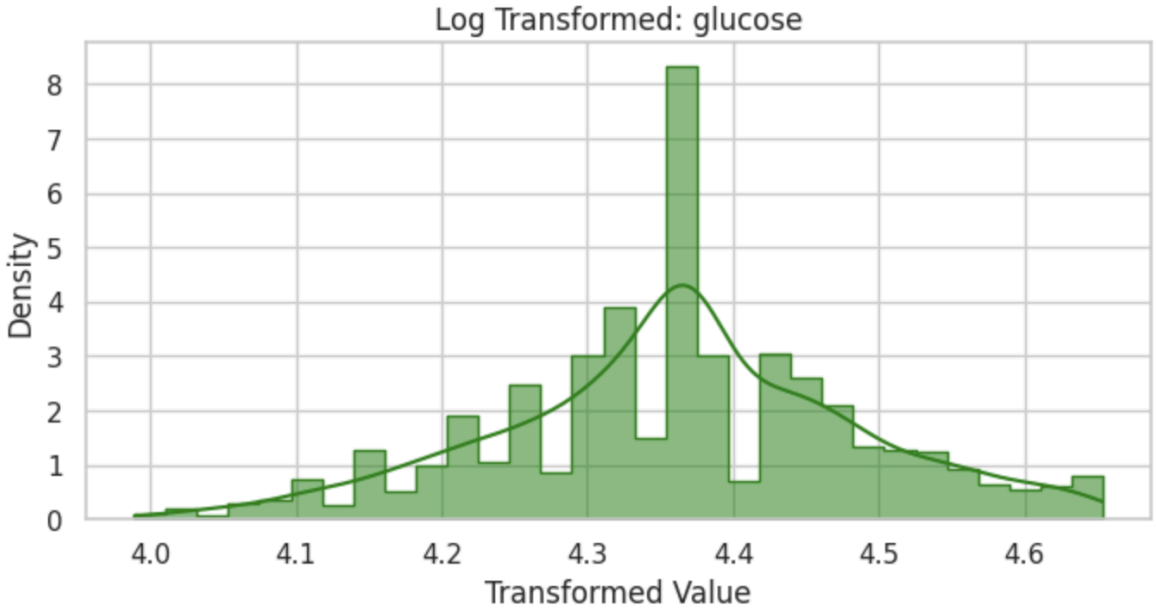
- Standardization
- Log transformation
- Oversampling

