Coronary Heart Disease Prediction Final Report

Yan Fang

8872881296

Objects

Predicting

Predicting the 10-year risk of coronary heart disease (TenYearCHD as target).

Analyzing

Analyzing factors that influence heart disease risk.

Dataset Summary

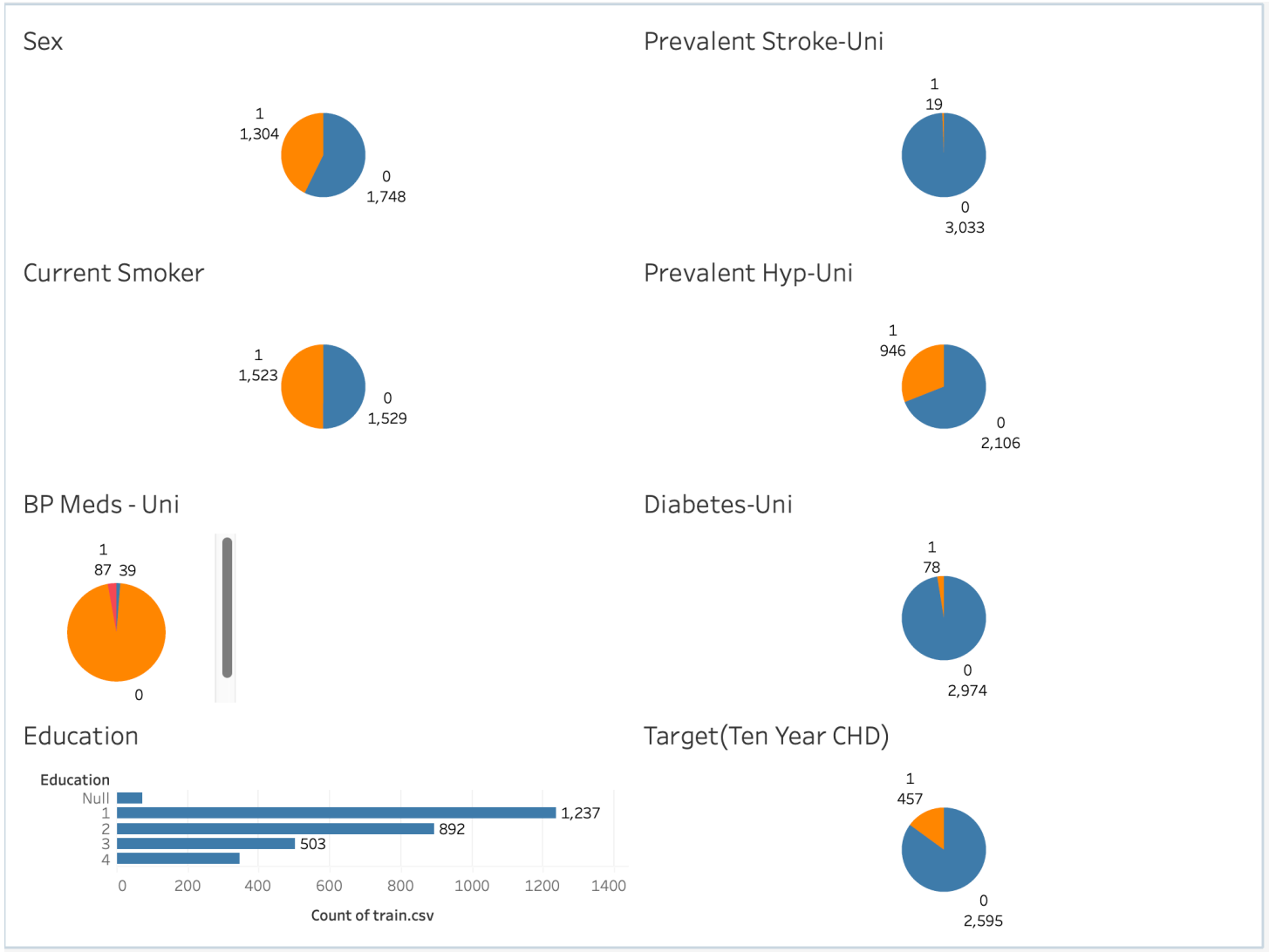
Data Quality

- 1. The cigsPerDay field has a significant number of missing values, which can be attributed to its association with the currentSmoker field. In cases where currentSmoker is 0, indicating the person does not smoke, the cigsPerDay might be left blank.
- 2. Outliers will be discussed along our Univariate Analysis

```
male                0
age                 0
education          72
currentSmoker       0
cigsPerDay       1549
BPMeds             39
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            38
sysBP               0
diaBP               0
BMI                15
heartRate           1
glucose           292
a1c               292
income              0
dtype: int64
```

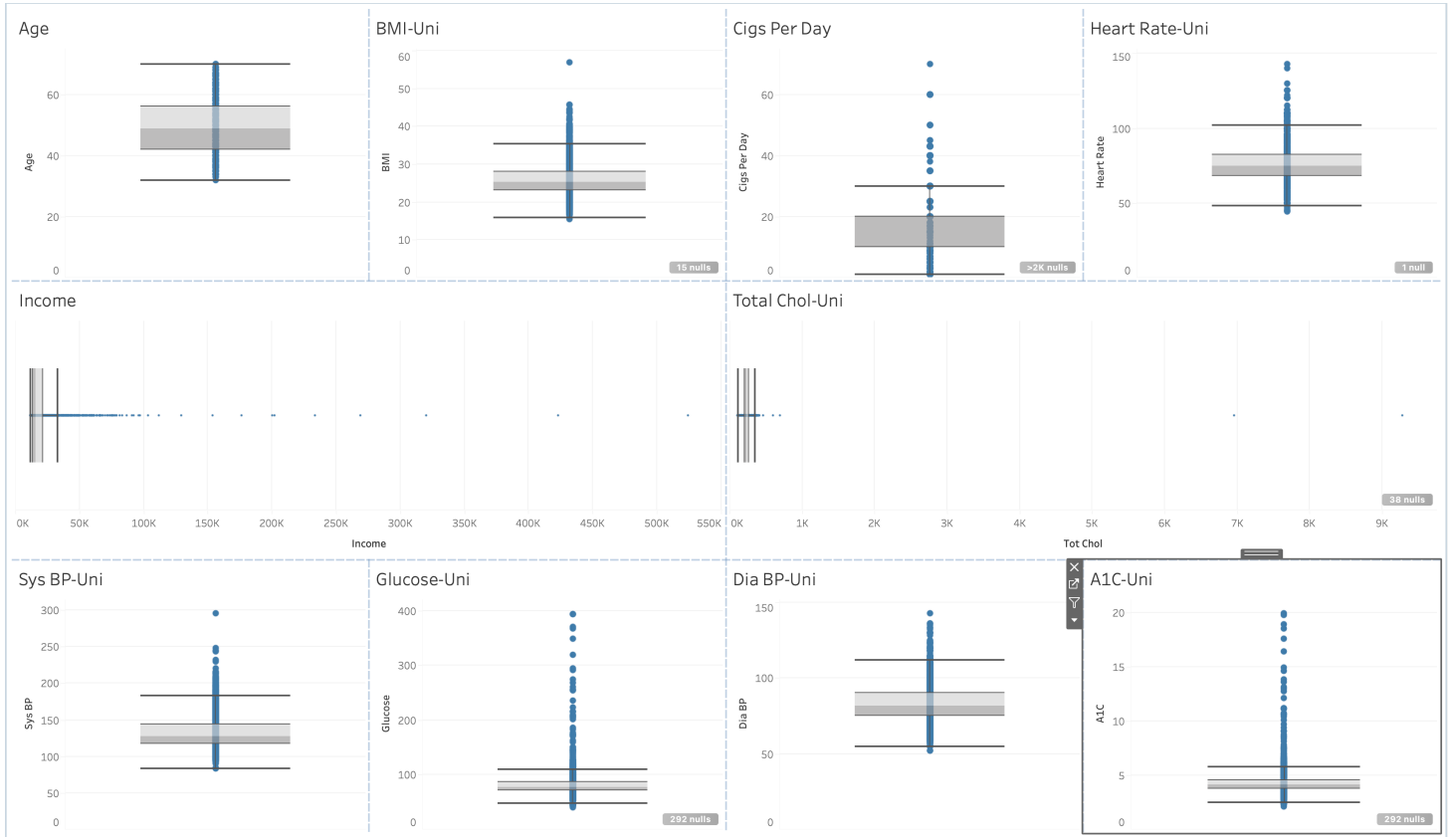
Univariate Analysis – Categorical

- The features such as prevalentStroke, diabetes, and BP Meds are highly imbalanced, and our target variable is also imbalanced. We might need to use oversampling techniques before training our model. Additionally, it would be beneficial to select models that are effective at managing imbalanced features such as



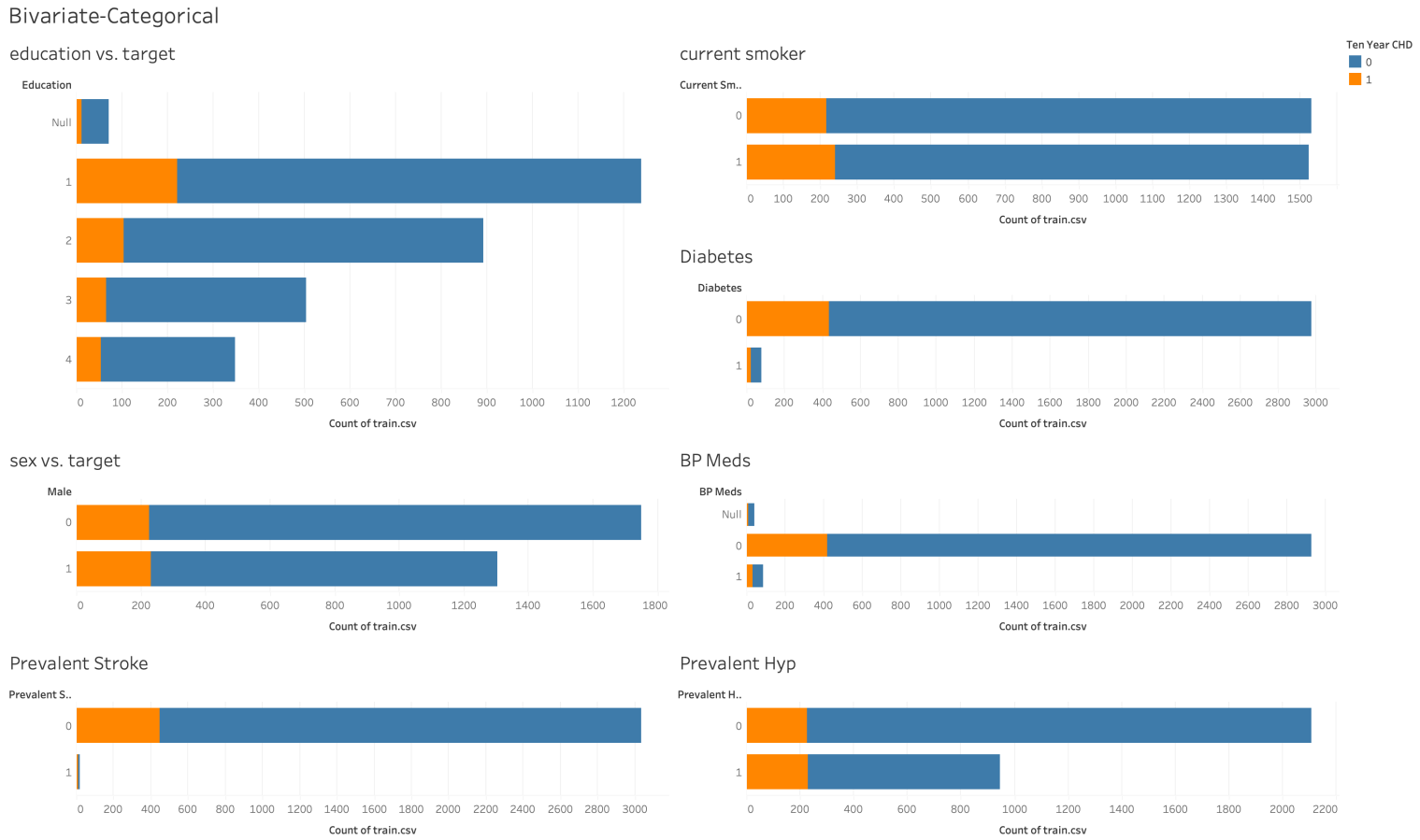
Univariate Analysis - Numerical

- Income and totChol have several extreme outliers; we may need to apply winsorizing to these data points.
- Most features, except for Age, are somewhat skewed and might require a log transformation to reduce bias towards the majority class.



Bivariate Analysis-Categorical

- Education vs. Target: Higher education levels (3 and 4) are associated with fewer CHD cases compared to lower levels (1 and 2). There is also a category for missing data ('Null').
- Sex vs. Target: Males show a higher incidence of both CHD and non-CHD cases than females.
- Prevalent Stroke vs. Target: Non-stroke individuals outnumber stroke patients significantly, but stroke patients have a higher rate of CHD.
- Current Smoker vs. Target: Both groups are well-represented; however, smokers tend to have a slightly higher CHD rate than non-smokers.
- Diabetes vs. Target: Non-diabetic individuals are more common, but diabetics show a higher CHD rate.
- BP Meds vs. Target: More individuals are off BP meds, but those on meds have a higher incidence of CHD. Missing data is noted.
- Prevalent Hypertension vs. Target: Those without prevalent hypertension are more common, yet those with it have a higher CHD rate.



