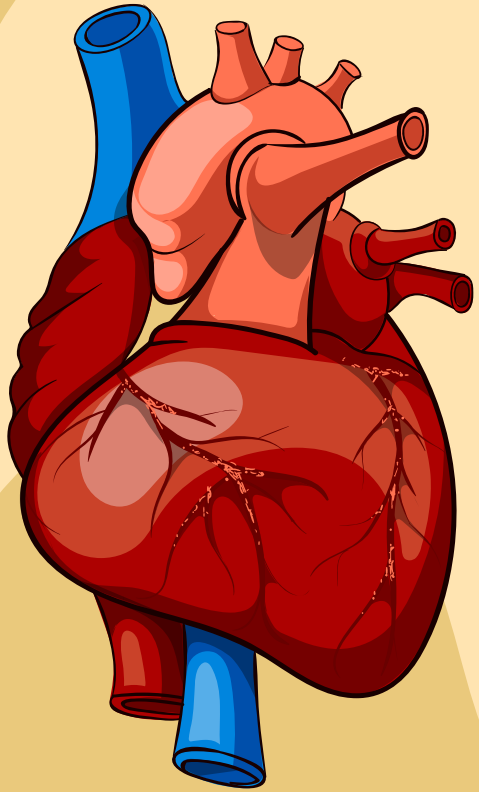
ISE 543 – Enterprise Business Intelligence and System Analytics