Log-transform

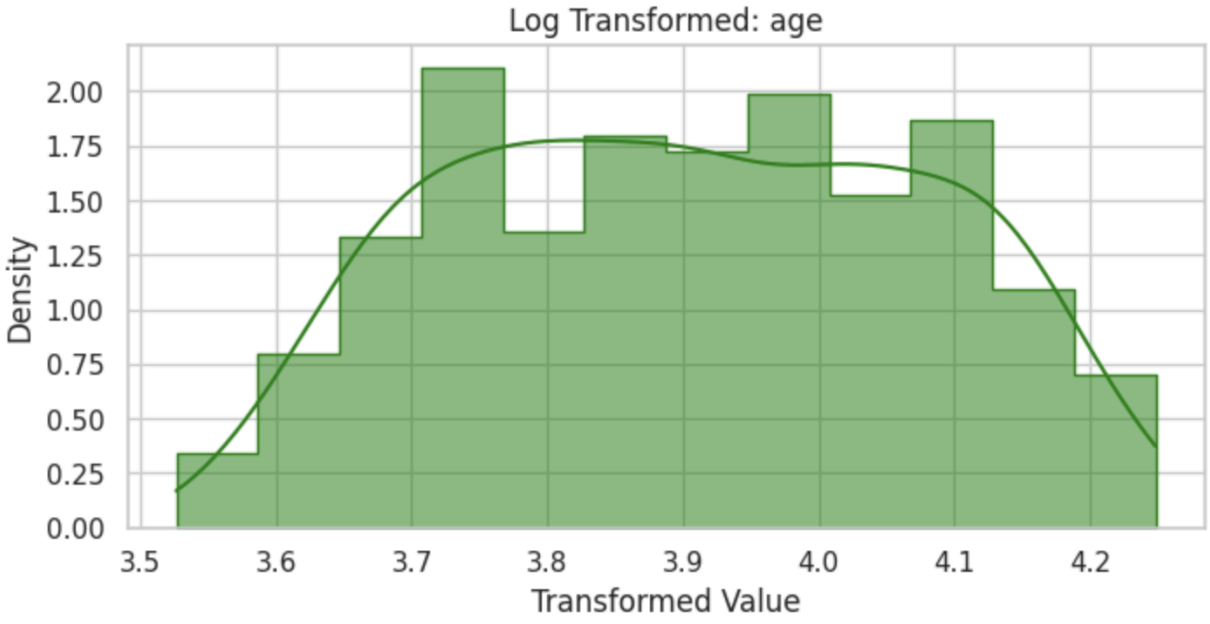
```
Training Data Histograms:
```

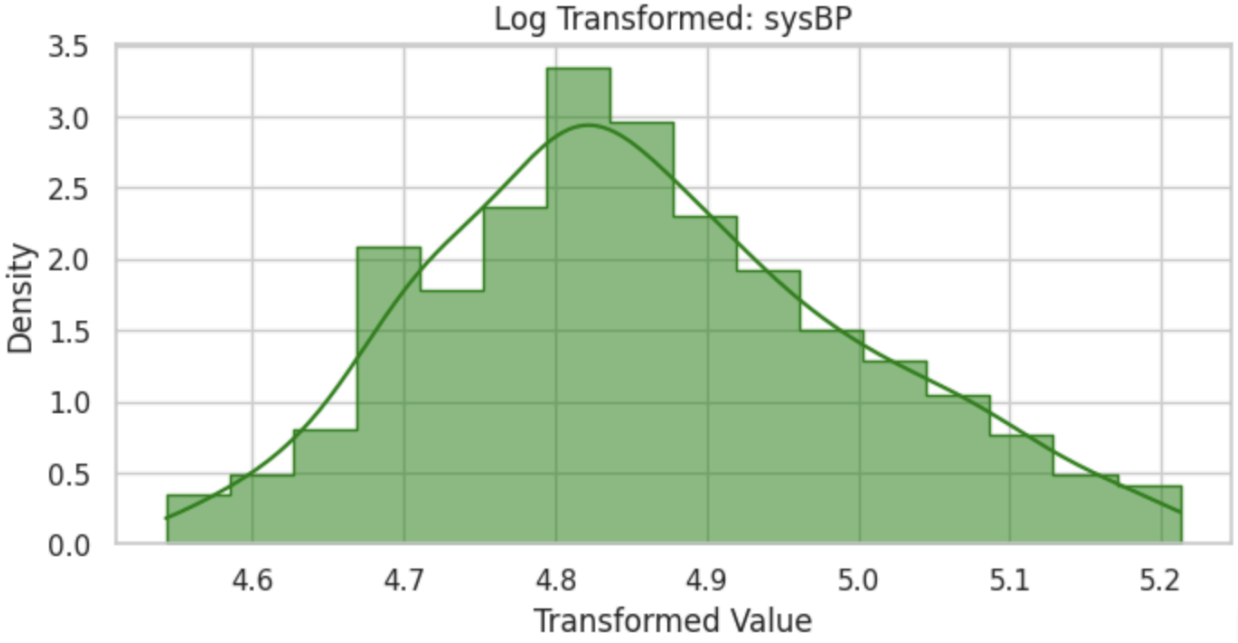


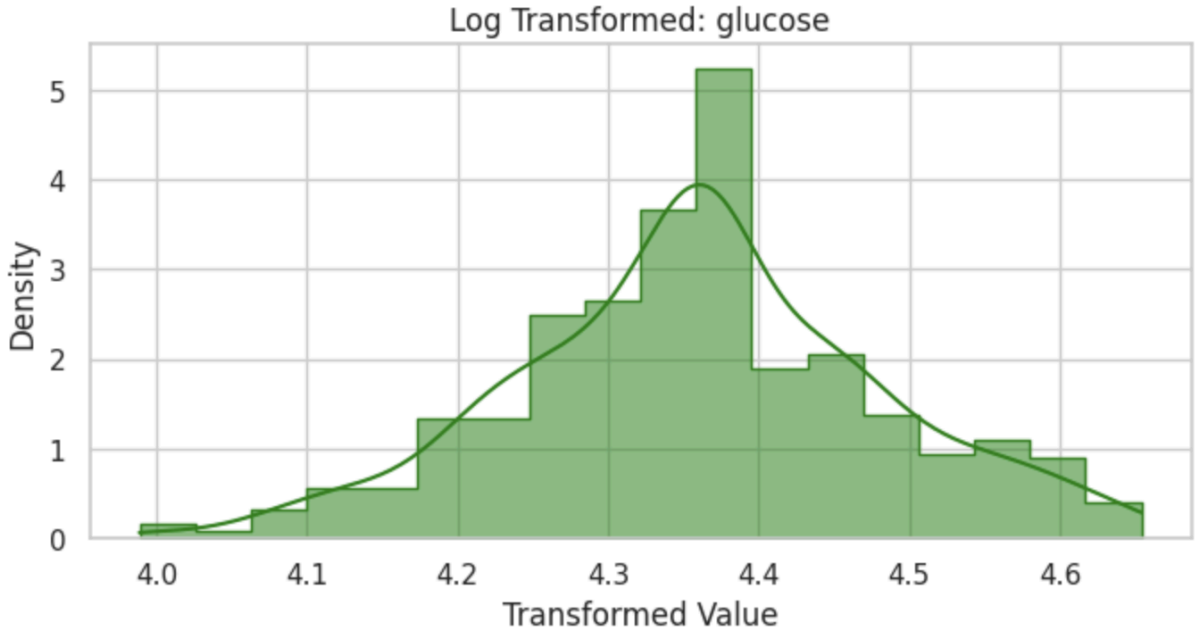




Validation Data Histograms:




Log Transformed: sysBP
Density
Transformed Value
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
4.6
4.7
4.8
4.9
5.0
5.1
5.2


Log Transformed: glucose
Density
Transformed Value
5
4
3
2
1
0
4.0
4.1
4.2
4.3
4.4
4.5
4.6

Standardization

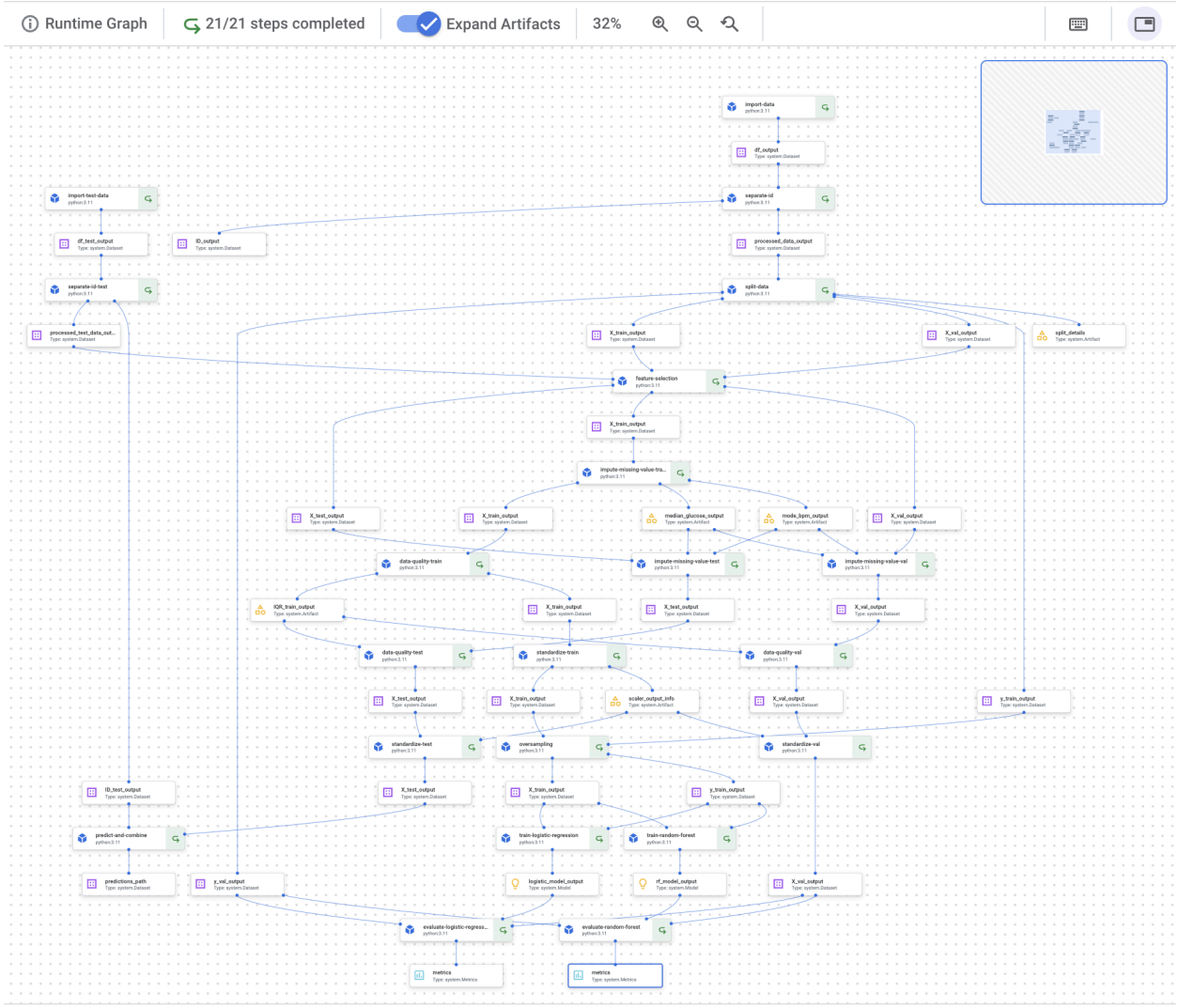
- Standardize `continuous_features = ['age', 'sysBP','glucose']`