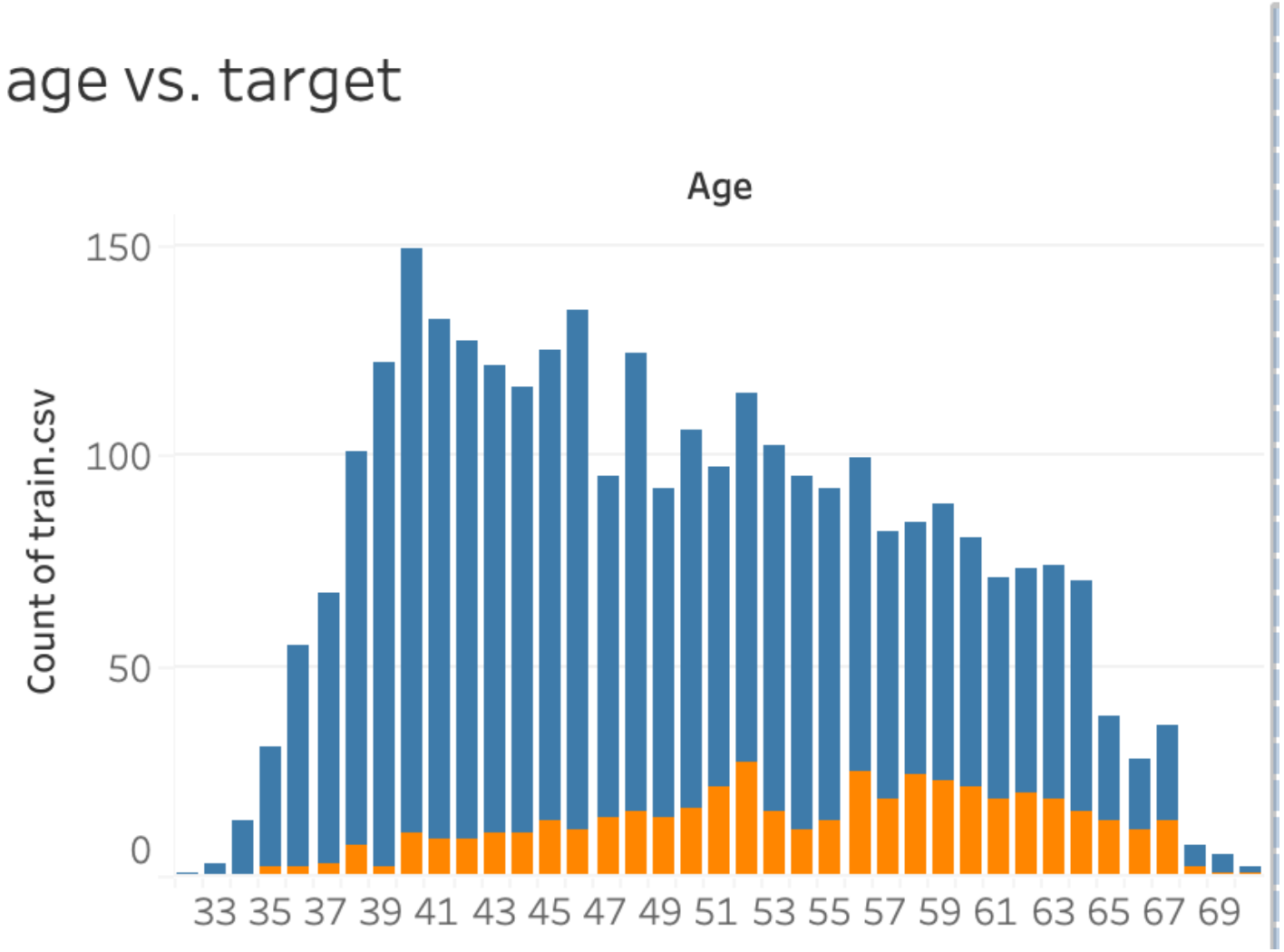
Bivariate Analysis- Numerical

Summary and Potential feature selection

- **Age**: There's a clear trend where the incidence of CHD increases with age, particularly noticeable after the mid-50s. **This is potentially an important feature to classifying CHD**
- **Income**: Most of the data falls in the lower income brackets, with a small number of CHD cases spread across various income levels. The incidence of CHD doesn't show a clear pattern relative to income based on this chart.
- **Cigarettes per Day:** Most people either don't smoke or smoke about 20 cigarettes per day. There is a notable number of CHD cases among heavier smokers, suggesting a possible link between smoking more cigarettes and higher CHD risk.
- **A1C:** The A1C are mostly normal excluding null values but it shows some instances of CHD as A1C levels rise, indicating a potential risk factor for CHD with higher A1C levels.
- **Total Cholesterol (totChol):** The data categorizes cholesterol levels from normal to high. Individuals with high and abnormally high cholesterol levels show a significant number of CHD cases, emphasizing high cholesterol as a risk factor for CHD.
- **Systolic Blood Pressure (Sys BP)**: Higher systolic blood pressure categories show more CHD cases. This suggests that higher BP is associated with an increased risk of CHD.
- **Diastolic Blood Pressure (Dia BP**): Similar to systolic BP, higher diastolic blood pressure seems to correlate with a higher incidence of CHD, especially noticeable in the higher blood pressure ranges.
- **BMI**: The distribution primarily concentrates around a BMI of 20 to 30, with CHD cases more frequent in the higher BMI categories, suggesting a link between higher BMI and CHD risk.
- **Heart Rate**: A majority of the dataset falls within a normal heart rate range, with some CHD cases appearing more frequently at higher rates, indicating a potential association between higher heart rates and CHD risk.
- **Glucose**: Glucose is strongly correlated to A1C, its levels show a normal distribution with a few instances of CHD cases at higher glucose levels, pointing to high glucose as a risk factor for CHD.

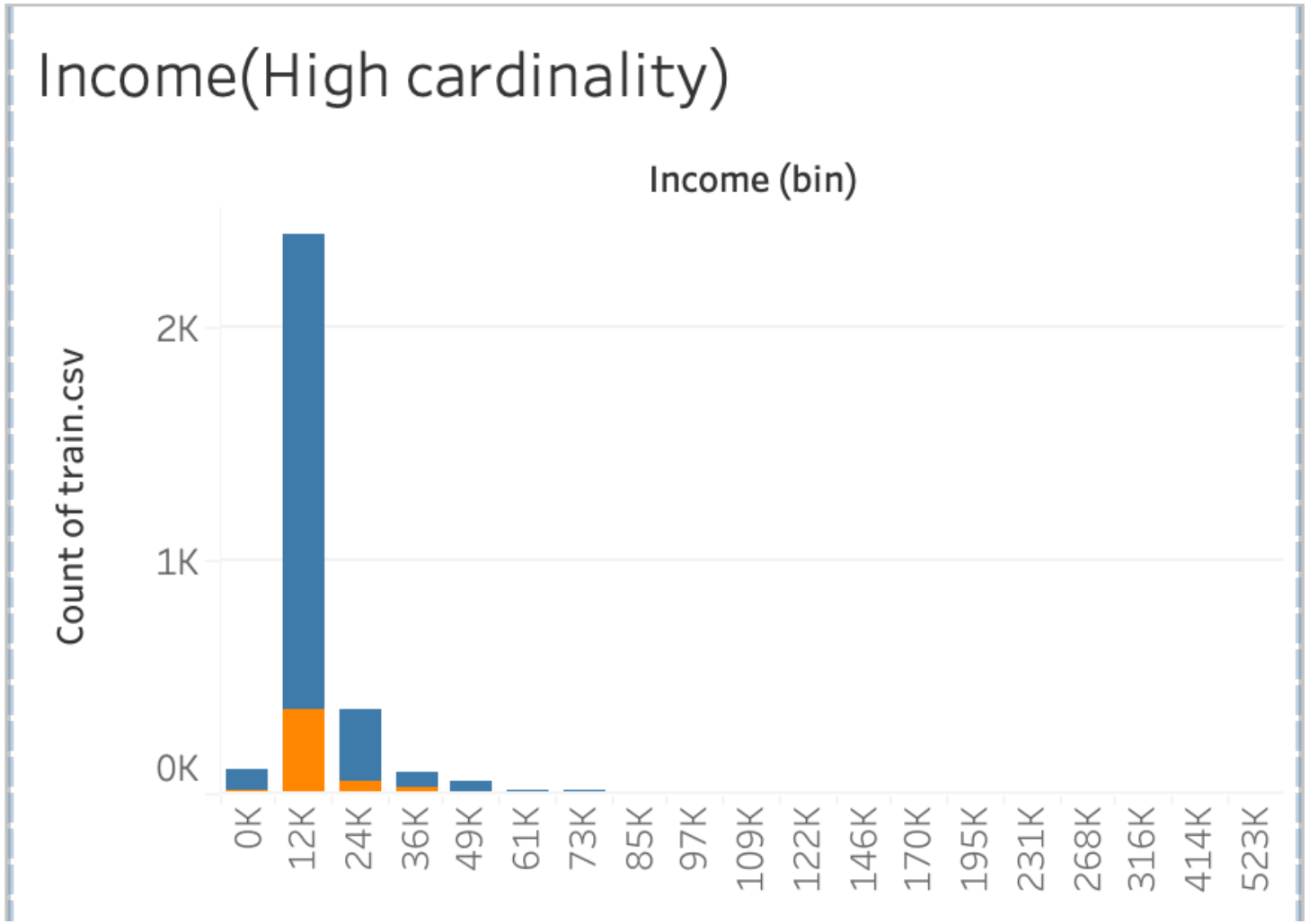
Age vs. TenYearCHD

- There's a clear trend where the incidence of CHD increases with age, particularly noticeable after the mid-50s. **This is potentially an important feature to classifying CHD**



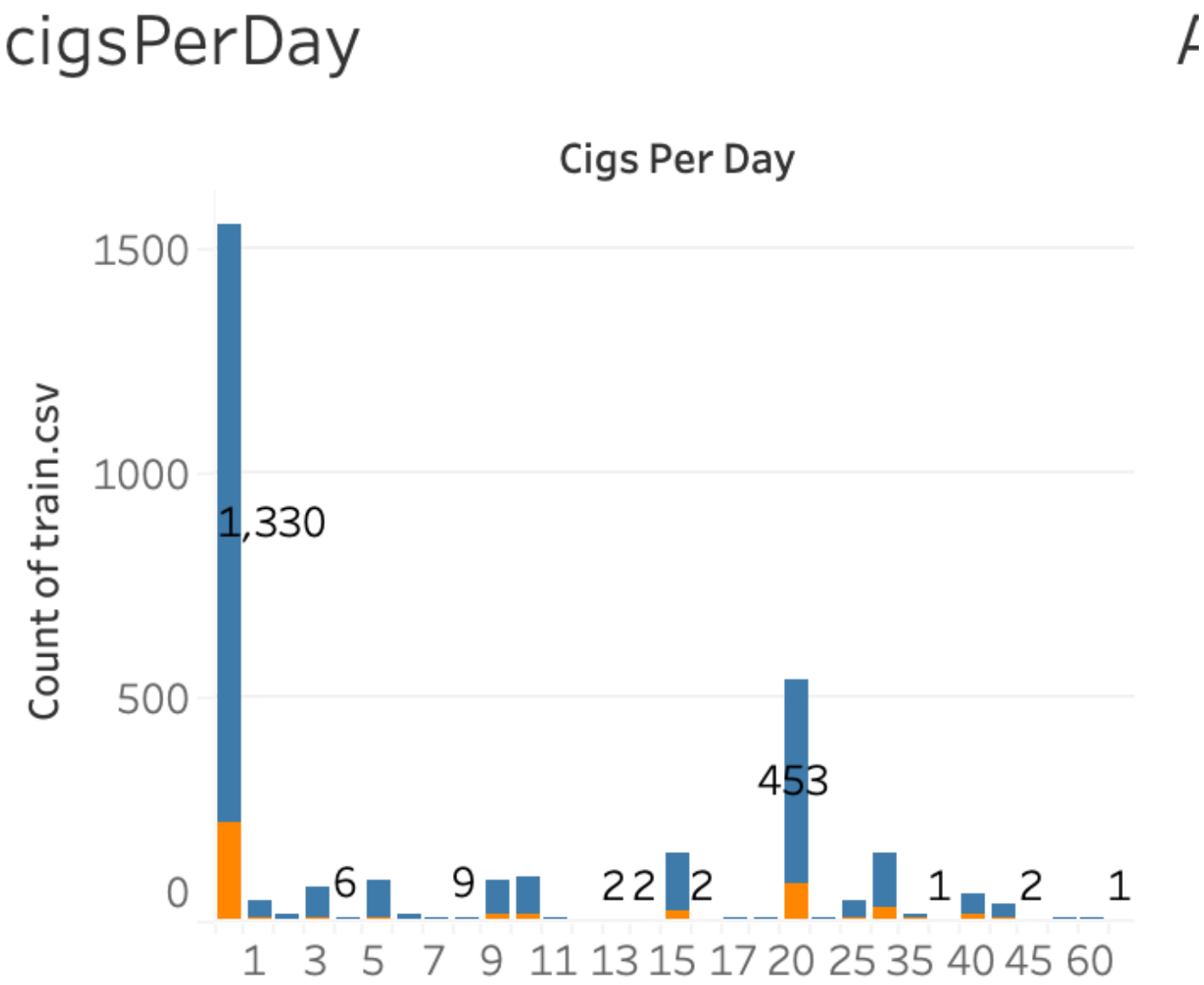
Income vs. Target

- Most of the data falls in the lower income brackets, with a small number of CHD cases spread across various income levels. The incidence of CHD doesn't show a clear pattern relative to income based on this chart.



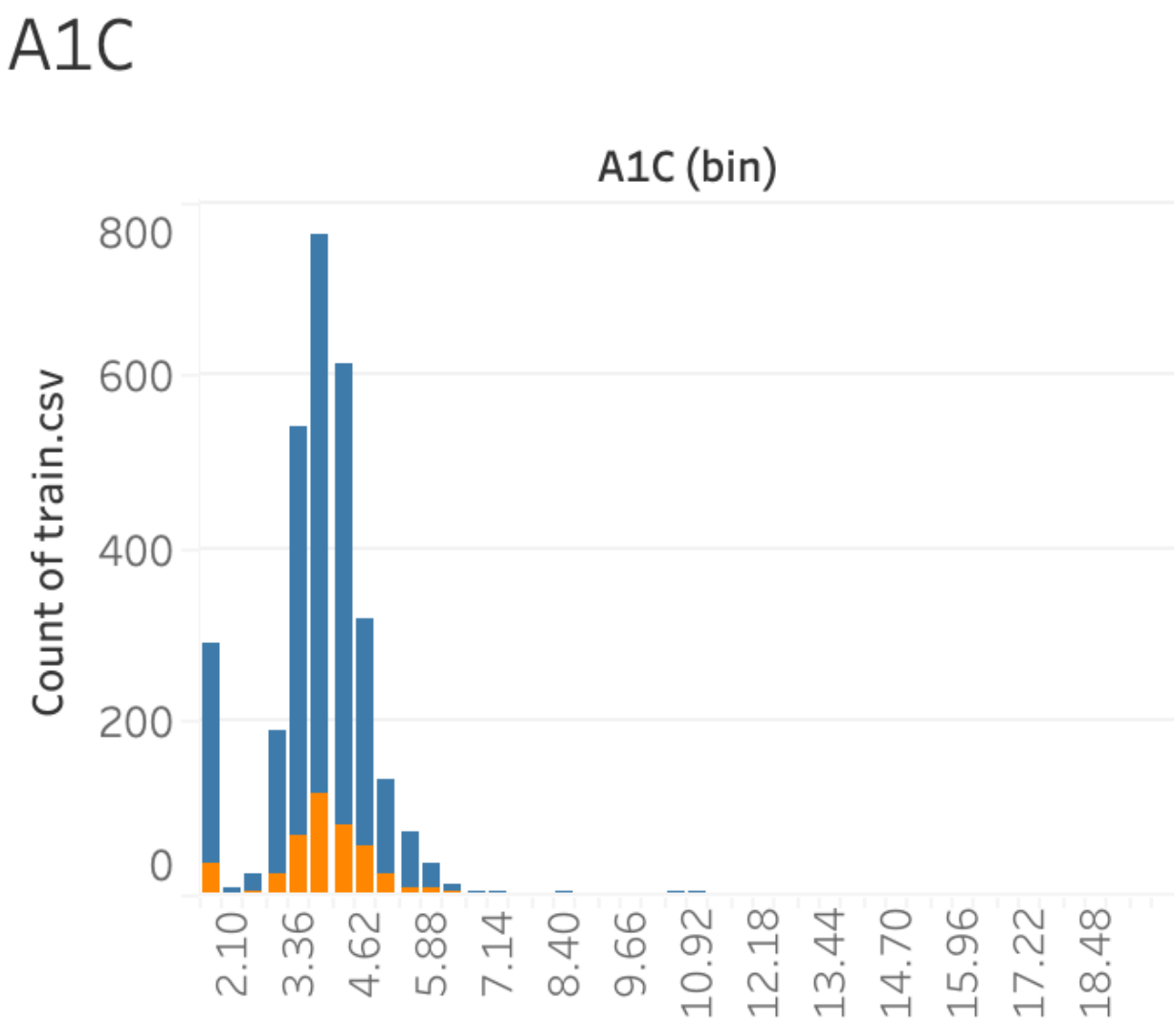
Cigarettes per Day vs. Target

- Most people either don't smoke or smoke about 20 cigarettes per day. There is a notable number of CHD cases among heavier smokers, suggesting a possible link between smoking more cigarettes and higher CHD risk.