FINAL PROJECT – SPRING 2024

NAME -SHREE VAISHNAVI BACHA
USC ID - 6230407082



CHD PREDICTION USING FRAMINGHAM HEART STUDY

FINAL PROJECT REPORT

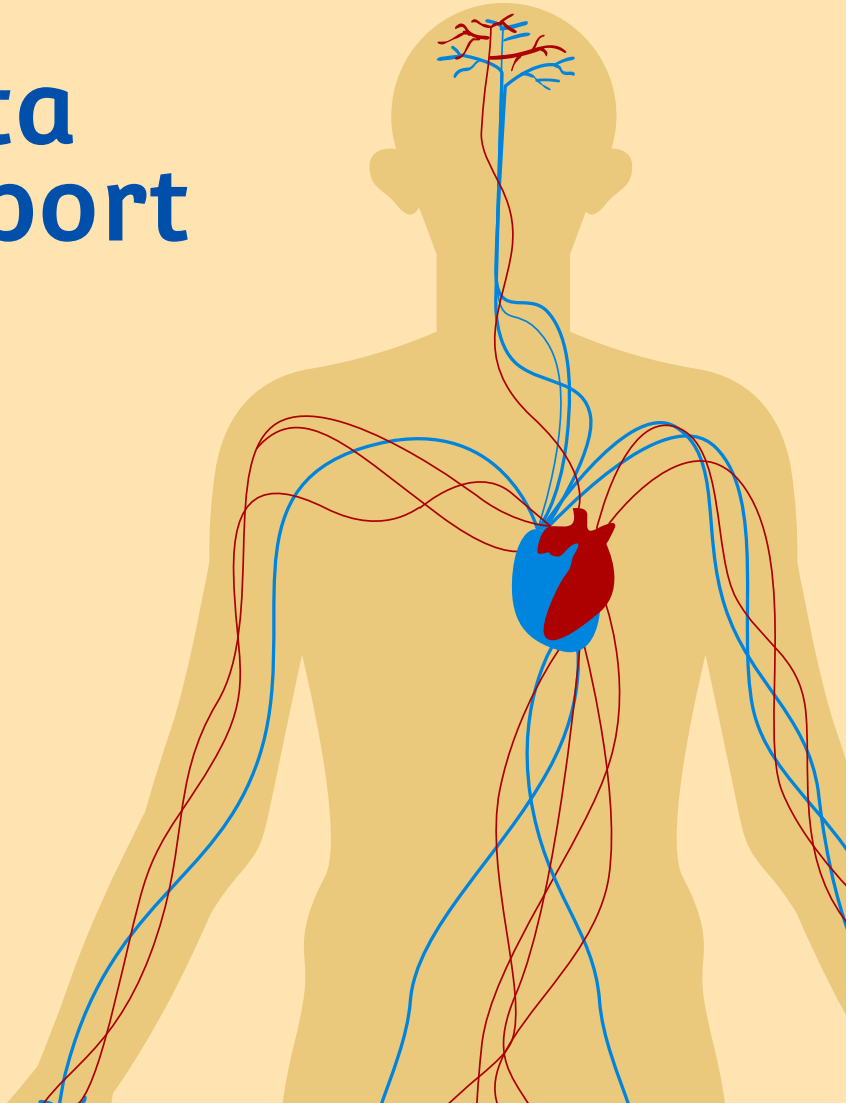
Framingham Heart Study

Heart disease is the leading cause of death worldwide. About 17.9 million people died from coronary heart disease (CHD) in 2016 – over 25% of all deaths that year across the globe.

In 1928, the U.S government started to track a cohort of people in Framingham, MA. The study comprised initially 5,209 participants, who were given a questionnaire and a medical exam every 2 years. Data were also collected on the physical and behavioral characteristics. Over the years, the study has grown to include multiple generations and more variables. The dataset is known as Framingham Heart Study.

Exploratory Data Analysis (EDA) Report

- Business Objective
- Dataset Summary
- Data Quality Summary
- Univariate Analysis
- Bivariate Analysis



Business Objective

Develop a model to predict which patients will experience CHD (Coronary Heart Disease) within 10 years of their first examination. We want to build a classification model to predict whether a patient will experience CHD within 10 years of their first examination using patients demographic and clinical characteristics given in the Framingham Heart Study Dataset.

Business Users: The model can be used by the Physicians to proactively identify patients who are at high-risk of developing CHD and intervene with preventive measures such as medication prescriptions and lifestyle modifications. This approach aligns well with improving patient outcomes and reducing healthcare costs associated with treating advanced stages of the disease.

Dataset Summary

- This version of the Framingham Heart Study dataset has 19 variables (columns) and 3816 observations (rows).

- The response variable is TenYearCHD which a binary categorical variable.
  - 1 if patient has experienced CHD within 10 years of first examination,
  - 0 otherwise

- Unit of Analysis/Observation: One row represents a unique ID that is a unique patient.
- Unique Identifier: The patientID is the unique identifier.

Data Dictionary

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)

Dataset Summary- Variables

Categorical Variables

Predictors:
- Male
- Education
- Current Smoker
- BPMeds
- Prevalent Stroke
- Prevalent Hypertension
- Diabetes

Response:
- TenYearCHD

Numerical Variables

Predictors:
- Age
- Income
- Cigarettes per day
- TotChol
- Sys BP
- Dia BP
- BMI
- Heart Rate
- Glucose
- A1c

Unique Identifier (Numeric):
- PatientID

Data Quality Summary
Missing Values

Exploratory Data Analysis on the Train Dataset

Missing Values

```
CHD_train.isnull().sum()
```

```
patientID            0
male                 0
age                  0
education           69
currentSmoker        0
cigsPerDay        1377
BPMeds              28
prevalentStroke      0
prevalentHyp         0
diabetes             0
totChol             30
sysBP                0
diaBP                0
BMI                 14
heartRate            0
glucose            255
a1c                255
income               0
TenYearCHD           0
dtype: int64
```

The following Variables have missing values:

- education
- cigsPerDay
- BPMeds
- totChol
- BMI
- heartrate
- glucose
- a1c

Data Quality Summary

Data Types

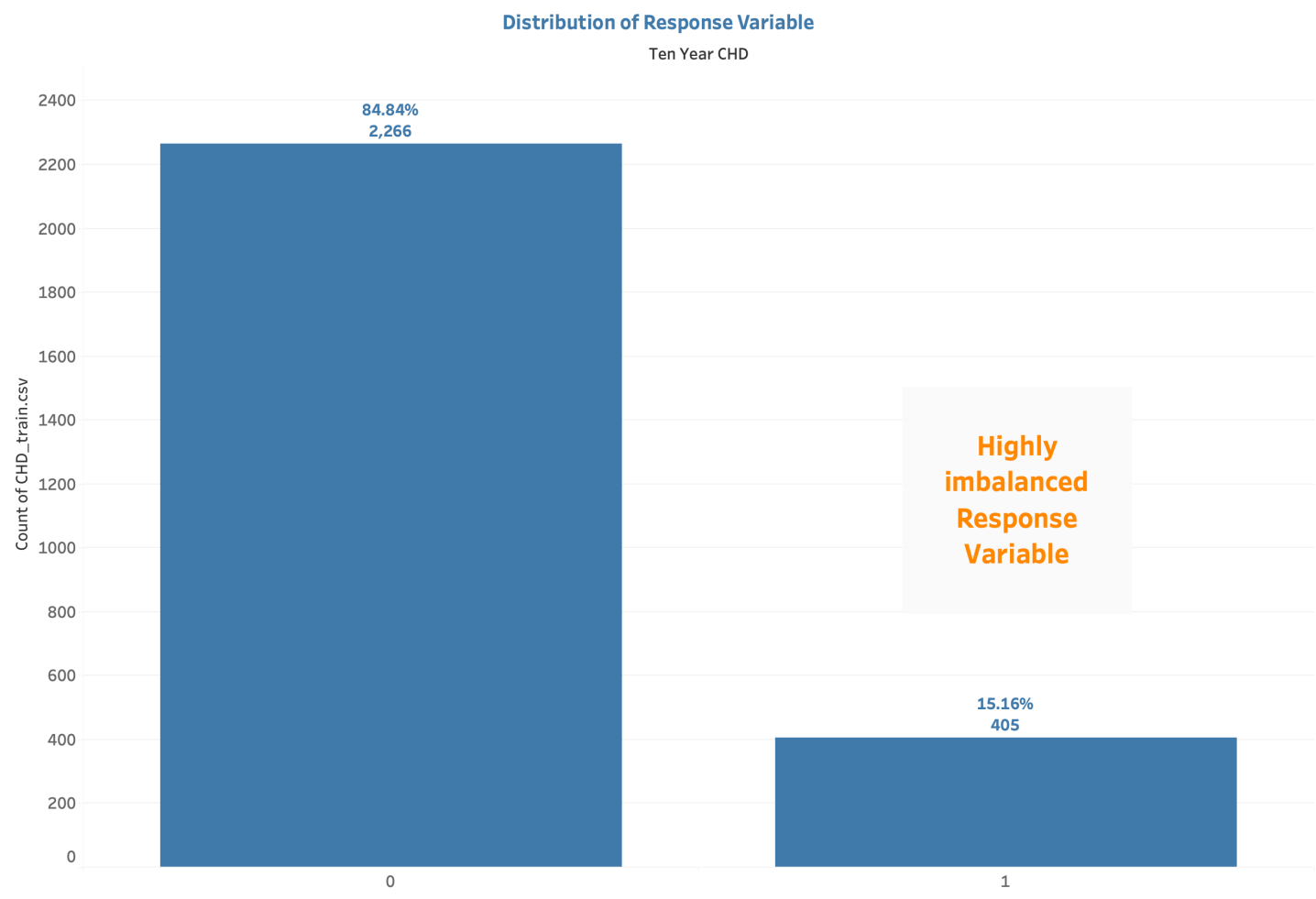
```
0s
CHD_train.dtypes

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
a1c                float64
income             float64
TenYearCHD           int64
dtype: object
```