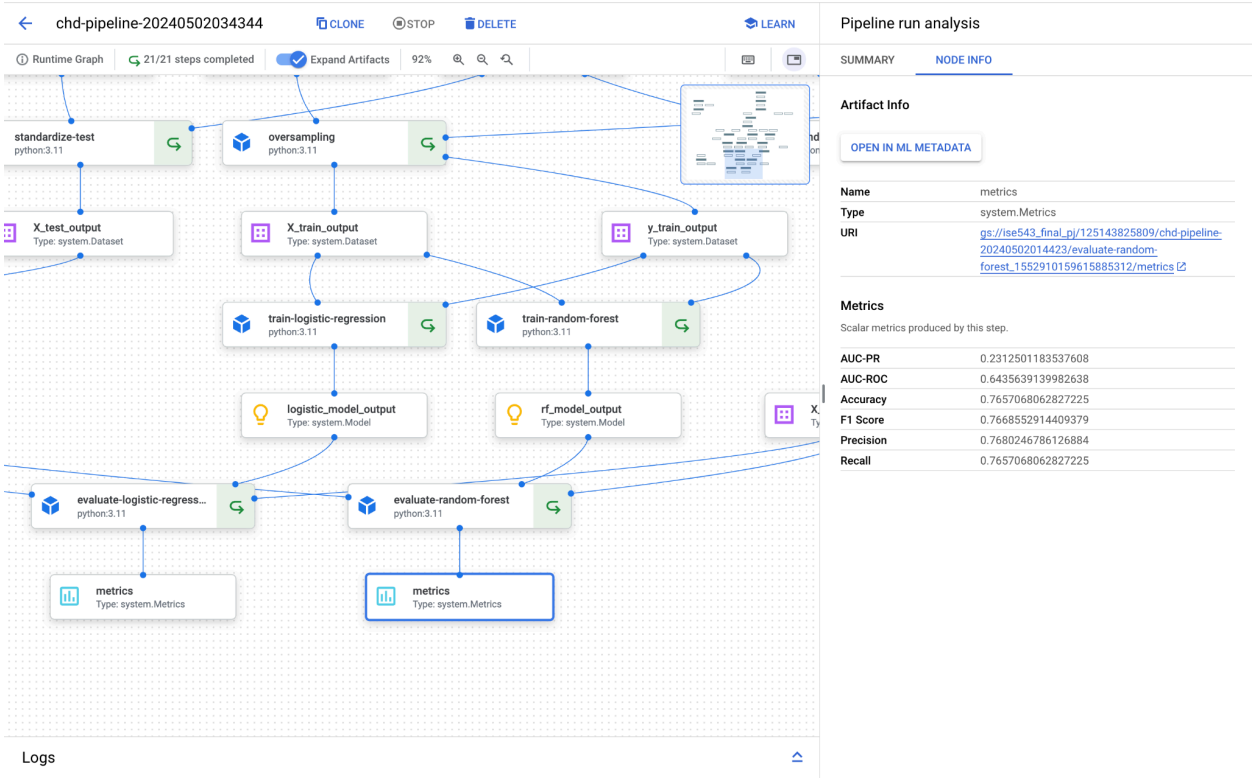
Oversampling

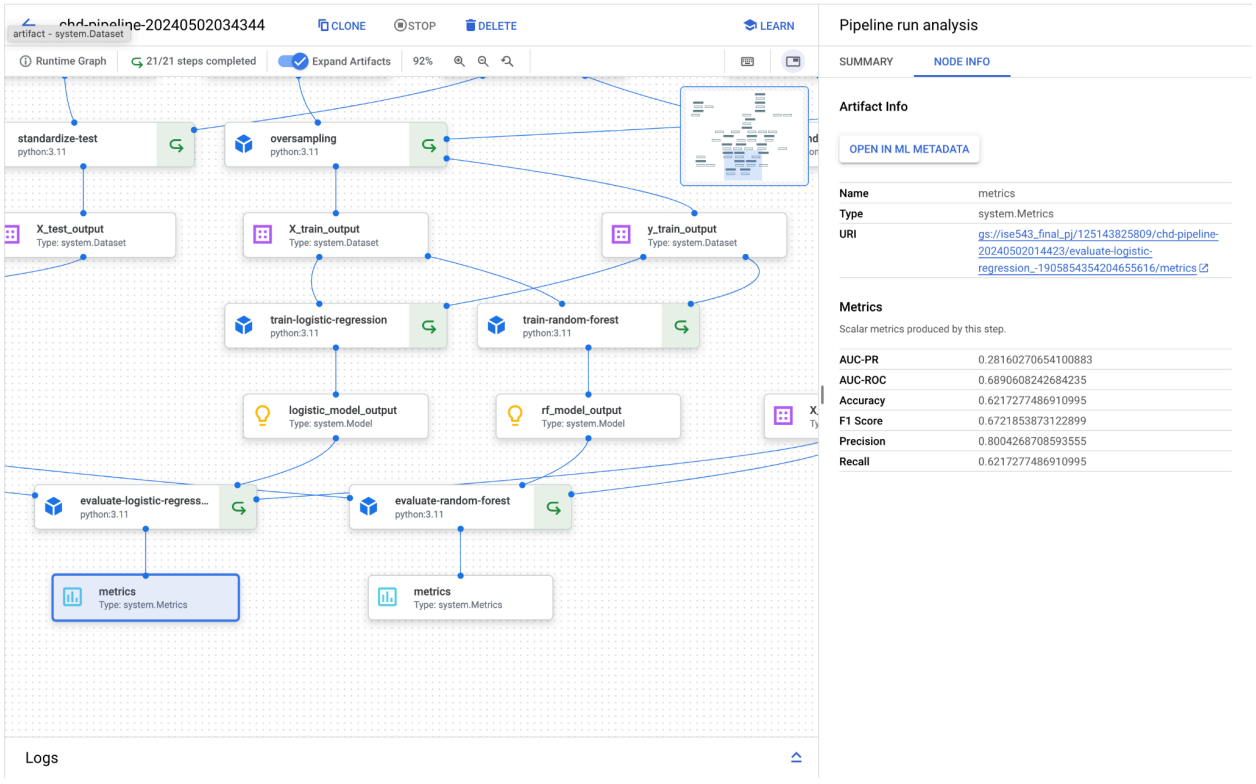
- Apply the oversampling method (SMOTE) only on the training set

```
Category distribution in the original training set:
 TenYearCHD
0    2416
1     382
Name: count, dtype: int64
Category distribution after applying SMOTE:
 TenYearCHD
0    2416
1    2416
Name: count, dtype: int64
```