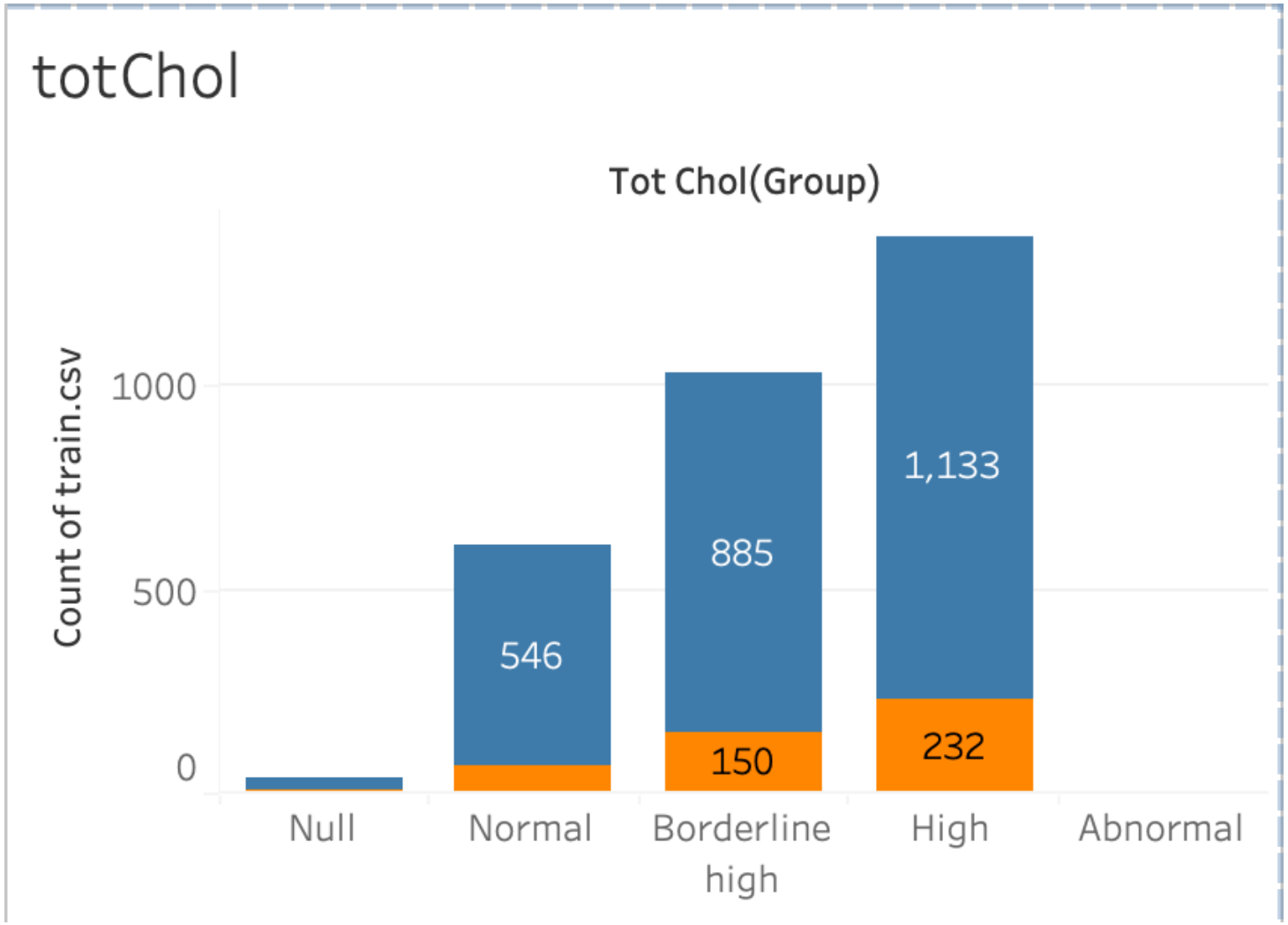
A1C vs. Target

- The A1C are mostly normal excluding null values but it shows some instances of CHD as A1C levels rise, indicating a potential risk factor for CHD with higher A1C levels.



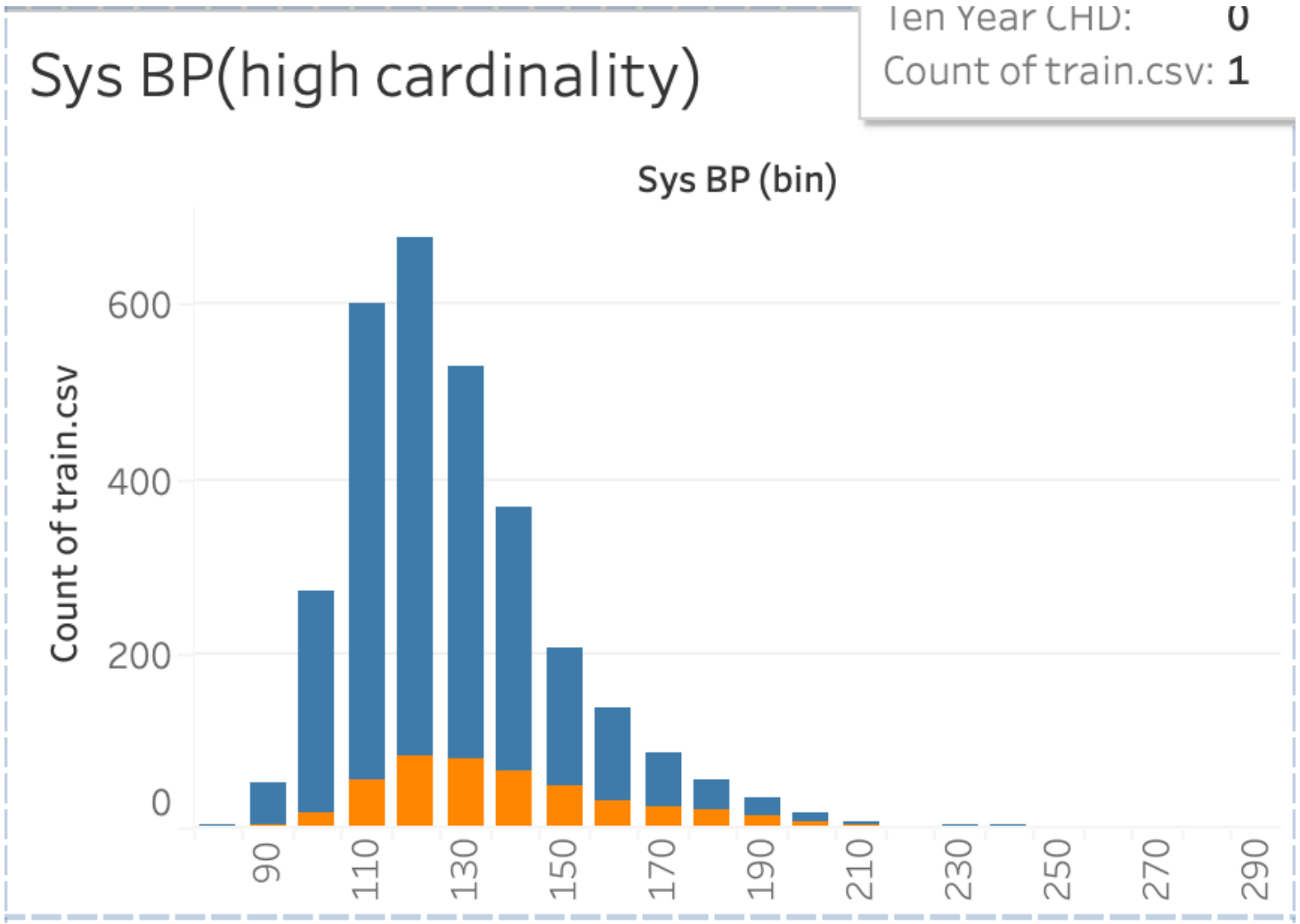
Total Cholesterol (totChol):

- The data categorizes cholesterol levels from normal to high. Individuals with high and abnormally high cholesterol levels show a significant number of CHD cases, emphasizing high cholesterol as a risk factor for CHD.



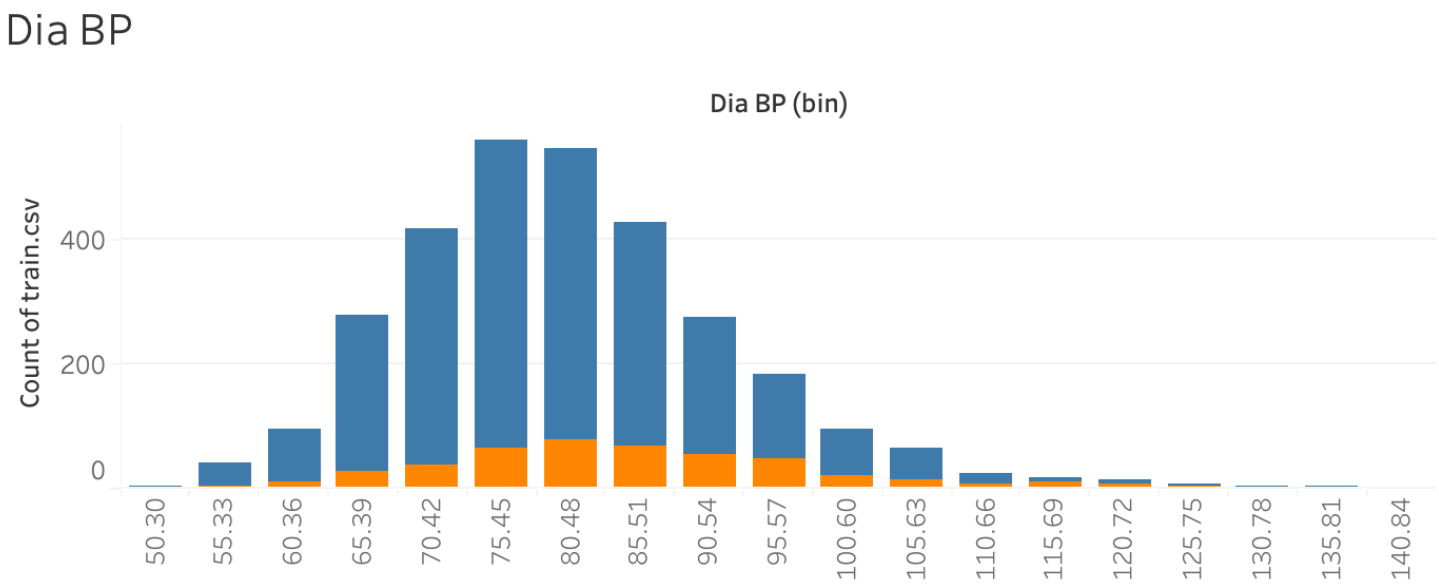
Systolic Blood Pressure(sysBP) vs. Target

- Higher systolic blood pressure categories show more CHD cases. This suggests that higher BP is associated with an increased risk of CHD.



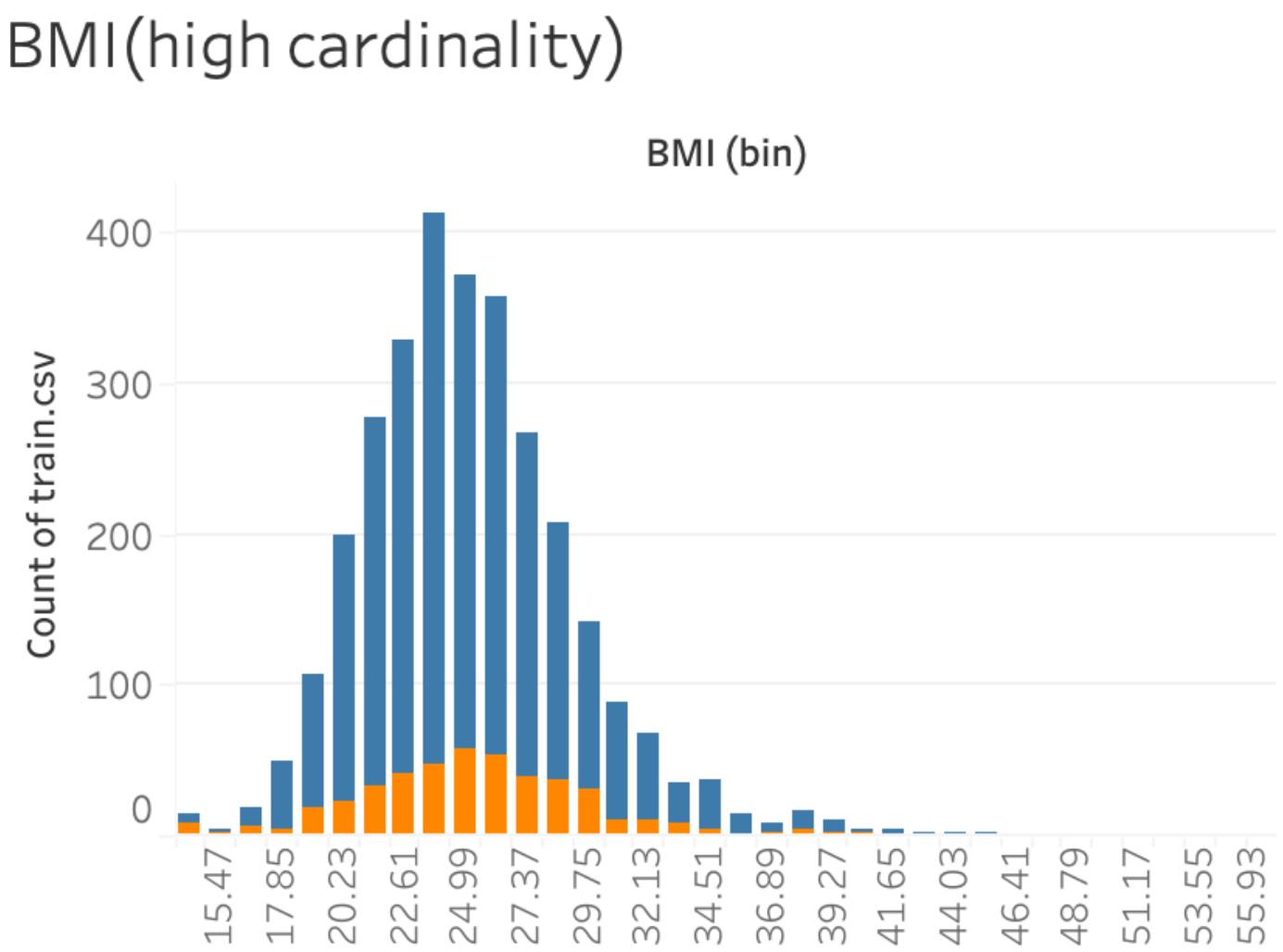
Diastolic Blood Pressure (Dia BP) vs. Target

- Similar to systolic BP, higher diastolic blood pressure seems to correlate with a higher incidence of CHD, especially noticeable in the higher blood pressure ranges.