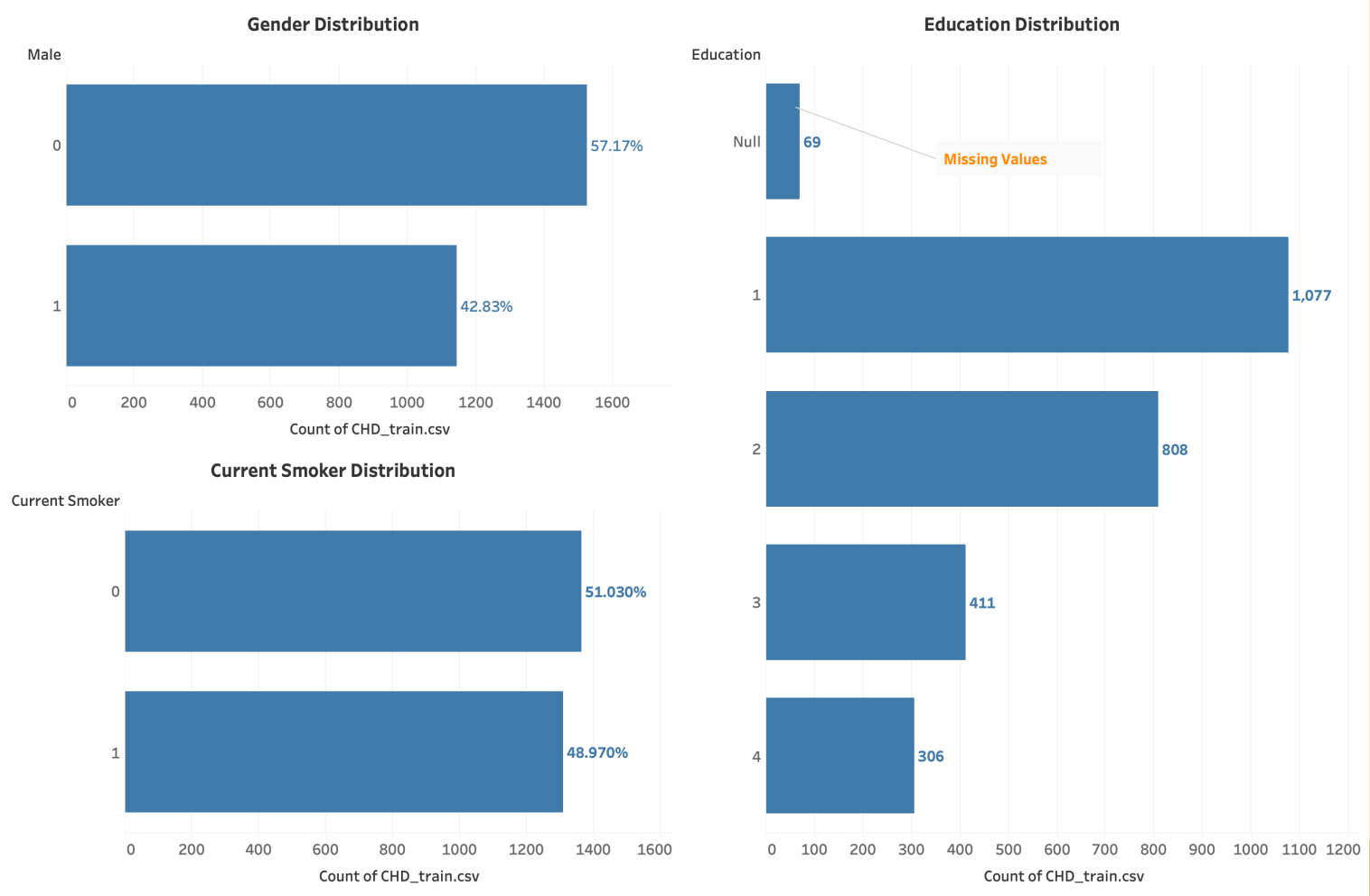
Categories that must be encoded before modelling:

- Male
- Education
- Current Smoker
- BPMeds
- Prevalent Stroke
- Prevalent Hypertension
- Diabetes
- TenYearCHD