Pipeline (I integrated inference pipeline and training pipeline)







So, I choose to use random forest model in my later prediction based on its higher F1-score.

Set parameters and initialize aiplatform client library

```
[ ] # Set parameters
    project_id = 'ise543-419203'
    location = 'us-central1'
```

```
[ ] from google.cloud import aiplatform
    aiplatform.init(project=project_id, location=location)
```

```
[ ] # !pip install kfp
```

```
[ ] # !pip install gcsfs
```

```
[ ] # !pip install fsspec
```

Define components

Import data

```
[ ] from kfp.dsl import pipeline, component
    from kfp.dsl import InputPath, OutputPath, Dataset
    from kfp.dsl import Artifact
    from kfp.dsl import Output, Input

    @component(base_image='python:3.11',
               packages_to_install=["pandas", "fsspec", "gcsfs"])
    def import_data(
        df_output: Output[Dataset]
    ):
        import pandas as pd

        # Specify the correct path to the CSV file in Google Cloud Storage
        df_path = 'gs://ise543_final_pj/Final Project Dataset.csv'


        # Read data from the CSV file
        df = pd.read_csv(df_path)


        # Save the DataFrame to the output path provided by the Kubeflow Pipeline component
        df.to_csv(df_output.path, index=False)
```

Import test data

```
@component(base_image='python:3.11',
           packages_to_install=["pandas", "fsspec", "gcsfs"])
def import_test_data(
    df_test_output: Output[Dataset]
):

    import pandas as pd

    #  test df_path
    df_test_path = 'gs://ise543_final_pj/Final Project Evaluation Dataset - Student(1).csv'


    df_test = pd.read_csv(df_test_path)

    df_test.to_csv(df_test_output.path, index=False)
```

Sperate ID

```
@component(base_image='python:3.11',
           packages_to_install=["pandas"])
def separate_id(
    df_input: Input[Dataset],
    ID_output: Output[Dataset],
    processed_data_output: Output[Dataset]
):
    import pandas as pd

    # Read data
    data = pd.read_csv(df_input.path)


    # Separate out the ID column and assume the column name is 'ID'
    ids = data[['patientID']]
    data_without_id = data.drop(columns=['patientID'])



    # Save the ID column to the specified output path
    ids.to_csv(ID_output.path, index=False)


    # Save processed data without ID to another output path
    data_without_id.to_csv(processed_data_output.path, index=False)
```

Sperate ID in test

```python
@component(base_image='python:3.11',
           packages_to_install=["pandas"])
def separate_id_test(
    df_test_input: Input[Dataset],
    ID_test_output: Output[Dataset],
    processed_test_data_output: Output[Dataset]
):
    import pandas as pd

    data_test = pd.read_csv(df_test_input.path)

    # Separate out the ID column and assume the column name is 'ID'
    ids_test = data_test[['patientID']]
    data_without_id_test = data_test.drop(columns=['patientID'])

    ids_test.to_csv(ID_test_output.path, index=False)

    data_without_id_test.to_csv(processed_test_data_output.path, index=False)
```

Split data into train and val

```python
@component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn"])
def split_data(
    processed_data_input: Input[Dataset],
    X_train_output: Output[Dataset],
    X_val_output: Output[Dataset],
    y_train_output: Output[Dataset],
    y_val_output: Output[Dataset],
    split_details: Output[Artifact],  # Output details of the split



    split_ratio: float = 0.2,
    random_seed: int = 42,
):
    import pandas as pd
    from sklearn.model_selection import train_test_split

    # Load the processed data
    data = pd.read_csv(processed_data_input.path)

    # Separate features and target using 'TenYearCHD' as the target column
    X = data.drop(columns=['TenYearCHD'])
    y = data['TenYearCHD']


    # Perform the split
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=split_ratio, random_state=random
```

```
    # Save the split datasets
    X_train.to_csv(X_train_output.path, index=False)
    X_val.to_csv(X_val_output.path, index=False)
    y_train.to_csv(y_train_output.path, index=False)
    y_val.to_csv(y_val_output.path, index=False)

    # Optionally save some details about the split
    details = f"Train size: {len(X_train)}, Validation size: {len(X_val)}"
    with open(split_details.path, 'w') as f:
        f.write(details)
```