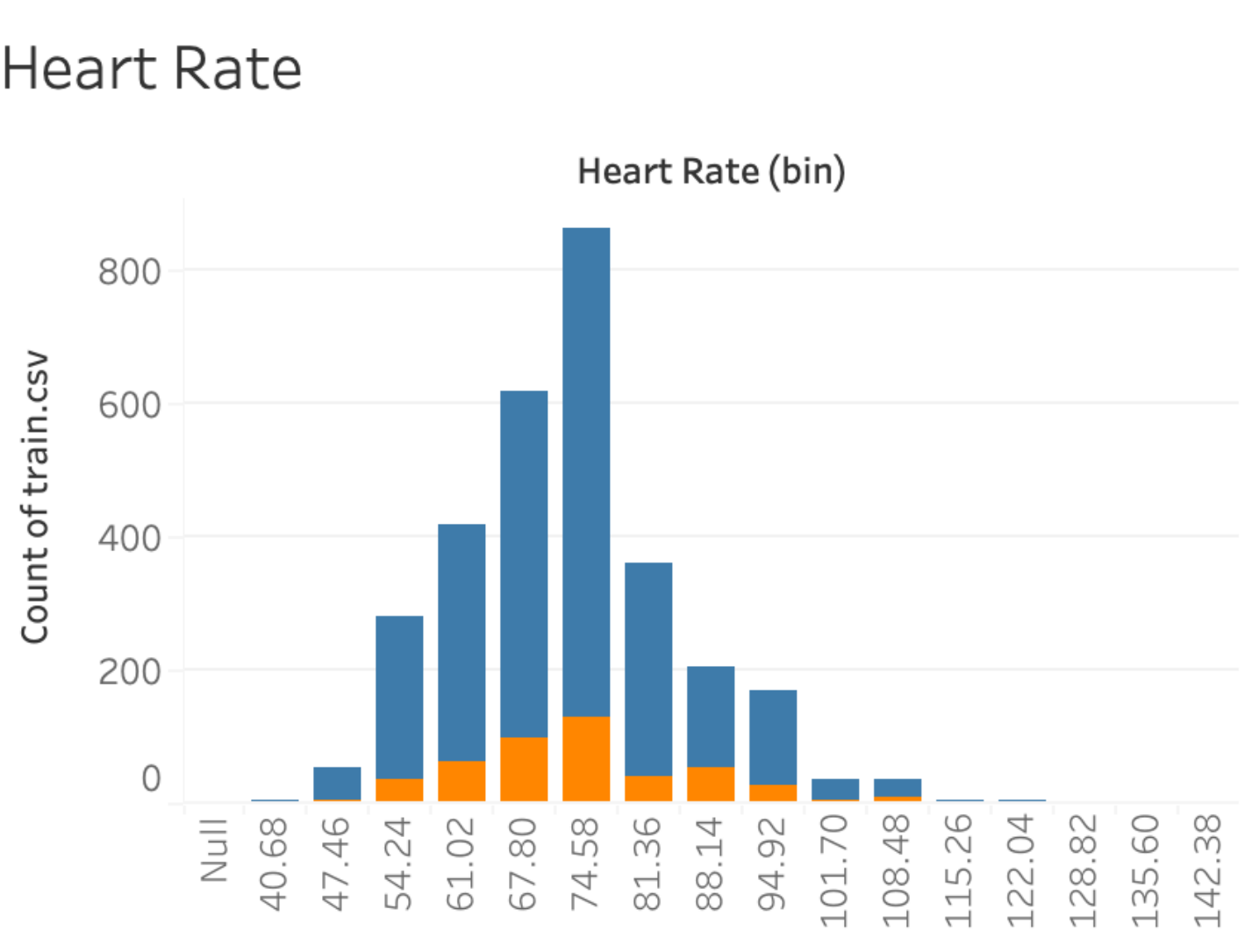
BMI vs. Target

- The distribution primarily concentrates around a BMI of 20 to 30, with CHD cases more frequent in the higher BMI categories, suggesting a link between higher BMI and CHD risk.



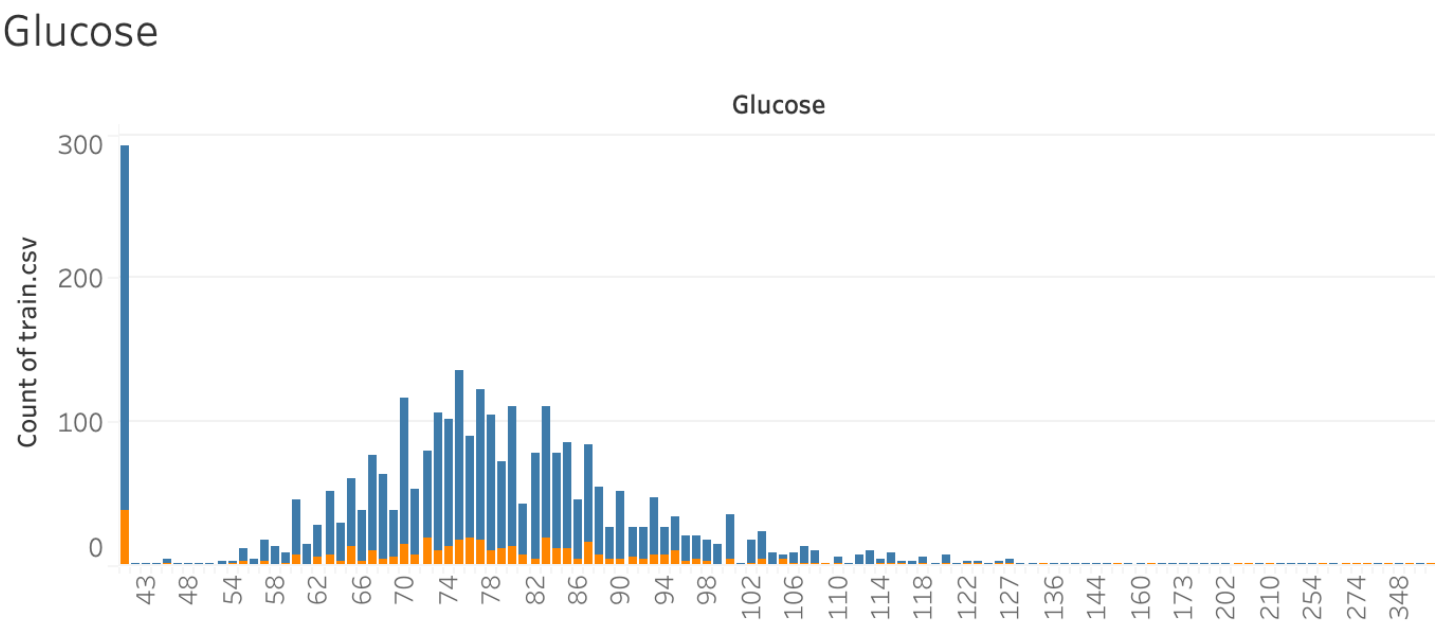
Heart Rate vs. Target

- A majority of the dataset falls within a normal heart rate range, with some CHD cases appearing more frequently at higher rates, indicating a potential association between higher heart rates and CHD risk.



Glucose vs. Target

- Glucose is strongly correlated to A1C, its levels show a normal distribution with a few instances of CHD cases at higher glucose levels, pointing to high glucose as a risk factor for CHD.



Data preparation plan

Missing Value Imputation	Impute cigsPerDay with 0 and others with median
Handling Skewed Data	Skewed data can distort the model's understanding of the true underlying patterns, leading to biased predictions
Correlation Analysis and Feature Heatmap	Helps us decide feature extraction or dropping redundant features
Handling High Cardinality	Income has very high cardinality.
Scaling Numerical Features	All numerical features will need scaling to ensure that they contribute equally to model training, preventing any single feature with larger scale from dominating the model's learning process.
Implementing Sampling techniques	Consider ADASYN which places more emphasis on generating synthetic samples in areas where the classification is difficult(minority class, considering our target is extremely imbalanced)

Missing Value Imputation

Given the relevance of the cigsPerDay feature to currentSmoker, missing values in cigsPerDay will be imputed with 0, under the assumption that non-smokers would logically report smoking 0 cigarettes.

For other features with missing values and existing outliers, the median will be used for imputation. This approach helps mitigate the influence of extreme values and provides a robust estimate for central tendency.

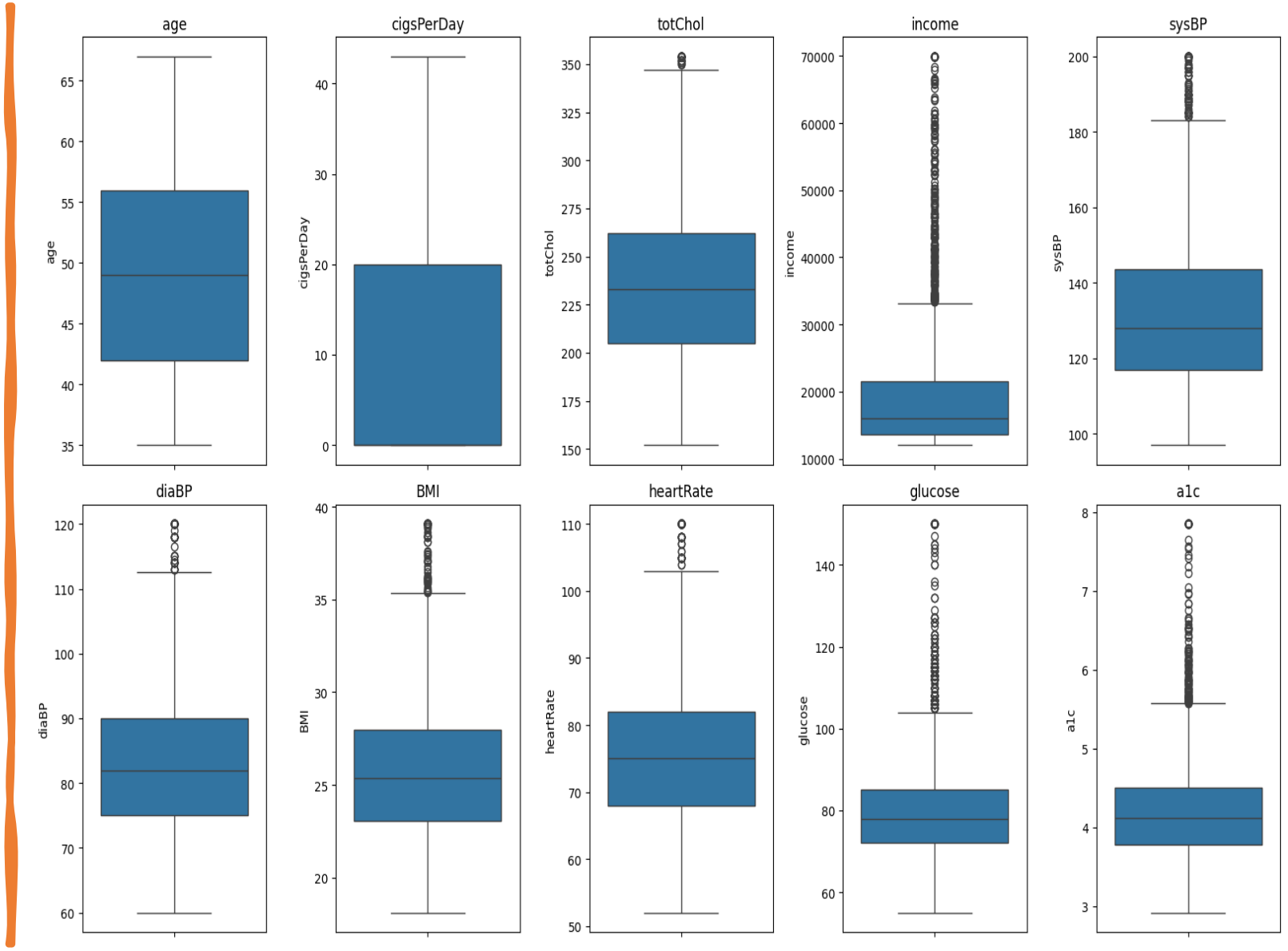
```
male                0
age                 0
education          72
currentSmoker       0
cigsPerDay       1549
BPMeds             39
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            38
sysBP               0
diaBP               0
BMI                15
heartRate           1
glucose           292
a1c               292
income              0
dtype: int64
```

Handling Skewed Data

- Skewed data can distort the model's understanding of the true underlying patterns, leading to biased predictions. To address this, the skewness of each numerical feature will be checked.
- Outliers will be managed using **winsorization**, limiting extreme values to reduce their influence on the distribution.

```
male              0.294230
age               0.247025
education         0.709639
currentSmoker     0.003934
cigsPerDay        1.244180
BPMeds            5.669338
prevalentStroke  12.561566
prevalentHyp      0.822237
diabetes          6.015813
totChol          37.508099
sysBP             1.197448
diaBP             0.746069
BMI               0.997801
heartRate         0.677382
glucose           6.881594
a1c               6.656094
income           13.904230
dtype: float64
```

Data after Winsorization



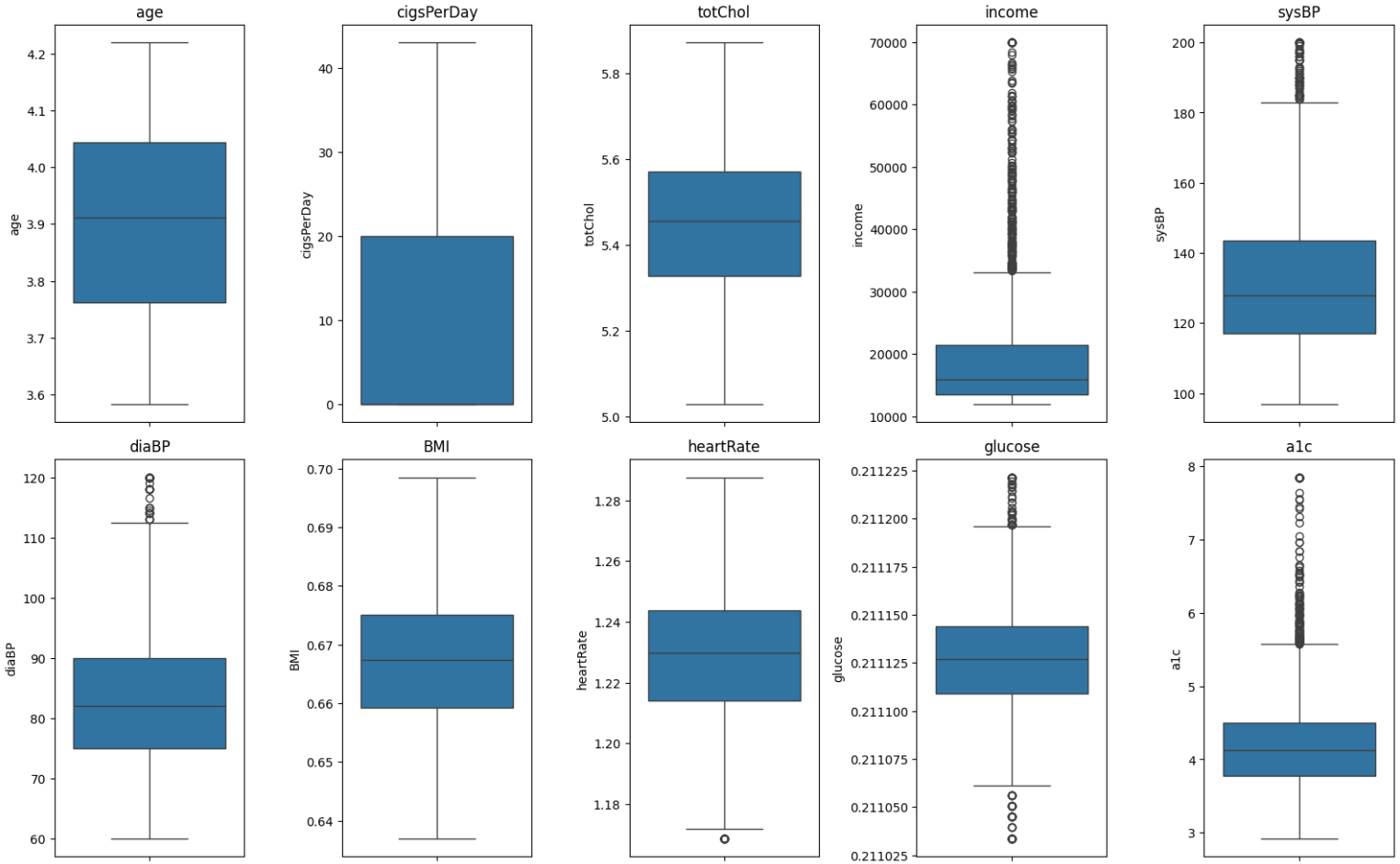
Transformations for Skewness Reduction

- Log transformation applied and for features that remain stubbornly skewed, such as BMI, heart rate, and glucose levels, a Box-Cox transformation will be considered. This transformation is more flexible and can handle different degrees and directions of skewness.

```
(age          0.010868
 BMI          0.277066
 heartRate    0.119826
 totChol     -0.049241
 glucose      0.920343
 dtype: float64,
 age         -0.047269
 BMI          0.216307
 heartRate    0.099227
 totChol     -0.087706
 glucose      1.066490
 dtype: float64)
```