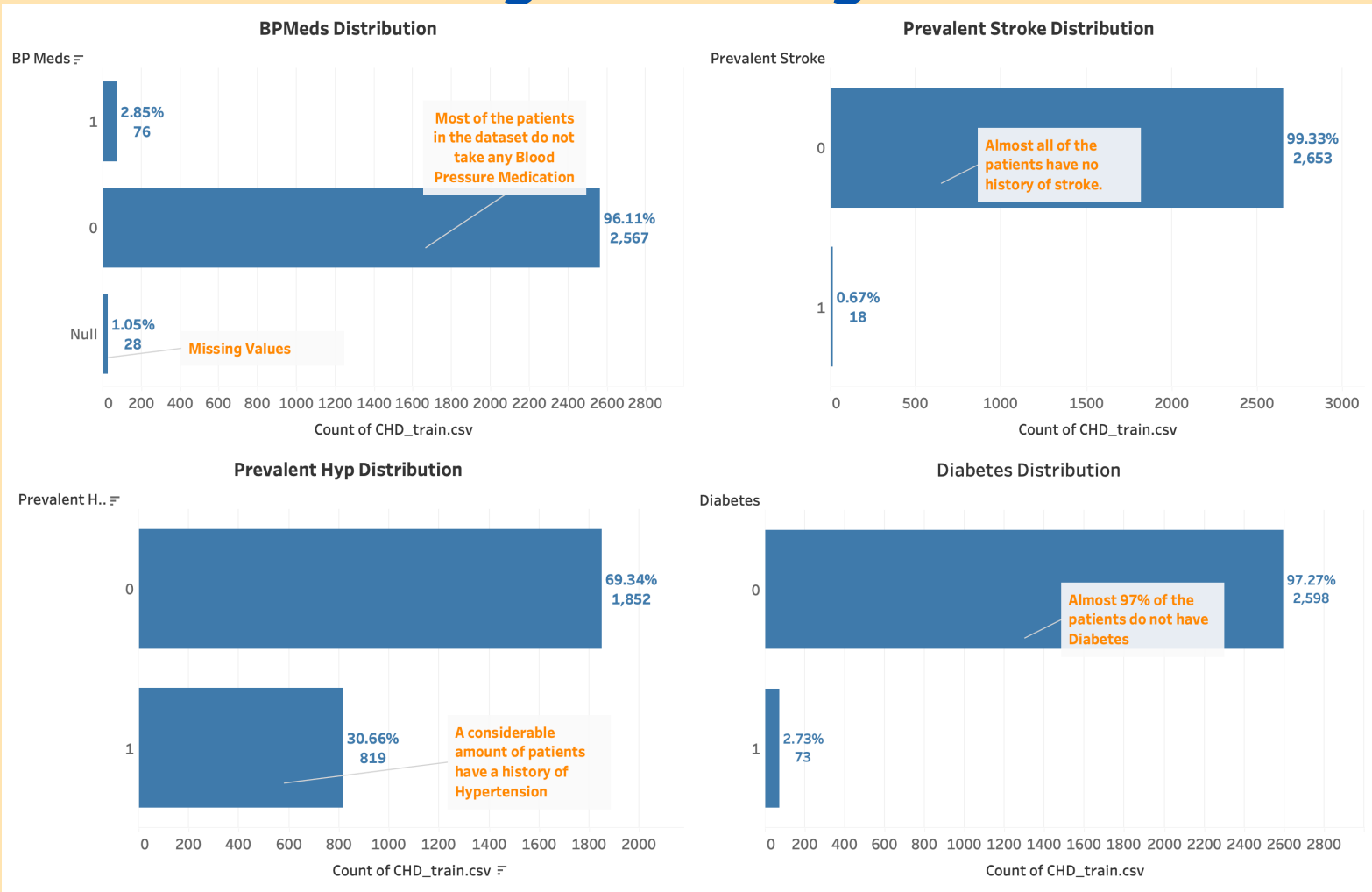
Univariate Analysis – Response Variable