Feature selection

```
from kfp.v2.dsl import component, InputPath, Input, Output, Dataset

@component(base_image='python:3.11',
           packages_to_install=["pandas"])
def feature_selection(
    X_train_input: Input[Dataset],
    X_train_output: Output[Dataset],
    X_val_input: Input[Dataset],
    X_val_output: Output[Dataset],
    X_test_input: Input[Dataset],
    X_test_output: Output[Dataset]

):
    import pandas as pd

    # Use the .path property to read and write data
    X_train = pd.read_csv(X_train_input.path)
    X_val = pd.read_csv(X_val_input.path)

    X_test = pd.read_csv(X_test_input.path)

    # Select specific feature columns
    X_train = X_train[['age', 'sysBP', 'prevalentHyp', 'glucose', 'BPMeds']]
    X_val = X_val[['age', 'sysBP', 'prevalentHyp', 'glucose', 'BPMeds']]

    X_test = X_test[['age', 'sysBP', 'prevalentHyp', 'glucose', 'BPMeds']]

    # Save to the path provided by Dataset
    X_train.to_csv(X_train_output.path, index=False)
    X_val.to_csv(X_val_output.path, index=False)
```

```
    X_test.to_csv(X_test_output.path, index=False)
```

```
<ipython-input-12-4ccef15704c7>:1: DeprecationWarning: The module `kfp.v2` is deprecated and will be re
  from kfp.v2.dsl import component, InputPath, Input, Output, Dataset
```

Data preprocessing

Impute 'BPMeds', 'glucose' in X_train

```
[ ]  from kfp.dsl import component, Input, Output, Dataset, Artifact

     @component(base_image='python:3.11',
                packages_to_install=["pandas"])
     def impute_missing_value_train(
                X_train_input: Input[Dataset],
                X_train_output: Output[Dataset],
                mode_bpm_output: Output[Artifact],
                median_glucose_output: Output[Artifact]
     ):
         import pandas as pd

         # Load data from input paths
         X_train = pd.read_csv(X_train_input.path)  # Use .path to access the dataset

         # Calculate the mode of 'BPMeds' and the median of 'glucose'
         mode_bpm = X_train['BPMeds'].mode()[0]
         median_glucose = X_train['glucose'].median()

         # Write mode and median to their respective output paths
         with open(mode_bpm_output.path, 'w') as file:
             file.write(str(mode_bpm))
         with open(median_glucose_output.path, 'w') as file:
             file.write(str(median_glucose))
```

```
        # Impute missing values in the training dataset
        X_train['BPMeds'] = X_train['BPMeds'].fillna(mode_bpm)
        X_train['glucose'] = X_train['glucose'].fillna(median_glucose)

        # Save the imputed training dataset
        X_train.to_csv(X_train_output.path, index=False)
```

Impute 'BPMeds', 'glucose' in X_val

```
[ ] from kfp.dsl import component, Input, Output, Dataset, Artifact

    @component(base_image='python:3.11',
               packages_to_install=["pandas"])
    def impute_missing_value_val(
               X_val_input: Input[Dataset],
               X_val_output: Output[Dataset],
               mode_bpm_info: Input[Artifact],
               median_glucose_info: Input[Artifact]
    ):
        import pandas as pd

        # Load validation dataset
        X_val = pd.read_csv(X_val_input.path)

        # Read mode and median values from files
        with open(mode_bpm_info.path, 'r') as file:
            mode_bpm = float(file.read())
        with open(median_glucose_info.path, 'r') as file:
            median_glucose = float(file.read())

        # Impute missing values in the validation dataset
        X_val['BPMeds'] = X_val['BPMeds'].fillna(mode_bpm)
        X_val['glucose'] = X_val['glucose'].fillna(median_glucose)

        # Save the imputed validation dataset
        X_val.to_csv(X_val_output.path, index=False)
```

Impute 'BPMeds', 'glucose' in X_test

```
    from kfp.dsl import component, Input, Output, Dataset, Artifact

    @component(base_image='python:3.11',
               packages_to_install=["pandas"])
    def impute_missing_value_test(
               X_test_input: Input[Dataset],
               X_test_output: Output[Dataset],
               mode_bpm_info: Input[Artifact],
               median_glucose_info: Input[Artifact]
    ):
        import pandas as pd

        # Load validation dataset
        X_test = pd.read_csv(X_test_input.path)

        # Read mode and median values from files
        with open(mode_bpm_info.path, 'r') as file:
            mode_bpm = float(file.read())
        with open(median_glucose_info.path, 'r') as file:
            median_glucose = float(file.read())

        # Impute missing values in the validation dataset
        X_test['BPMeds'] = X_test['BPMeds'].fillna(mode_bpm)
        X_test['glucose'] = X_test['glucose'].fillna(median_glucose)

        # Save the imputed validation dataset
        X_test.to_csv(X_test_output.path, index=False)
```

Data quality steps (dealing with outliers and skewness) in X_train

```
[ ] @component(base_image='python:3.11',
               packages_to_install=["pandas", "numpy"])
    def data_quality_train(
        X_train_input: Input[Dataset],
        X_train_output: Output[Dataset],
        IQR_train_output: Output[Artifact]
    ):
        import pandas as pd
        import numpy as np

        X_train = pd.read_csv(X_train_input.path)
        def log_transform(x):
            return np.log1p(x)

        continuous_features = ['age', 'sysBP', 'glucose']
        X_train[continuous_features] = X_train[continuous_features].apply(log_transform)

        Q1 = X_train[continuous_features].quantile(0.25)
        Q3 = X_train[continuous_features].quantile(0.75)
        IQR = Q3 - Q1

        # Explicitly set the index for bounds DataFrame to match continuous_features
        bounds = pd.DataFrame({
            'lower_bound': Q1 - 1.5 * IQR,
            'upper_bound': Q3 + 1.5 * IQR
        }, index=continuous_features)
```

```
        # Save bounds to CSV ensuring index is included
        bounds.to_csv(IQR_train_output.path)

        for feature in continuous_features:
            lower_bound = bounds.loc[feature, 'lower_bound']
            upper_bound = bounds.loc[feature, 'upper_bound']
            X_train[feature] = np.where(X_train[feature] < lower_bound, lower_bound, X_train[feature])
            X_train[feature] = np.where(X_train[feature] > upper_bound, upper_bound, X_train[feature])

        X_train.to_csv(X_train_output.path, index=False)
```