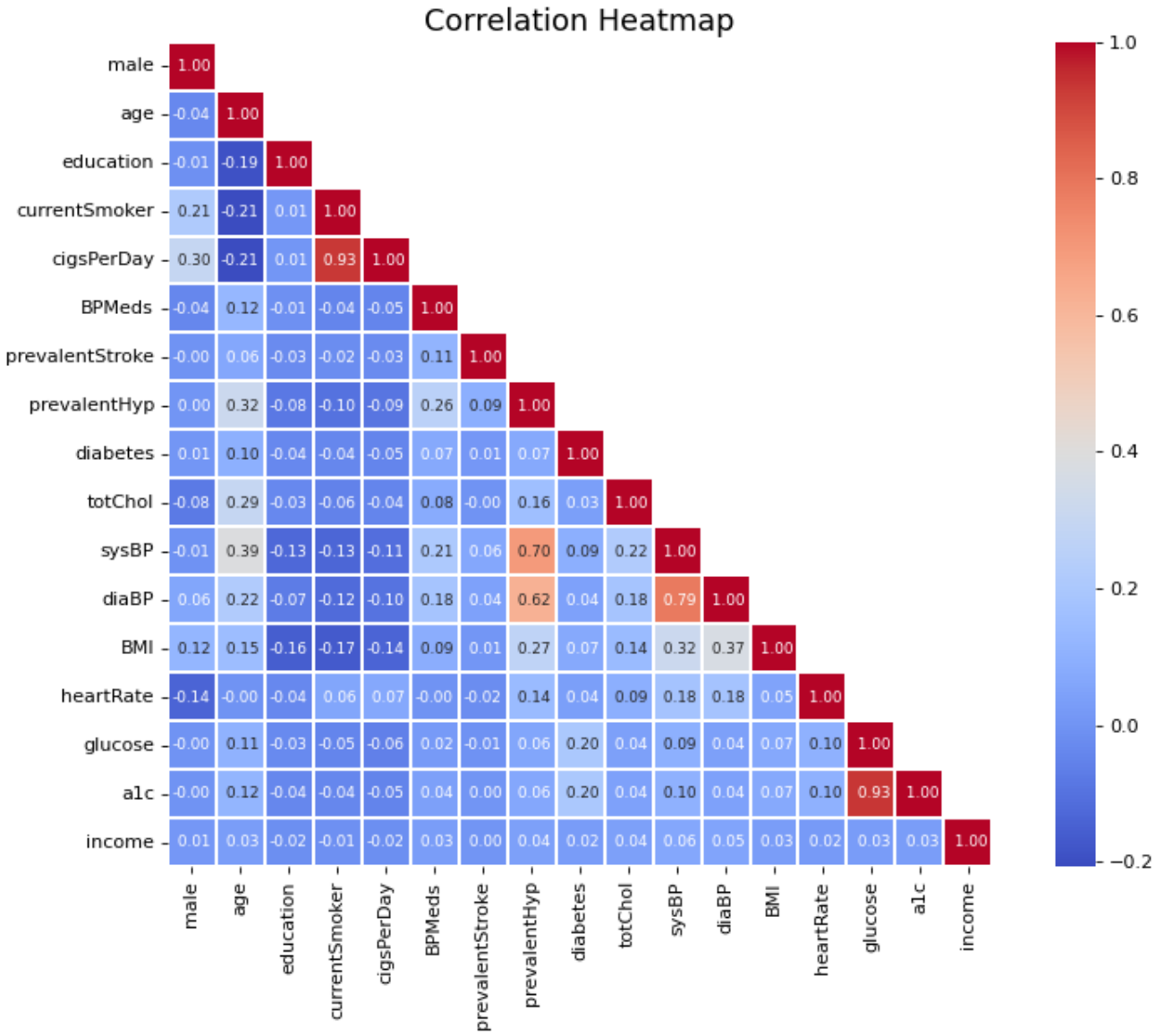
```
(age          0.010868
 BMI          0.000447
 heartRate    0.000547
 totChol     -0.049241
 glucose     -0.047288
 dtype: float64,
 age         -0.047269
 BMI         -0.058026
 heartRate   -0.017454
 totChol     -0.087706
 glucose      0.131245
 dtype: float64)
```

Numerical feature after transformatioin



Feature Heatmap

- Based on feature heatmap I have visualized the correlations between features.
- Preliminary analysis suggests strong correlations between sysBP, diaBP, and prevalentHyp; between glucose and a1c; and between cigsPerDay and currentSmoker.
- Understanding these relationships is critical to avoid multicollinearity in our predictive model.



Feature Reduction

- Based on the identified correlations, a1c will be dropped to avoid redundancy with glucose. The blood pressure-related features (sysBP, diaBP, prevalentHyp) will be combined into a single categorical feature bp_category.
- Similarly, currentSmoker and cigsPerDay will be merged into a smoking_status indicator, reducing data dimensionality and simplifying the model.

Categorize Smoking

```
X_train_comb = X_train_bin.copy()
X_test_comb = X_test_bin.copy()
def categorize_smoking(row):
    if row['currentSmoker'] == 0:
        return 'non-smoker'
    elif row['cigsPerDay'] <= 5:
        return 'light smoker'
    elif row['cigsPerDay'] <= 20:
        return 'moderate smoker'
    else:
        return 'heavy smoker'
X_train_comb['smoking_status'] = X_train_comb.apply(categorize_smoking, axis=1)
X_train_comb.drop(columns=['currentSmoker', 'cigsPerDay'], inplace=True)

X_test_comb['smoking_status'] = X_test_comb.apply(categorize_smoking, axis=1)
X_test_comb.drop(columns=['currentSmoker', 'cigsPerDay'], inplace=True)
```

Categorize BP

```
def create_bp_category(row):
    if row['prevalentHyp'] == 1:
        return 'Hypertension'
    else:
        if row['sysBP'] < 120 and row['diaBP'] < 80:
            return 'Normal'
        elif (row['sysBP'] >= 120 and row['sysBP'] < 140) or (row['diaBP'] >= 80 and row['diaBP'] < 90):
            return 'Prehypertension'
        elif row['sysBP'] >= 140 or row['diaBP'] >= 90:
            return 'Hypertension'

X_train_comb['bp_category'] = X_train_comb.apply(create_bp_category, axis=1)
X_train_comb = X_train_comb.drop(['sysBP', 'diaBP', 'prevalentHyp'], axis=1)

X_test_comb['bp_category'] = X_test_comb.apply(create_bp_category, axis=1)
X_test_comb = X_test_comb.drop(['sysBP', 'diaBP', 'prevalentHyp'], axis=1)
```

Scaling numerical data

- All numerical features will undergo scaling to ensure that they contribute equally to model training, preventing any single feature with larger scale from dominating the model's learning process.

	male	age	education	BPMeds	prevalentStroke	diabetes	totChol	BMI	heartRate	glucose	...
1078	0	-1.150215	1	0	0	0	-0.298757	-0.259511	0.451415	0.000758	...
1957	0	-0.731424	1	0	0	0	0.535625	0.152209	1.583443	-0.548769	...
2694	1	-0.598151	1	0	0	1	-0.607426	0.641839	1.519882	2.927215	...
2002	1	-0.598151	1	0	0	0	-1.948002	-1.308955	-0.703979	-1.743426	...
1213	0	1.107921	1	1	0	0	1.353996	0.671140	0.833897	1.781598	...
...	...	...	...	...	...	...	...	...	...	...	...
3129	0	0.259275	2	1	0	0	1.087950	1.401146	-1.227272	-0.449962	...
2287	1	1.490660	1	0	0	1	0.621015	1.100776	-0.703979	1.781598	...
2567	1	1.107921	1	0	0	0	1.409377	0.840441	-2.321087	1.137919	...
700	1	1.673149	1	0	0	0	-1.814769	1.932100	0.037355	0.000758	...
839	1	0.483090	4	0	0	0	-0.633943	0.304801	-0.228306	1.137919	...

764 rows × 21 columns

Logistic Regression Performance

Logistic Regression

```
[ ]  def logistic_reg(X_train, y_train, X_test, y_test):
         model = LogisticRegression(class_weight='balanced', max_iter=1000, )
         model.fit(X_train, y_train)

         # Predict
         y_pred = model.predict(X_test)
         y_prob = model.predict_proba(X_test)[:, 1]

         # Evaluating the model
         accuracy = accuracy_score(y_test, y_pred)
         conf_matrix = confusion_matrix(y_test, y_pred)
         classif_report = classification_report(y_test, y_pred)
         auc_score = roc_auc_score(y_test, y_prob)

         print("Accuracy:", accuracy)
         print("Confusion Matrix:\n", conf_matrix)
         print("Classification Report:\n", classif_report)
         print("AUC (Area Under Curve):", auc_score)

     logistic_reg(X_train_adasyn, y_train_adasyn, X_test_scaled, y_test)
```

```
Accuracy: 0.7801047120418848
Confusion Matrix:
 [[567  75]
 [ 93  29]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.88      0.87       642
           1       0.28      0.24      0.26       122

    accuracy                           0.78       764
   macro avg       0.57      0.56      0.56       764
weighted avg       0.77      0.78      0.77       764

AUC (Area Under Curve): 0.6844134620295184
```

GaussianNB

Surprisingly good at predicting minority clas

GaussianNB

doesn't fit for imbalanced data since it assume all features are uniformly distributed

```python
def gaussian_nb(X_train, y_train, X_test, y_test):
  model = GaussianNB()
  model.fit(X_train, y_train)

  #Predict
  y_pred = model.predict(X_test)
  y_prob = model.predict_proba(X_test)[:, 1]

  # Evaluating the model
  accuracy = accuracy_score(y_test, y_pred)
  conf_matrix = confusion_matrix(y_test, y_pred)
  classif_report = classification_report(y_test, y_pred)
  auc_score = roc_auc_score(y_test, y_prob)

  print("Accuracy:", accuracy)
  print("Confusion Matrix:\n", conf_matrix)
  print("Classification Report:\n", classif_report)
  print("AUC (Area Under Curve):", auc_score)

gaussian_nb(X_train_adasyn, y_train_adasyn, X_test_scaled, y_test)
```

```
Accuracy: 0.6387434554973822
Confusion Matrix:
 [[406 236]
 [ 40  82]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.63      0.75       642
           1       0.26      0.67      0.37       122

    accuracy                           0.64       764
   macro avg       0.58      0.65      0.56       764
weighted avg       0.81      0.64      0.69       764

AUC (Area Under Curve): 0.6994535519125683
```

SVM

Support Vector Machine

slowest

```
[ ]  def support_vector_machine(X_train, y_train, X_test, y_test):
         model = SVC(kernel='rbf', class_weight='balanced', C=1.5, probability=True, random_state=42)
         model.fit(X_train, y_train)

         # Predict
         y_pred = model.predict(X_test)
         y_prob = model.predict_proba(X_test)[:, 1]

         # Evaluating the model
         accuracy = accuracy_score(y_test, y_pred)
         conf_matrix = confusion_matrix(y_test, y_pred)
         classif_report = classification_report(y_test, y_pred)
         auc_score = roc_auc_score(y_test, y_prob)

         print("Accuracy:", accuracy)
         print("Confusion Matrix:\n", conf_matrix)
         print("Classification Report:\n", classif_report)
         print("AUC (Area Under Curve):", auc_score)

     support_vector_machine(X_train_adasyn, y_train_adasyn, X_test_scaled, y_test)
```

```
Accuracy: 0.7945026178010471
Confusion Matrix:
 [[570  72]
 [ 85  37]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       642
           1       0.34      0.30      0.32       122

    accuracy                           0.79       764
   macro avg       0.60      0.60      0.60       764
weighted avg       0.79      0.79      0.79       764

AUC (Area Under Curve): 0.6175310249731882
```

Random Forest

Random Forest

```
[ ]  def random_forest(X_train, y_train, X_test, y_test):
         # Creating and training the RandomForest classifier
         model = RandomForestClassifier(n_estimators=100, random_state=42, cl
         model.fit(X_train, y_train)

         y_pred = model.predict(X_test)
         y_prob = model.predict_proba(X_test)[:, 1]

         # Evaluating the model
         accuracy = accuracy_score(y_test, y_pred)
         conf_matrix = confusion_matrix(y_test, y_pred)
         classif_report = classification_report(y_test, y_pred)
         auc_score = roc_auc_score(y_test, y_prob)

         print("Accuracy:", accuracy)
         print("Confusion Matrix:\n", conf_matrix)
         print("Classification Report:\n", classif_report)
         print("AUC (Area Under Curve):", auc_score)

     random_forest(X_train_adasyn, y_train_adasyn, X_test_scaled, y_test)
```

```
Accuracy: 0.7997382198952879
Confusion Matrix:
 [[596  46]
 [107  15]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       642
           1       0.25      0.12      0.16       122

    accuracy                           0.80       764
   macro avg       0.55      0.53      0.53       764
weighted avg       0.75      0.80      0.77       764

AUC (Area Under Curve): 0.6554440529084317
```

LightGBM

- LightGBM achieves high accuracy in predictive majority class due to its advanced algorithm, I choose LightGBM instead of XGBoost because it uses lower memory and handles imbalanced data well.

```
  # Train the model
  clf.fit(X_train, y_train)

  # Predict on the test set
  y_pred = clf.predict(X_test)
  y_prob = clf.predict_proba(X_test)[:, 1]

  # Evaluating the model
  accuracy = accuracy_score(y_test, y_pred)
  conf_matrix = confusion_matrix(y_test, y_pred)
  classif_report = classification_report(y_test, y_pred)
  auc_score = roc_auc_score(y_test, y_prob)

  print("Accuracy:", accuracy)
  print("Confusion Matrix:\n", conf_matrix)
  print("Classification Report:\n", classif_report)
  print("AUC (Area Under Curve):", auc_score)

lightgbm(X_train_adasyn, y_train_adasyn, X_test_scaled, y_test)
```

```
[LightGBM] [Info] Number of positive: 463, number of negative: 2589
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001179 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1485
[LightGBM] [Info] Number of data points in the train set: 3052, number of used features: 17
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.151704 -> initscore=-1.721300
[LightGBM] [Info] Start training from score -1.721300
Accuracy: 0.8324607329842932
Confusion Matrix:
 [[622  26]
 [102  14]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.96      0.91       648
           1       0.35      0.12      0.18       116

    accuracy                           0.83       764
   macro avg       0.60      0.54      0.54       764
weighted avg       0.78      0.83      0.80       764

AUC (Area Under Curve): 0.6368534482758621
```

ADASYN

- Considered using ADASYN instead of SMOTE since we have lots of categorical feature
- But I discard this step in my final pipeline. Since although oversampling contributes to predicting minor class but synthetic data creates noise for majority data as well.

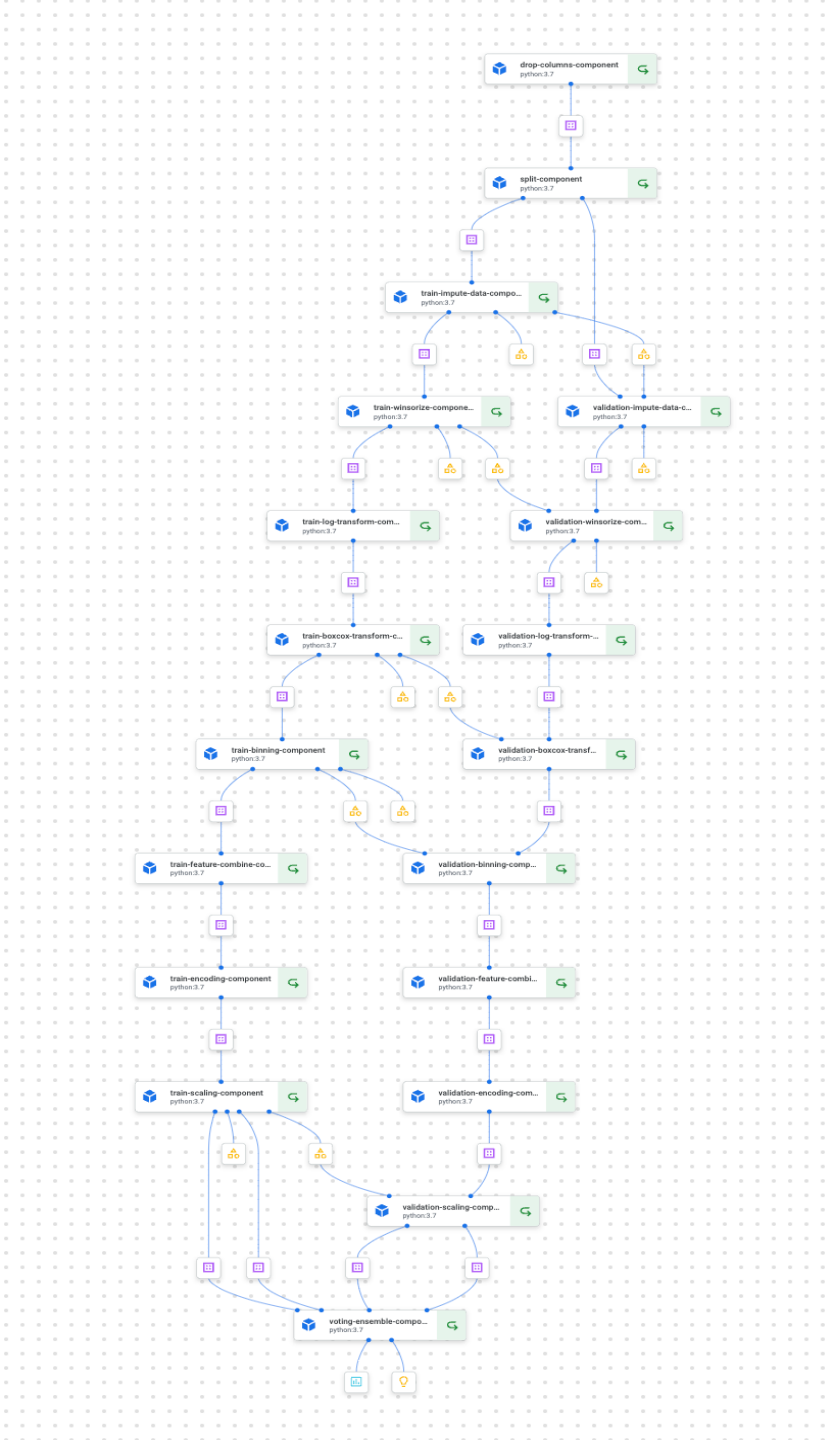
```
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train_scaled, y_train)

class_distribution = pd.Series(y_train_adasyn).value_counts()
class_distribution
```

```
TenYearCHD
1    2679
0    2595
Name: count, dtype: int64
```

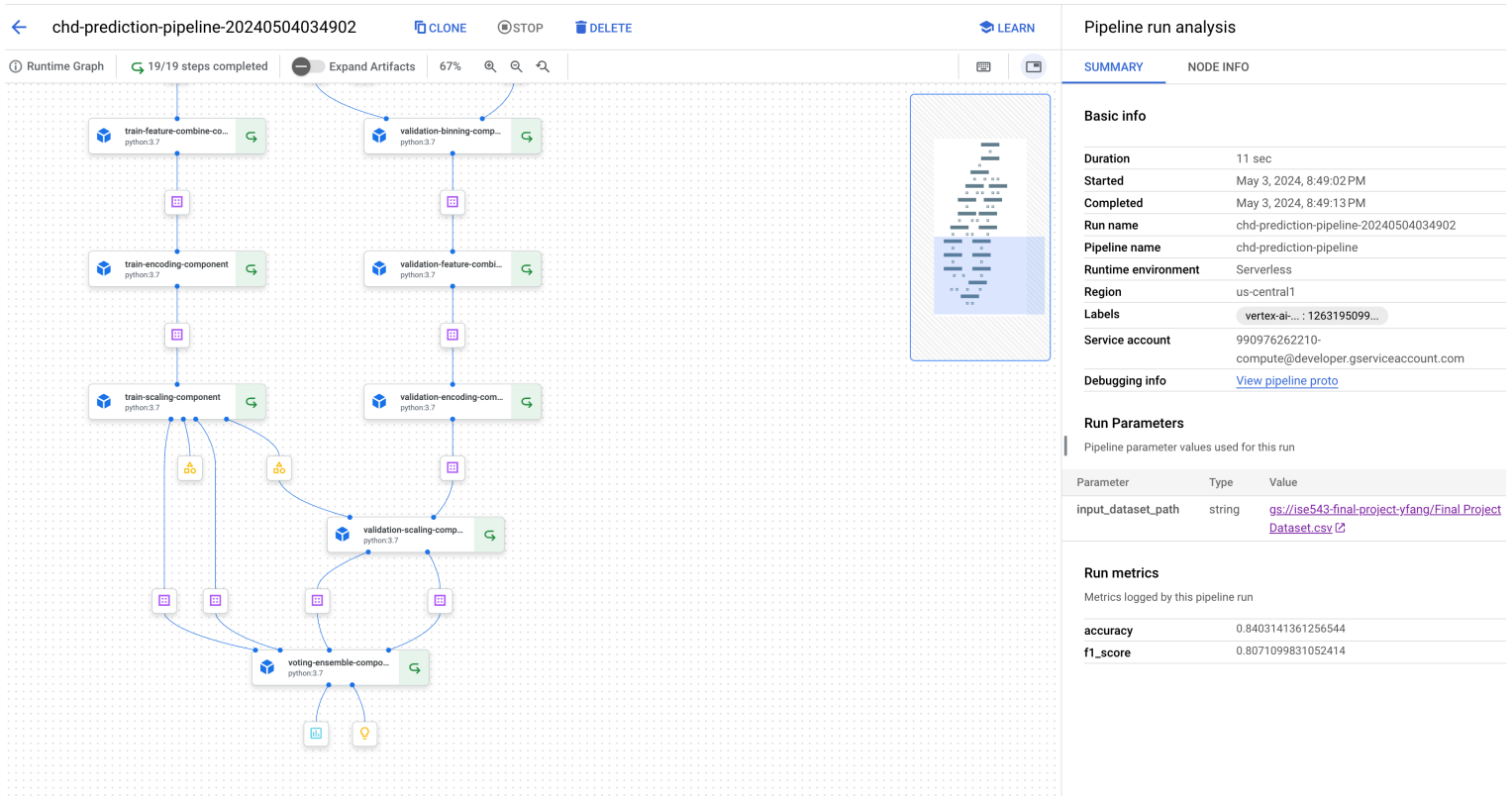
```
categorical_columns = X_train_adasyn.columns[X_train_adasyn.dtypes == 'object'].tolist()
categorical_columns.extend(['education', 'BPMeds', 'prevalentStroke', 'diabetes'])

categorical_imbalances = {col: X_train_adasyn[col].value_counts(normalize=True) for col in categorical_colu
categorical_imbalances
```

```
{'education': education
 1    0.495449
 2    0.260144
 3    0.146758
 4    0.097649
 Name: proportion, dtype: float64,
 'BPMeds': BPMeds
 0    0.979522
 1    0.020478
 Name: proportion, dtype: float64,
 'prevalentStroke': prevalentStroke
 0    0.996018
 1    0.003982
 Name: proportion, dtype: float64,
 'diabetes': diabetes
 0    0.97592
 1    0.02408
 Name: proportion, dtype: float64}
```

Model Pipeline



Validation result(f1 score:0.807 accuracy: 0.84)



Drop A1C, PaiteintID

[14] 0s

```
@component(packages_to_install=["pandas", "gcsfs", "fsspec"])
def drop_columns_component(
    input_dataset_path: str,
    output_dataset_path: OutputPath('Dataset'),
    columns_to_drop: list
):
    import pandas as pd

    # Load the dataset
    df = pd.read_csv(input_dataset_path)

    # Drop the specified columns
    df = df.drop(columns=columns_to_drop)

    # Save the dataset
    df.to_csv(output_dataset_path, index=False)
```

```
/usr/local/lib/python3.10/dist-packages/kfp/dsl/component_decorator.py:119: FutureWarning: Python 3.7 has reached end-of-life.
  return component_factory.create_component_from_func(
```

Split Data

```
[15] @component(packages_to_install=["pandas", "scikit-learn", "gcsfs", "fsspec"])
     def split_component(
         input_dataset_path: InputPath('Dataset'),
         train_dataset_path: OutputPath('Dataset'),
         validation_dataset_path: OutputPath('Dataset'),
         test_size: float = 0.1,
         random_state: int = 42):
         import pandas as pd
         from sklearn.model_selection import train_test_split

         # Load the dataset
         df = pd.read_csv(input_dataset_path)

         # Split into train+validation and test
         train_df, val_df = train_test_split(df, test_size=test_size, random_state=random_state)

         # Save the split datasets
         train_df.to_csv(train_dataset_path, index=False)
         val_df.to_csv(validation_dataset_path, index=False)
```

Impute train

```python
@component(packages_to_install=["pandas", "scikit-learn", "numpy", "gcsfs", "fsspec"])
def train_impute_data_component(
    train_dataset_path: InputPath('Dataset'),
    imputed_train_dataset_path: OutputPath('Dataset'),
    imputation_params_path: OutputPath(),
    imputation_metadata: Output[Artifact]  # Output for metadata
):
    import pandas as pd
    import numpy as np
    import json
    from sklearn.impute import SimpleImputer
    train_df = pd.read_csv(train_dataset_path)
    # Handling 'cigsPerDay' custom imputation
    train_df.loc[train_df['currentSmoker'] == 0, 'cigsPerDay'] = train_df.loc[train_df['currentSmoker'] == 0, 'cigsPerDay'].fillna(0)
    median_cigs = train_df[train_df['currentSmoker'] == 1]['cigsPerDay'].median()
    train_df.loc[train_df['currentSmoker'] == 1, 'cigsPerDay'] = train_df.loc[train_df['currentSmoker'] == 1, 'cigsPerDay'].fillna(median_cigs)
    # General median and mode imputation
    median_imputer = SimpleImputer(strategy='median')
    mode_imputer = SimpleImputer(strategy='most_frequent')
    median_cols = ['glucose', 'totChol', 'BMI', 'heartRate']
    mode_cols = ['education', 'BPMeds']
    train_df[median_cols] = median_imputer.fit_transform(train_df[median_cols])
    train_df[mode_cols] = mode_imputer.fit_transform(train_df[mode_cols])

    # Convert 'BPMeds' and 'Education' to integer
    train_df['BPMeds'] = train_df['BPMeds'].astype(int)
    train_df['education'] = train_df['education'].astype(int)

    imputation_params = {
        'median_cigs': median_cigs,
        'median_values': median_imputer.statistics_.tolist(),
        'mode_values': mode_imputer.statistics_.tolist()
    }
    with open(imputation_params_path, 'w') as f:
        json.dump(imputation_params, f)
```

Impute Validation

```
[17] @component(packages_to_install=["pandas", "scikit-learn", "numpy", "gcsfs", "fsspec"])
     def validation_impute_data_component(
         validation_dataset_path: InputPath('Dataset'),
         imputation_params_path: InputPath(),
         imputed_validation_dataset_path: OutputPath('Dataset'),
         validation_imputation_metadata: Output[Artifact]  # Output for metadata
     ):
         import pandas as pd
         import json
         import numpy as np
         from sklearn.impute import SimpleImputer

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

         # Load imputation parameters from the JSON file
         with open(imputation_params_path, 'r') as f:
             imputation_params = json.load(f)

         # Applying custom imputation for 'cigsPerDay'
         validation_df.loc[validation_df['currentSmoker'] == 0, 'cigsPerDay'] = validation_df.loc[validation_df['currentSmoker'] == 0, 'cigsPerDay'].fillna(0)
         validation_df.loc[validation_df['currentSmoker'] == 1, 'cigsPerDay'] = validation_df.loc[validation_df['currentSmoker'] == 1, 'cigsPerDay'].fillna(imputation_params['m

         # Applying general median and mode imputation using parameters from the training dataset
         median_cols = ['glucose', 'totChol', 'BMI', 'heartRate']
         mode_cols = ['education', 'BPMeds']
         median_imputer = SimpleImputer(strategy='median')
         mode_imputer = SimpleImputer(strategy='most_frequent')
         median_imputer.statistics_ = np.array(imputation_params['median_values'])
         mode_imputer.statistics_ = np.array(imputation_params['mode_values'])
         validation_df[median_cols] = median_imputer.transform(validation_df[median_cols])
         validation_df[mode_cols] = mode_imputer.transform(validation_df[mode_cols])

         # Convert 'BPMeds' and 'Education' to integer
         validation_df['BPMeds'] = validation_df['BPMeds'].astype(int)
         validation_df['education'] = validation_df['education'].astype(int)

         # Save the imputed validation dataset
         validation_df.to_csv(imputed_validation_dataset_path, index=False)
```

Collapse 1 child cell under Impute Validation
(Press <Shift> to also collapse sibling sections)

Winsorize train

```
[24] @component(packages_to_install=["pandas", "numpy", "gcsfs", "fsspec"])
     def train_winsorize_component(
         train_dataset_path: InputPath('Dataset'),
         winsorized_train_dataset_path: OutputPath('Dataset'),
         winsorize_params_path: OutputPath(),
         winsorize_metadata: Output[Artifact]  # Output for metadata
     ):
         import pandas as pd
         import numpy as np
         import json

         train_df = pd.read_csv(train_dataset_path)

         numerical_columns = ['age', 'cigsPerDay', 'totChol', 'income', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']
         winsorize_thresholds = {}

         for col in numerical_columns:
             sorted_train_data = np.sort(train_df[col])
             lower_limit_index = int(0.01 * len(sorted_train_data))  # 1% lower limit
             upper_limit_index = int(0.99 * len(sorted_train_data)) - 1  # 99% upper limit
             winsorize_thresholds[col] = (float(sorted_train_data[lower_limit_index]), float(sorted_train_data[upper_limit_index]))

             # Apply winsorization to train dataset
             train_df[col] = np.where(train_df[col] < sorted_train_data[lower_limit_index], sorted_train_data[lower_limit_index], train_df[col])
             train_df[col] = np.where(train_df[col] > sorted_train_data[upper_limit_index], sorted_train_data[upper_limit_index], train_df[col])

         # Save winsorize parameters
         with open(winsorize_params_path, 'w') as f:
             json.dump(winsorize_thresholds, f)

         # Saving metadata
         winsorize_metadata.metadata = {col: list(thresholds) for col, thresholds in winsorize_thresholds.items()}

         train_df.to_csv(winsorized_train_dataset_path, index=False)
```