Data quality steps (dealing with outliers and skewness) in X_val

```
[ ] @component(base_image='python:3.11',
               packages_to_install=["pandas", "numpy"])
    def data_quality_val(
        X_val_input: Input[Dataset],
        IQR_train_input: Input[Artifact],
        X_val_output: Output[Dataset]
    ):
        import pandas as pd
        import numpy as np

        X_val = pd.read_csv(X_val_input.path)
        IQR_stats = pd.read_csv(IQR_train_input.path, index_col=0)

        # Check the index after loading
        print("Indices in IQR_stats:", IQR_stats.index)

        def log_transform(x):
            return np.log1p(x)

        continuous_features = ['age', 'sysBP', 'glucose']
        X_val[continuous_features] = X_val[continuous_features].apply(log_transform)

        for feature in continuous_features:
            lower_bound = IQR_stats.at[feature, 'lower_bound']
            upper_bound = IQR_stats.at[feature, 'upper_bound']
            X_val[feature] = np.where(X_val[feature] < lower_bound, lower_bound, X_val[feature])
            X_val[feature] = np.where(X_val[feature] > upper_bound, upper_bound, X_val[feature])

        X_val.to_csv(X_val_output.path, index=False)
```

Data quality steps (dealing with outliers and skewness) in X_test

```
[ ] @component(base_image='python:3.11',
               packages_to_install=["pandas", "numpy"])
    def data_quality_test(
        X_test_input: Input[Dataset],
        IQR_train_input: Input[Artifact],
        X_test_output: Output[Dataset]
    ):
        import pandas as pd
        import numpy as np

        X_test = pd.read_csv(X_test_input.path)
        IQR_stats = pd.read_csv(IQR_train_input.path, index_col=0)

        # Check the index after loading
        print("Indices in IQR_stats:", IQR_stats.index)

        def log_transform(x):
            return np.log1p(x)

        continuous_features = ['age', 'sysBP', 'glucose']
        X_test[continuous_features] = X_test[continuous_features].apply(log_transform)

        for feature in continuous_features:
            lower_bound = IQR_stats.at[feature, 'lower_bound']
            upper_bound = IQR_stats.at[feature, 'upper_bound']
            X_test[feature] = np.where(X_test[feature] < lower_bound, lower_bound, X_test[feature])
            X_test[feature] = np.where(X_test[feature] > upper_bound, upper_bound, X_test[feature])

        X_test.to_csv(X_test_output.path, index=False)
```

Feature engineering

Standardization X_train

```
[ ] @component(base_image='python:3.11',
               packages_to_install=["pandas", "scikit-learn", "joblib"])  # Added joblib for model saving
    def standardize_train(
        X_train_input: Input[Dataset],
        X_train_output: Output[Dataset],
        scaler_output_info: Output[Artifact]
    ):
        import pandas as pd
        from sklearn.preprocessing import StandardScaler
        import joblib  # Import joblib for saving the scaler

        X_train = pd.read_csv(X_train_input.path)
        scaler = StandardScaler()
        continuous_features = ['age', 'sysBP', 'glucose']

        # Fit the scaler and transform the data
        X_train_scaled = scaler.fit_transform(X_train[continuous_features])

        # Convert the scaled array back into a DataFrame
        X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=continuous_features)

        # Combine with non-scaled columns if needed
        X_train[continuous_features] = X_train_scaled_df

        # Save the DataFrame to CSV
        X_train.to_csv(X_train_output.path, index=False)
```

```
    # Save the scaler model to a file
    joblib.dump(scaler, scaler_output_info.path)
```

Standardization X_val

```
@component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn", "joblib"])  # Added joblib for model loading
def standardize_val(
    X_val_input: Input[Dataset],
    X_val_output: Output[Dataset],
    scaler_input_info: Input[Artifact]
):
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    import joblib  # Import joblib for loading the scaler

    X_val = pd.read_csv(X_val_input.path)

    # Load the scaler model from file
    scaler = joblib.load(scaler_input_info.path)
    continuous_features = ['age', 'sysBP', 'glucose']

    # Transform the validation data
    X_val_scaled = scaler.transform(X_val[continuous_features])

    # Convert the scaled array back into a DataFrame
    X_val_scaled_df = pd.DataFrame(X_val_scaled, columns=continuous_features)

    # Combine with non-scaled columns if needed
    X_val[continuous_features] = X_val_scaled_df

    # Save the DataFrame to CSV
    X_val.to_csv(X_val_output.path, index=False)
```

Standardization X_test

```
[ ] @component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn", "joblib"])  # Added joblib for model loading
def standardize_test(
    X_test_input: Input[Dataset],
    X_test_output: Output[Dataset],
    scaler_input_info: Input[Artifact]
):
    import pandas as pd
    from sklearn.preprocessing import StandardScaler
    import joblib  # Import joblib for loading the scaler

    X_test = pd.read_csv(X_test_input.path)

    # Load the scaler model from file
    scaler = joblib.load(scaler_input_info.path)
    continuous_features = ['age', 'sysBP', 'glucose']

    # Transform the validation data
    X_test_scaled = scaler.transform(X_test[continuous_features])

    # Convert the scaled array back into a DataFrame
    X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=continuous_features)

    # Combine with non-scaled columns if needed
    X_test[continuous_features] = X_test_scaled_df

    # Save the DataFrame to CSV
    X_test.to_csv(X_test_output.path, index=False)
```

Apply oversampling method (SMOTE) only on training set

```
@component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn","imbalanced-learn"])  # Correct package name
def oversampling(
    X_train_input: Input[Dataset],
    y_train_input: Input[Dataset],  # Adding y_train as an input
    X_train_output: Output[Dataset],
    y_train_output: Output[Dataset],  # Adding an output for the resampled y_train
    random_state: int = 42
):
    import pandas as pd
    from imblearn.over_sampling import SMOTE

    # Load input data
    X_train = pd.read_csv(X_train_input.path)
    y_train = pd.read_csv(y_train_input.path)  # Assuming y_train is also a CSV file

    # Create a SMOTE instance with the provided random state
    sm = SMOTE(random_state=random_state)

    # Apply SMOTE resampling
    X_train_SMOTE, y_train_SMOTE = sm.fit_resample(X_train, y_train)

    # Save the resampled data
    X_train_SMOTE.to_csv(X_train_output.path, index=False)
    y_train_SMOTE.to_csv(y_train_output.path, index=False)  # Save the resampled target variable
```

Modeling

Logistic regression

Train lr

```
[ ] from kfp.dsl import Metrics
    from kfp.dsl import Model

    @component(base_image='python:3.11', packages_to_install=["pandas", "scikit-learn", "joblib"])
    def train_logistic_regression(
        X_train_input: Input[Dataset],
        y_train_input: Input[Dataset],
        logistic_model_output: Output[Model]  # Corrected the name here for clarity
    ):

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


        # Load the training data
        X_train_SMOTE = pd.read_csv(X_train_input.path)
        y_train_SMOTE = pd.read_csv(y_train_input.path)

        # Create a logistic regression model
        logistic_model = LogisticRegression()
```

```
    logistic_model.fit(X_train_SMOTE, y_train_SMOTE)


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

Evaluate lr on val