Winsorize validation

```
[25] @component(packages_to_install=["pandas", "numpy", "gcsfs", "fsspec"])
     def validation_winsorize_component(
         validation_dataset_path: InputPath('Dataset'),
         winsorize_params_path: InputPath(),
         winsorized_validation_dataset_path: OutputPath('Dataset'),
         validation_winsorize_metadata: Output[Artifact]  # Output for metadata
     ):
         import pandas as pd
         import numpy as np
         import json

         validation_df = pd.read_csv(validation_dataset_path)

         # Load winsorize parameters from the JSON file
         with open(winsorize_params_path, 'r') as f:
             winsorize_thresholds = json.load(f)

         numerical_columns = ['age', 'cigsPerDay', 'totChol', 'income', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']

         for col in numerical_columns:
             lower_limit, upper_limit = winsorize_thresholds[col]

             # Apply winsorization to validation dataset
             validation_df[col] = np.where(validation_df[col] < lower_limit, lower_limit, validation_df[col])
             validation_df[col] = np.where(validation_df[col] > upper_limit, upper_limit, validation_df[col])

         # Saving metadata
         validation_winsorize_metadata.metadata = {col: list(thresholds) for col, thresholds in winsorize_thresholds.items()}

         validation_df.to_csv(winsorized_validation_dataset_path, index=False)
```

Log Transformation Train

```
[26] @component(packages_to_install=["pandas", "numpy", "gcsfs", "fsspec"])
     def train_log_transform_component(
         train_dataset_path: InputPath('Dataset'),
         log_transformed_train_dataset_path: OutputPath('Dataset')
     ):
         import pandas as pd
         import numpy as np

         train_df = pd.read_csv(train_dataset_path)

         skewed_features = ['age', 'BMI', 'heartRate', 'totChol', 'glucose']
         for feature in skewed_features:
             train_df[feature] = np.log1p(train_df[feature])

         train_df.to_csv(log_transformed_train_dataset_path, index=False)
```

Log Transformation Validation

```
[27] @component(packages_to_install=["pandas", "numpy", "gcsfs", "fsspec"])
     def validation_log_transform_component(
         validation_dataset_path: InputPath('Dataset'),
         log_transformed_validation_dataset_path: OutputPath('Dataset')
     ):
         import pandas as pd
         import numpy as np

         validation_df = pd.read_csv(validation_dataset_path)

         skewed_features = ['age', 'BMI', 'heartRate', 'totChol', 'glucose']
         for feature in skewed_features:
             validation_df[feature] = np.log1p(validation_df[feature])

         validation_df.to_csv(log_transformed_validation_dataset_path, index=False)
```

Box-cox Transform Train

```
@component(packages_to_install=["pandas", "numpy", "scikit-learn", "scipy", "gcsfs", "fsspec"])
def train_boxcox_transform_component(
    (parameter) boxcox_transformed_train_dataset_path: Any
    boxcox_transformed_train_dataset_path: OutputPath('Dataset'),
    boxcox_transformers_path: OutputPath(),
    boxcox_metadata: Output[Artifact]
):
    import pandas as pd
    import numpy as np
    from scipy.stats import boxcox
    from sklearn.preprocessing import PowerTransformer
    import pickle

    train_df = pd.read_csv(train_dataset_path)

    skewed_features_second = ['BMI', 'heartRate', 'glucose']
    transformers = {feature: PowerTransformer(method='box-cox', standardize=False) for feature in skewed_features_second}
    for feature in skewed_features_second:
        train_df[feature] = transformers[feature].fit_transform(train_df[[feature]].values)

    with open(boxcox_transformers_path, 'wb') as f:
        pickle.dump(transformers, f)

    boxcox_metadata.metadata = {feature: float(transformer.lambdas_[0]) for feature, transformer in transformers.items()}

    train_df.to_csv(boxcox_transformed_train_dataset_path, index=False)
```

Box-cox Transform Validation

[33]
```
@component(packages_to_install=["pandas", "numpy", "scikit-learn", "scipy", "gcsfs", "fsspec"])
def validation_boxcox_transform_component(
    validation_dataset_path: InputPath('Dataset'),
    boxcox_transformers_path: InputPath(),
    boxcox_transformed_validation_dataset_path: OutputPath('Dataset')
):
    import pandas as pd
    import numpy as np
    from scipy.stats import boxcox
    from sklearn.preprocessing import PowerTransformer
    import pickle

    validation_df = pd.read_csv(validation_dataset_path)

    with open(boxcox_transformers_path, 'rb') as f:
        transformers = pickle.load(f)

    skewed_features_second = ['BMI', 'heartRate', 'glucose']
    for feature in skewed_features_second:
        validation_df[feature] = transformers[feature].transform(validation_df[[feature]].values)

    validation_df.to_csv(boxcox_transformed_validation_dataset_path, index=False)
```

Binning train

```
[37] @component(packages_to_install=["pandas", "numpy", "gcsfs", "fsspec"])
     def train_binning_component(
         train_dataset_path: InputPath('Dataset'),
         binned_train_dataset_path: OutputPath('Dataset'),
         bin_edges_path: OutputPath(),
         binning_metadata: Output[Artifact]
     ):
         import pandas as pd
         import numpy as np
         import json

         train_df = pd.read_csv(train_dataset_path)

         q = train_df['income'].quantile([0.25, 0.5, 0.75]).values
         train_df['income_category'] = pd.qcut(train_df['income'], q=[0, 0.25, 0.5, 0.75, 1], labels=['Low', 'Middle',
         train_df.drop(columns='income', inplace=True)

         with open(bin_edges_path, 'w') as f:
             json.dump({'q': q.tolist()}, f)

         binning_metadata.metadata = {'bin_edges': q.tolist()}

         train_df.to_csv(binned_train_dataset_path, index=False)
```

Binning validation

```
@component(packages_to_install=["pandas", "numpy", "gcsfs", "fsspec"])
def validation_binning_component(
    validation_dataset_path: InputPath('Dataset'),
    bin_edges_path: InputPath(),
    binned_validation_dataset_path: OutputPath('Dataset')
):
    import pandas as pd
    import numpy as np
    import json

    validation_df = pd.read_csv(validation_dataset_path)

    with open(bin_edges_path, 'r') as f:
        bin_edges = json.load(f)['q']

    validation_df['income_category'] = pd.cut(validation_df['income'], bins=[validation_df['income'].m
    validation_df.drop(columns='income', inplace=True)

    validation_df.to_csv(binned_validation_dataset_path, index=False)
```

Feature Combine Train

```
[36] @component(packages_to_install=["pandas", "gcsfs", "fsspec"])
     def train_feature_combine_component(
         train_dataset_path: InputPath('Dataset'),
         combined_train_dataset_path: OutputPath('Dataset')
     ):
         import pandas as pd

         train_df = pd.read_csv(train_dataset_path)

         def categorize_smoking(row):
             if row['currentSmoker'] == 0:
                 return 'non-smoker'
             elif row['cigsPerDay'] <= 5:
                 return 'light smoker'
             elif row['cigsPerDay'] <= 20:
                 return 'moderate smoker'
             else:
                 return 'heavy smoker'

         train_df['smoking_status'] = train_df.apply(categorize_smoking, axis=1)
         train_df.drop(columns=['currentSmoker', 'cigsPerDay'], inplace=True)

         def create_bp_category(row):
             if row['prevalentHyp'] == 1:
                 return 'Hypertension'
             else:
                 if row['sysBP'] < 120 and row['diaBP'] < 80:
                     return 'Normal'
                 elif (row['sysBP'] >= 120 and row['sysBP'] < 140) or (row['diaBP'] >= 80 and row['diaBP'] < 90):
                     return 'Prehypertension'
                 elif row['sysBP'] >= 140 or row['diaBP'] >= 90:
                     return 'Hypertension'

         train_df['bp_category'] = train_df.apply(create_bp_category, axis=1)
         train_df = train_df.drop(['sysBP', 'diaBP', 'prevalentHyp'], axis=1)

         train_df.to_csv(combined_train_dataset_path, index=False)
```

Feature Combine Validation

```
[39] @component(packages_to_install=["pandas", "gcsfs", "fsspec"])
     def validation_feature_combine_component(
         validation_dataset_path: InputPath('Dataset'),
         combined_validation_dataset_path: OutputPath('Dataset')
     ):
         import pandas as pd

         validation_df = pd.read_csv(validation_dataset_path)

         def categorize_smoking(row):
             if row['currentSmoker'] == 0:
                 return 'non-smoker'
             elif row['cigsPerDay'] <= 5:
                 return 'light smoker'
             elif row['cigsPerDay'] <= 20:
                 return 'moderate smoker'
             else:
                 return 'heavy smoker'

         validation_df['smoking_status'] = validation_df.apply(categorize_smoking, axis=1)
         validation_df.drop(columns=['currentSmoker', 'cigsPerDay'], inplace=True)

         def create_bp_category(row):
             if row['prevalentHyp'] == 1:
                 return 'Hypertension'
             else:
                 if row['sysBP'] < 120 and row['diaBP'] < 80:
                     return 'Normal'
                 elif (row['sysBP'] >= 120 and row['sysBP'] < 140) or (row['diaBP'] >= 80 and row['diaBP'] < 90):
                     return 'Prehypertension'
                 elif row['sysBP'] >= 140 or row['diaBP'] >= 90:
                     return 'Hypertension'

         validation_df['bp_category'] = validation_df.apply(create_bp_category, axis=1)
         validation_df = validation_df.drop(['sysBP', 'diaBP', 'prevalentHyp'], axis=1)

         validation_df.to_csv(combined_validation_dataset_path, index=False)
```

Encode train

```
[43] @component(packages_to_install=["pandas", "gcsfs", "fsspec"])
     def train_encoding_component(
         train_dataset_path: InputPath('Dataset'),
         encoded_train_dataset_path: OutputPath('Dataset')
     ):
         import pandas as pd

         train_df = pd.read_csv(train_dataset_path)
         train_df = pd.get_dummies(train_df, columns=['income_category', 'smoking_status', 'bp_category'])

         train_df.to_csv(encoded_train_dataset_path, index=False)
```

Encode validation

```
[44] @component(packages_to_install=["pandas", "gcsfs", "fsspec"])
     def validation_encoding_component(
         validation_dataset_path: InputPath('Dataset'),
         encoded_validation_dataset_path: OutputPath('Dataset')
     ):
         import pandas as pd

         validation_df = pd.read_csv(validation_dataset_path)
         validation_df = pd.get_dummies(validation_df, columns=['income_category', 'smoking_status', 'bp_category'])

         validation_df.to_csv(encoded_validation_dataset_path, index=False)
```

Scaling train

```
[63] @component(packages_to_install=["pandas", "scikit-learn", "gcsfs", "fsspec"])
     def train_scaling_component(
         train_dataset_path: InputPath('Dataset'),
         scaled_train_features_path: OutputPath('Dataset'),
         scaled_train_target_path: OutputPath('Dataset'),
         scaler_path: OutputPath(),
         scaling_metadata: Output[Artifact]
     ):
         import pandas as pd
         from sklearn.preprocessing import StandardScaler
         import pickle

         train_df = pd.read_csv(train_dataset_path)

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

         scaler = StandardScaler()
         numerical_features = ['age', 'totChol', 'BMI', 'heartRate', 'glucose']
         X_train[numerical_features] = scaler.fit_transform(X_train[numerical_features])

         with open(scaler_path, 'wb') as f:
             pickle.dump(scaler, f)

         scaling_metadata.metadata = {
             'mean': scaler.mean_.tolist(),
             'scale': scaler.scale_.tolist()
         }

         X_train.to_csv(scaled_train_features_path, index=False)
         y_train.to_csv(scaled_train_target_path, index=False)
```

Scaling validation

```
[62] @component(packages_to_install=["pandas", "scikit-learn", "gcsfs", "fsspec"])
     def validation_scaling_component(
         validation_dataset_path: InputPath('Dataset'),
         scaler_path: InputPath(),
         scaled_validation_features_path: OutputPath('Dataset'),
         scaled_validation_target_path: OutputPath('Dataset')
     ):
         import pandas as pd
         from sklearn.preprocessing import StandardScaler
         import pickle

         validation_df = pd.read_csv(validation_dataset_path)

         X_val = validation_df.drop('TenYearCHD', axis=1)
         y_val = validation_df['TenYearCHD']

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

         numerical_features = ['age', 'totChol', 'BMI', 'heartRate', 'glucose']
         X_val[numerical_features] = scaler.transform(X_val[numerical_features])

         X_val.to_csv(scaled_validation_features_path, index=False)
         y_val.to_csv(scaled_validation_target_path, index=False)
```

Voting ensemble method

```python
from kfp.v2.dsl import Metrics, Output, InputPath, OutputPath, Artifact
@component(packages_to_install=["pandas", "scikit-learn", "lightgbm", "imbalanced-learn==0.9.0", "gcsfs", "fsspec"])
def voting_ensemble_component(
    train_features_path: InputPath('Dataset'),
    train_target_path: InputPath('Dataset'),
    validation_features_path: InputPath('Dataset'),
    validation_target_path: InputPath('Dataset'),
    model_path: OutputPath('Model'),
    evaluation_metrics: Output[Metrics]):
    import pandas as pd
    from sklearn.ensemble import VotingClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.svm import SVC
    from imblearn.ensemble import BalancedRandomForestClassifier
    import lightgbm as lgb
    from sklearn.naive_bayes import GaussianNB
    from sklearn.metrics import accuracy_score, f1_score
    import pickle
    X_train = pd.read_csv(train_features_path)
    y_train = pd.read_csv(train_target_path)
    X_val = pd.read_csv(validation_features_path)
    y_val = pd.read_csv(validation_target_path)
    # Create instances of individual models
    lr = LogisticRegression(class_weight='balanced', max_iter=1000)
    svm = SVC(kernel='sigmoid', class_weight='balanced', probability=True, random_state=42)
    brf = BalancedRandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
    lgbm = lgb.LGBMClassifier(num_leaves=63, max_depth=-1, learning_rate=0.05, n_estimators=1000, scale_pos_weight=1, random_state=42)
    gn = GaussianNB()
    estimators = [
        ('lr', lr),
        ('svm', svm),
        ('brf', brf),
        ('lgbm', lgbm),
        ('gn', gn)
    ]
    model = VotingClassifier(estimators=estimators, voting='soft')
    model.fit(X_train, y_train)
    # Save the trained model
    with open(model_path, 'wb') as file:
        pickle.dump(model, file)
    # Predict and evaluate
    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred, average='weighted')
    evaluation_metrics.log_metric('accuracy', accuracy)
    evaluation_metrics.log_metric('f1_score', f1)
```

Initial Pipeline

```
[ ]  @pipeline(name="chd-prediction-pipeline")
     def chd_prediction_pipeline(input_dataset_path: str):
         # Run the components in order
         processed_dataset_path = drop_columns_component(input_dataset_path=input_dataset_path,
                                                         columns_to_drop=['patientID', 'a1c'])
         split_result = split_component(input_dataset_path=processed_dataset_path.output)

         train_impute_result = train_impute_data_component(
             train_dataset_path=split_result.outputs['train_dataset_path']
         )

         validation_impute_result = validation_impute_data_component(
             validation_dataset_path=split_result.outputs['validation_dataset_path'],
             imputation_params_path=train_impute_result.outputs['imputation_params_path']
         )

         train_winsorize_result = train_winsorize_component(
             train_dataset_path=train_impute_result.outputs['imputed_train_dataset_path']
         )

         validation_winsorize_result = validation_winsorize_component(
             validation_dataset_path=validation_impute_result.outputs['imputed_validation_dataset_path'],
             winsorize_params_path=train_winsorize_result.outputs['winsorize_params_path']
         )

         train_log_transform_result = train_log_transform_component(
             train_dataset_path=train_winsorize_result.outputs['winsorized_train_dataset_path']
         )

         validation_log_transform_result = validation_log_transform_component(
             validation_dataset_path=validation_winsorize_result.outputs['winsorized_validation_dataset_path']
         )
```

```
     )

     validation_binning_result = validation_binning_component(
         validation_dataset_path=validation_boxcox_transform_result.outputs['boxcox_transformed_validation_dataset_path'],
         bin_edges_path=train_binning_result.outputs['bin_edges_path']
     )

     train_feature_combine_result = train_feature_combine_component(
         train_dataset_path=train_binning_result.outputs['binned_train_dataset_path']
     )

     validation_feature_combine_result = validation_feature_combine_component(
         validation_dataset_path=validation_binning_result.outputs['binned_validation_dataset_path']
     )

     train_encoding_result = train_encoding_component(
         train_dataset_path=train_feature_combine_result.outputs['combined_train_dataset_path']
     )

     validation_encoding_result = validation_encoding_component(
         validation_dataset_path=validation_feature_combine_result.outputs['combined_validation_dataset_path']
     )

     train_scaling_result = train_scaling_component(
         train_dataset_path=train_encoding_result.outputs['encoded_train_dataset_path']
     )

     validation_scaling_result = validation_scaling_component(
         validation_dataset_path=validation_encoding_result.outputs['encoded_validation_dataset_path'],
         scaler_path=train_scaling_result.outputs['scaler_path']
     )

     voting_ensemble_result = voting_ensemble_component(
         train_features_path=train_scaling_result.outputs['scaled_train_features_path'],
         train_target_path=train_scaling_result.outputs['scaled_train_target_path'],
         validation_features_path=validation_scaling_result.outputs['scaled_validation_features_path'],
```

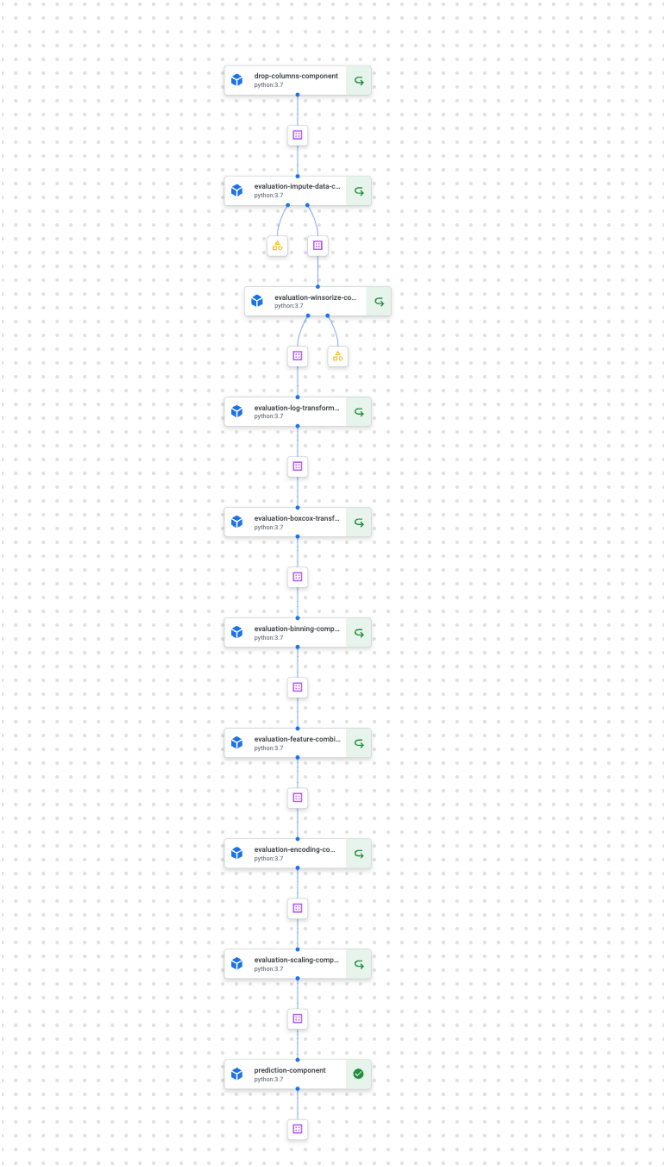
Compile and run

Compile and run Initial Pipeline

```
[113] from kfp.v2 import compiler

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

      pipeline_job = aiplatform.PipelineJob(
          display_name='chd-prediction-pipeline',
          template_path='chd_prediction_pipeline.json',
          pipeline_root='gs://ise543-final-project-yfang',
          enable_caching=True,
          parameter_values={
              'input_dataset_path': 'gs://ise543-final-project-yfang/Final Project Dataset.csv',
          }
      )

      pipeline_job.run()
```

```
INFO:google.cloud.aiplatform.pipeline_jobs:Creating PipelineJob
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob created. Resource name: projects/990976262210/l
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/9909762
INFO:google.cloud.aiplatform.pipeline_jobs:View Pipeline Job:
https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/chd-prediction-pipelin
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob projects/990976262210/locations/us-central1/pip
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob run completed. Resource name: projects/99097626
```

Inference Pipeline



Components are similar with validation components, all artifacts are imported from training set

Inference Pipeline

```
[138] imputation_params_path = 'gs://ise543-final-project-yfang/990976262210/chd-prediction-pipeline-20240503212758/train-impute-data-component_7933296970762813440/imputation_pa
      winsorize_params_path = 'gs://ise543-final-project-yfang/990976262210/chd-prediction-pipeline-20240503212758/train-winsorize-component_2168689447728578560/winsorize_params
      boxcox_transformers_path = 'gs://ise543-final-project-yfang/990976262210/chd-prediction-pipeline-20240503212758/train-boxcox-transform-component_3321610952335425536/boxco
      bin_edges_path = 'gs://ise543-final-project-yfang/990976262210/chd-prediction-pipeline-20240503212758/train-binning-component_-3595918075305656320/bin_edges_path'
      scaler_path = 'gs://ise543-final-project-yfang/990976262210/chd-prediction-pipeline-20240504010640/train-scaling-component_471676813140033536/scaler_path'
      model_path = 'gs://ise543-final-project-yfang/990976262210/chd-prediction-pipeline-20240504013540/voting-ensemble-component_-2551364436732411904/model_path'

      @pipeline(name="chd-prediction-inference-pipeline")
      def chd_prediction_inference_pipeline(
          evaluation_dataset_path: str,
          model_path: str = model_path,
          imputation_params_path: str = imputation_params_path,
          winsorize_params_path: str = winsorize_params_path,
          boxcox_transformers_path: str = boxcox_transformers_path,
          bin_edges_path: str = bin_edges_path,
          scaler_path: str = scaler_path
      ):
          # Preprocessing components for the evaluation dataset
          processed_evaluation_dataset_path = drop_columns_component(input_dataset_path=evaluation_dataset_path, columns_to_drop=['patientID', 'a1c'])

          evaluation_impute_result = evaluation_impute_data_component(
              evaluation_dataset_path=processed_evaluation_dataset_path.output,
              imputation_params_path=imputation_params_path
          )

          evaluation_winsorize_result = evaluation_winsorize_component(
              evaluation_dataset_path=evaluation_impute_result.outputs['imputed_evaluation_dataset_path'],
              winsorize_params_path=winsorize_params_path
          )

          evaluation_log_transform_result = evaluation_log_transform_component(
              evaluation_dataset_path=evaluation_winsorize_result.outputs['winsorized_evaluation_dataset_path']
          )

          evaluation_boxcox_transform_result = evaluation_boxcox_transform_component(
              evaluation_dataset_path=evaluation_log_transform_result.outputs['log_transformed_evaluation_dataset_path'],
              boxcox_transformers_path=boxcox_transformers_path
          )
```

Inference pipeline cont.

```
)

evaluation_log_transform_result = evaluation_log_transform_component(
    evaluation_dataset_path=evaluation_winsorize_result.outputs['winsorized_evaluation_dataset_path']
)

evaluation_boxcox_transform_result = evaluation_boxcox_transform_component(
    evaluation_dataset_path=evaluation_log_transform_result.outputs['log_transformed_evaluation_dataset_path'],
    boxcox_transformers_path=boxcox_transformers_path
)

evaluation_binning_result = evaluation_binning_component(
    evaluation_dataset_path=evaluation_boxcox_transform_result.outputs['boxcox_transformed_evaluation_dataset_path'],
    bin_edges_path=bin_edges_path
)

evaluation_feature_combine_result = evaluation_feature_combine_component(
    evaluation_dataset_path=evaluation_binning_result.outputs['binned_evaluation_dataset_path']
)

evaluation_encoding_result = evaluation_encoding_component(
    evaluation_dataset_path=evaluation_feature_combine_result.outputs['combined_evaluation_dataset_path']
)

evaluation_scaling_result = evaluation_scaling_component(
    evaluation_dataset_path=evaluation_encoding_result.outputs['encoded_evaluation_dataset_path'],
    scaler_path=scaler_path
)

# Prediction component
prediction_result = prediction_component(
    evaluation_dataset_path=evaluation_scaling_result.outputs['scaled_evaluation_features_path'],
    original_evaluation_dataset_path=evaluation_dataset_path,
    model_path=model_path
)
```

Prediction component

Prediction

```
[137] @component(packages_to_install=["pandas", "scikit-learn", "lightgbm", "imbalanced-learn==0.9.0", "gcsfs", "fsspec"])
      def prediction_component(
          evaluation_dataset_path: InputPath('Dataset'),
          original_evaluation_dataset_path: str,
          model_path: str,
          predictions_path: OutputPath('Dataset')
      ):
          import pandas as pd
          import pickle
          import gcsfs

          # Load the preprocessed evaluation dataset
          evaluation_data = pd.read_csv(evaluation_dataset_path)

          # Load the original evaluation dataset
          original_evaluation_data = pd.read_csv(original_evaluation_dataset_path)

          # Load the trained model
          fs = gcsfs.GCSFileSystem()
          with fs.open(model_path, 'rb') as file:
              model = pickle.load(file)

          # Make predictions on the evaluation dataset
          predictions = model.predict(evaluation_data)

          # Create a DataFrame with the patientID and predictions
          predictions_df = pd.DataFrame({
              'patientID': original_evaluation_data['patientID'],
              'TenYearCHD': predictions
          })

          # Save the predictions with patientID to a CSV file
          predictions_df.to_csv(predictions_path, index=False)
```

Compile and run

Compile and run Inference Pipeline

```python
from kfp.v2 import compiler

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

pipeline_job = aiplatform.PipelineJob(
    display_name='chd-prediction-inference-pipeline',
    template_path='chd_prediction_inference_pipeline.json',
    pipeline_root='gs://ise543-final-project-yfang',
    enable_caching=True,
    parameter_values={
        'evaluation_dataset_path': 'gs://ise543-final-project-yfang/Final-Project-Evaluation-Dataset.csv'
    }
)

pipeline_job.run()
```

```
INFO:google.cloud.aiplatform.pipeline_jobs:Creating PipelineJob
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob created. Resource name: projects/990976262210/locations/us-central1/pipelineJobs/chd-predicti
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/990976262210/locations/us-central1/pipelineJobs/chd-p
INFO:google.cloud.aiplatform.pipeline_jobs:View Pipeline Job:
https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/chd-prediction-inference-pipeline-20240504050147?project=99097626221
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob projects/990976262210/locations/us-central1/pipelineJobs/chd-prediction-inference-pipeline-20
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob projects/990976262210/locations/us-central1/pipelineJobs/chd-prediction-inference-pipeline-20
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob projects/990976262210/locations/us-central1/pipelineJobs/chd-prediction-inference-pipeline-20
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob projects/990976262210/locations/us-central1/pipelineJobs/chd-prediction-inference-pipeline-20
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.pipeline_jobs:PipelineJob run completed. Resource name: projects/990976262210/locations/us-central1/pipelineJobs/chd-pr
```

Predicted Dataset

990976262210_chd-

patientID	TenYearCHD
110399	0
189047	0
957019	0
208967	0
230935	0
216024	0
368834	0
135175	0
294070	0
595710	0
425597	0
650137	0
590019	0
925626	0
276518	0
342284	0
469306	0
197764	0
416488	0
208652	0
562216	0
115448	0
224178	0
271781	0
887721	1
338781	0
262782	0
583133	0
196173	0
779064	0
676839	0
141292	0
881246	0
982834	0

Final Score

Autograder Results

Check submitted files (0/0)

```
Required CSV file submitted.
```

1) F1-Score (80.58/100)

```
Test Failed: 80.58 != 100 : The F1-Score is 0.8057822515121422.
```

Summary

- In this project, we developed a machine learning model to predict the risk of coronary heart disease (CHD) in patients based on various health-related features. The goal was to build a robust and accurate model that can assist in early detection and prevention of CHD.
- We performed extensive data preprocessing to ensure the quality and suitability of the data for modeling. The preprocessing steps included:
- Dropping unnecessary columns (patientID and a1c)
- Splitting the data into training and validation sets
- Imputing missing values using median and mode imputation
- Winsorizing outliers to handle extreme values
- Applying log transformation and Box-Cox transformation to handle skewed features
- Binning the income feature into categories
- Combining smoking and blood pressure features into new categorical features
- Encoding categorical variables using one-hot encoding
- Scaling numerical features using standardization

Summary

- Model Selection:
  - After preprocessing the data, we explored various machine learning algorithms to find the best-performing model. We chose the Voting Ensemble method, which combines multiple diverse models to make predictions. The ensemble consisted of the following models:
  - Logistic Regression, Support Vector Machine (SVM), Balanced Random Forest, LightGBM, Gaussian Naive Bayes
- During the modeling process, we observed class imbalance in the dataset, with the minority class (patients with CHD) being underrepresented. We considered sampling techniques to balance the classes but decided against it due to the trade-off between minority class accuracy and overall F1 score. Instead, we used balanced class weights, selected models like Gaussian Naive Bayes that handle imbalance well, and evaluated performance using appropriate metrics. The Gaussian Naive Bayes classifier performed particularly well in predicting the minority class, contributing to the ensemble's overall performance.
- The Voting Ensemble method was selected because it leverages the strengths of different algorithms and provides a robust and accurate prediction by aggregating their individual predictions.
- Lastly we evaluated the performance of the Voting Ensemble model using appropriate metrics such as accuracy, F1 score, and performance on the minority class. The model achieved satisfactory results, indicating its effectiveness in predicting CHD risk.

Summary

- To operationalize the trained model, we developed an inference pipeline that automates the process of making predictions on new, unseen data. The inference pipeline includes the following steps:
- Data input: Receiving new patient data for prediction
- Data preprocessing: Applying the same preprocessing steps used during model training
- Model loading: Loading the trained Voting Ensemble model
- Prediction: Feeding the preprocessed data into the model to generate CHD risk predictions
- Post-processing: Adding the patientID back to the predictions for identification
- The inference pipeline ensures that the model can be easily deployed and used for real-world predictions, enabling healthcare professionals to leverage the model's insights in patient care and decision-making.
- In conclusion, this project demonstrates the development of a machine learning model for predicting coronary heart disease risk. Through extensive data preprocessing, careful model selection, and the use of a Voting Ensemble method, we achieved a robust and accurate model. The Gaussian Naive Bayes classifier's performance on the minority class highlights its importance in handling class imbalance. The inference pipeline enables the model to be deployed and utilized in real-world scenarios, potentially aiding in early detection and prevention of CHD.