```
from kfp.dsl import Metrics

@component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn", "joblib"])
def evaluate_logistic_regression(
    X_val_input: Input[Dataset],
    y_val_input: Input[Dataset],
    logistic_model_input: Input[Model],
    metrics: Output[Metrics]
):

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

    # Load the validation data
    X_val_scaled = pd.read_csv(X_val_input.path)
    y_val = pd.read_csv(y_val_input.path)



    # Load the model
    logistic_model_file_path = logistic_model_input.path + "/model.joblib"
    logistic_trained_model = joblib.load(logistic_model_file_path)

    # Make predictions on the validation data
    y_val_pred = logistic_trained_model.predict(X_val_scaled)
    y_val_probs = logistic_trained_model.predict_proba(X_val_scaled)[:, 1]  # probabilities for the po
```

```
    # Calculate metrics
    logistic_accuracy = accuracy_score(y_val, y_val_pred)
    logistic_precision = precision_score(y_val, y_val_pred, average='weighted')
    logistic_recall = recall_score(y_val, y_val_pred, average='weighted')
    logistic_f1 = f1_score(y_val, y_val_pred, average='weighted')
    logistic_auc = roc_auc_score(y_val, y_val_probs)

    # Confusion matrix
    logistic_conf_matrix = confusion_matrix(y_val, y_val_pred)

    # Precision-recall curve and AUC-PR
    precision, recall, _ = precision_recall_curve(y_val, y_val_probs)
    logistic_auc_pr = auc(recall, precision)

    # Log metrics
    metrics.log_metric("Accuracy", logistic_accuracy)
    metrics.log_metric("Precision", logistic_precision)
    metrics.log_metric("Recall", logistic_recall)
    metrics.log_metric("F1 Score", logistic_f1)
    metrics.log_metric("AUC-ROC", logistic_auc)
    metrics.log_metric("AUC-PR", logistic_auc_pr)
```

Random forest

Train rf

```
@component(base_image='python:3.11', packages_to_install=["pandas", "scikit-learn", "joblib"])
def train_random_forest(
    X_train_input: Input[Dataset],
    y_train_input: Input[Dataset],
    rf_model_output: Output[Model]  # Corrected the name here for clarity
):

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

    # Load the training data
    X_train_SMOTE = pd.read_csv(X_train_input.path)
    y_train_SMOTE = pd.read_csv(y_train_input.path)

    # Create a random forest model
    rf_model = RandomForestClassifier()

    rf_model.fit(X_train_SMOTE, y_train_SMOTE)

    # Save the model to gcs
    os.makedirs(rf_model_output.path, exist_ok=True)
    joblib.dump(rf_model, os.path.join(rf_model_output.path, "model.joblib"))
```

Evaluate rf on val

```
@component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn", "joblib"])
def evaluate_random_forest(
    X_val_input: Input[Dataset],
    y_val_input: Input[Dataset],
    rf_model_input: Input[Model],
    metrics: Output[Metrics]
):

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

    # Load the validation data
    X_val_scaled = pd.read_csv(X_val_input.path)
    y_val = pd.read_csv(y_val_input.path)

    # Load the model
    rf_model_file_path = rf_model_input.path + "/model.joblib"
    rf_trained_model = joblib.load(rf_model_file_path)

    # Make predictions on the validation data
    y_val_pred = rf_trained_model.predict(X_val_scaled)
    y_val_probs = rf_trained_model.predict_proba(X_val_scaled)[:, 1]  # probabilities for the positive
```

```
    # Calculate metrics
    rf_accuracy = accuracy_score(y_val, y_val_pred)
    rf_precision = precision_score(y_val, y_val_pred, average='weighted')
    rf_recall = recall_score(y_val, y_val_pred, average='weighted')
    rf_f1 = f1_score(y_val, y_val_pred, average='weighted')
    rf_auc = roc_auc_score(y_val, y_val_probs)

    # Confusion matrix
    rf_conf_matrix = confusion_matrix(y_val, y_val_pred)

    # Precision-recall curve and AUC-PR
    precision, recall, _ = precision_recall_curve(y_val, y_val_probs)
    rf_auc_pr = auc(recall, precision)

    # Log metrics
    metrics.log_metric("Accuracy", rf_accuracy)
    metrics.log_metric("Precision", rf_precision)
    metrics.log_metric("Recall", rf_recall)
    metrics.log_metric("F1 Score", rf_f1)
    metrics.log_metric("AUC-ROC", rf_auc)
    metrics.log_metric("AUC-PR", rf_auc_pr)
```

Predict_and_combine

```
@component(base_image='python:3.11',
           packages_to_install=["pandas", "scikit-learn", "joblib", "gcsfs"])
def predict_and_combine(
    test_features_path: InputPath('Dataset'),
    patient_ids_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:
      model = joblib.load(f)

    # Load the test features
    test_features = pd.read_csv(test_features_path)

    # Load patient IDs
    patient_ids = pd.read_csv(patient_ids_path)

    # Make predictions
    predictions = model.predict(test_features)
```

```
    # Combine patientID and predictions
    results = pd.DataFrame({
        'patientID': patient_ids['patientID'],
        'TenYearCHD': predictions
    })

    # Save the results to the specified path
    results.to_csv(predictions_path, index=False)
```

Define pipeline

```
@pipeline(name='CHD_pipeline')
def CHD_pipeline():
    # Import data
    import_data_task = import_data()

    # Import test data
    import_test_data_task = import_test_data()

    # Sperate ID
    separate_id_task = separate_id(
        df_input=import_data_task.outputs['df_output']

    )

    separate_id_test_task = separate_id_test(
        df_test_input=import_test_data_task.outputs['df_test_output']
    )

    # drop_target_test_task = separate_id_test(
    #     processed_test_data_input=separate_id_test_task.outputs['processed_test_data_output']
    # )

    # Split the dataset into training and validation sets
    split_data_task = split_data(
        processed_data_input = separate_id_task. outputs['processed_data_output']
    )
```

```
# Feature selection
feature_selection_task = feature_selection(
    X_train_input=split_data_task.outputs['X_train_output'],
    X_val_input=split_data_task.outputs['X_val_output'],
    X_test_input=separate_id_test_task.outputs['processed_test_data_output']
)

# Data preprocessing
impute_missing_value_train_task = impute_missing_value_train(
    X_train_input=feature_selection_task.outputs['X_train_output']
)

impute_missing_value_val_task = impute_missing_value_val(
    X_val_input=feature_selection_task.outputs['X_val_output'],
    mode_bpm_info=impute_missing_value_train_task.outputs['mode_bpm_output'],
    median_glucose_info=impute_missing_value_train_task.outputs['median_glucose_output']
)

impute_missing_value_test_task = impute_missing_value_test(
    X_test_input=feature_selection_task.outputs['X_test_output'],
    mode_bpm_info=impute_missing_value_train_task.outputs['mode_bpm_output'],
    median_glucose_info=impute_missing_value_train_task.outputs['median_glucose_output']
)
```

```
data_quality_train_task = data_quality_train(
    X_train_input=impute_missing_value_train_task.outputs['X_train_output']
)

data_quality_val_task = data_quality_val(
    X_val_input=impute_missing_value_val_task.outputs['X_val_output'],
    IQR_train_input=data_quality_train_task.outputs['IQR_train_output']
)

data_quality_test_task = data_quality_test(
    X_test_input=impute_missing_value_test_task.outputs['X_test_output'],
    IQR_train_input=data_quality_train_task.outputs['IQR_train_output']
)


standardize_train_task = standardize_train(
    X_train_input=data_quality_train_task.outputs['X_train_output']
)

standardize_val_task = standardize_val(
    X_val_input=data_quality_val_task.outputs['X_val_output'],
    scaler_input_info=standardize_train_task.outputs['scaler_output_info']
)

standardize_test_task = standardize_test(
    X_test_input=data_quality_test_task.outputs['X_test_output'],
    scaler_input_info=standardize_train_task.outputs['scaler_output_info']
)
```

```
oversampling_task = oversampling(
    X_train_input=standardize_train_task.outputs['X_train_output'],
    y_train_input=split_data_task.outputs['y_train_output']
)

# Train the model
train_logistic_regression_task = train_logistic_regression(
    X_train_input=oversampling_task.outputs['X_train_output'],
    y_train_input=oversampling_task.outputs['y_train_output']
)

train_random_forest_task = train_random_forest(
    X_train_input=oversampling_task.outputs['X_train_output'],
    y_train_input=oversampling_task.outputs['y_train_output']
)



# Evaluate the model
evaluate_logistic_regression_task = evaluate_logistic_regression(
    X_val_input=standardize_val_task.outputs['X_val_output'],
    y_val_input=split_data_task.outputs['y_val_output'],
    logistic_model_input=train_logistic_regression_task.outputs['logistic_model_output']
)

evaluate_random_forest_task = evaluate_random_forest(
    X_val_input=standardize_val_task.outputs['X_val_output'],
    y_val_input=split_data_task.outputs['y_val_output'],
    rf_model_input=train_random_forest_task.outputs['rf_model_output']
)
```

```
    # Predict and combine
    predict_and_combine_task = predict_and_combine(
        test_features_path=standardize_test_task.outputs['X_test_output'],
        patient_ids_path=separate_id_test_task.outputs['ID_test_output'],
        model_path='gs://ise543_final_pj/125143825809/chd-pipeline-20240501211757/train-random-forest_

    )
```

Compile and run pipeline

```
from kfp import compiler

compiler.Compiler().compile(
    pipeline_func=CHD_pipeline,
    package_path = 'CHD_pipeline.json'
)

pipeline_job = aiplatform.PipelineJob(
    display_name='CHD_pipeline',
    template_path='CHD_pipeline.json',
    pipeline_root='gs://ise543_final_pj',

    enable_caching=True
)

pipeline_job.run()
```

```
INFO:google.cloud.aiplatform.pipeline_jobs:Creating PipelineJob
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob created. Resource name: projects/125143825809/lo
INFO:google.cloud.aiplatform.pipeline_jobs:To use this PipelineJob in another session:
INFO:google.cloud.aiplatform.pipeline_jobs:pipeline_job = aiplatform.PipelineJob.get('projects/12514382
INFO:google.cloud.aiplatform.pipeline_jobs:View Pipeline Job:
https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/chd-pipeline-2024050203
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob projects/125143825809/locations/us-central1/pipe
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob run completed. Resource name: projects/125143825
```

Summary Discussion

1. I separated patientID and saved them for the final prediction after importing the training and test data since patientID did not contribute to the response variable in the larter modeling.
2. I splitted the training data into training and validation datasets for the future comparison among different model and would choose the best performance model for the final prediction.
3. I did the EDA only based on the training dataset to the avoid potential data leakage.
4. I selected the same features for training, validation, test dataset mainly based on the heat map I drew in the EDA section.
5. In the data preprocessing part, I calculated the mode and median for discrete and continuous variables respectively in the training dataset.

6. I imputed missing values in the training data, and transferred the caculation results to the validation and test dataset to prevent the potential data leakage.
7. Besides, I applied IQR to deal with outliers for continuous variables. I set the standard of IQR according to the training dataset to avoid potential data leakage as well.
8. I cut the outliers directly in the training set and transferred the same IQR method to the validation and test dataset to ensure consistency in data processing.
9. In the respective of dealing with skewedness, I simply applied the log-transforamtion method directly to the training, validation and test datasets respectively since this step will not cause data leakage.
10. In the feature engineering section, I applied standard scaler in the training dataset and transferred the same scaler to the validation and test dataset to eliminated the disturbed effect due the different scale of features, improving the future model performance.
11. I applied SOMTE method to mitigate the effect of imbalanced data only the training dataset because either the validation or the test dataset has to be preserved real-word distribution.
12. In the modeling part, I tried the logistic regression and random forest model because one is good at predicting the linear relationship and the other one is good at predicting the nonlinear relationship. So, I thought it was reasonable to consider and compare the performance of these two models in the early stages.
13. I evaluated and compared the models performance on the validation dataset, and found that the random forest model had a better performance.
14. Therefore, I combined the separated patientID file and the remaining feature file from the test dataset to make the later final prediction.
15. Finally, I chose to use the trained random forest model to make the final predictions on the features of the test dataset and I got the TenYearCHD status corresponding to each patient.

Future Work

1. Try more models.
2. Hyperparameter tuning.
3. Optimize pipeline structure.