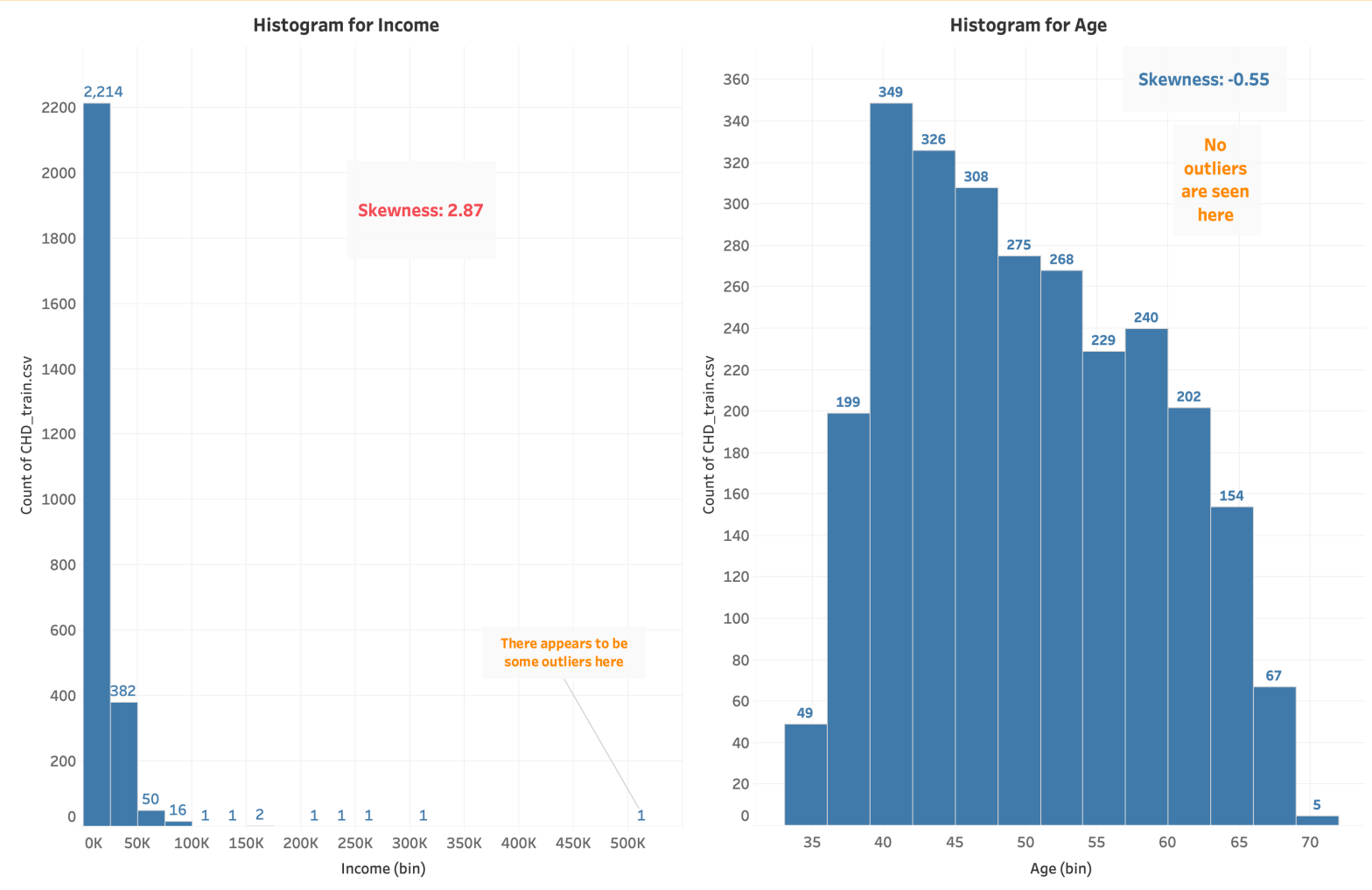
Univariate Analysis–Categorical Variables



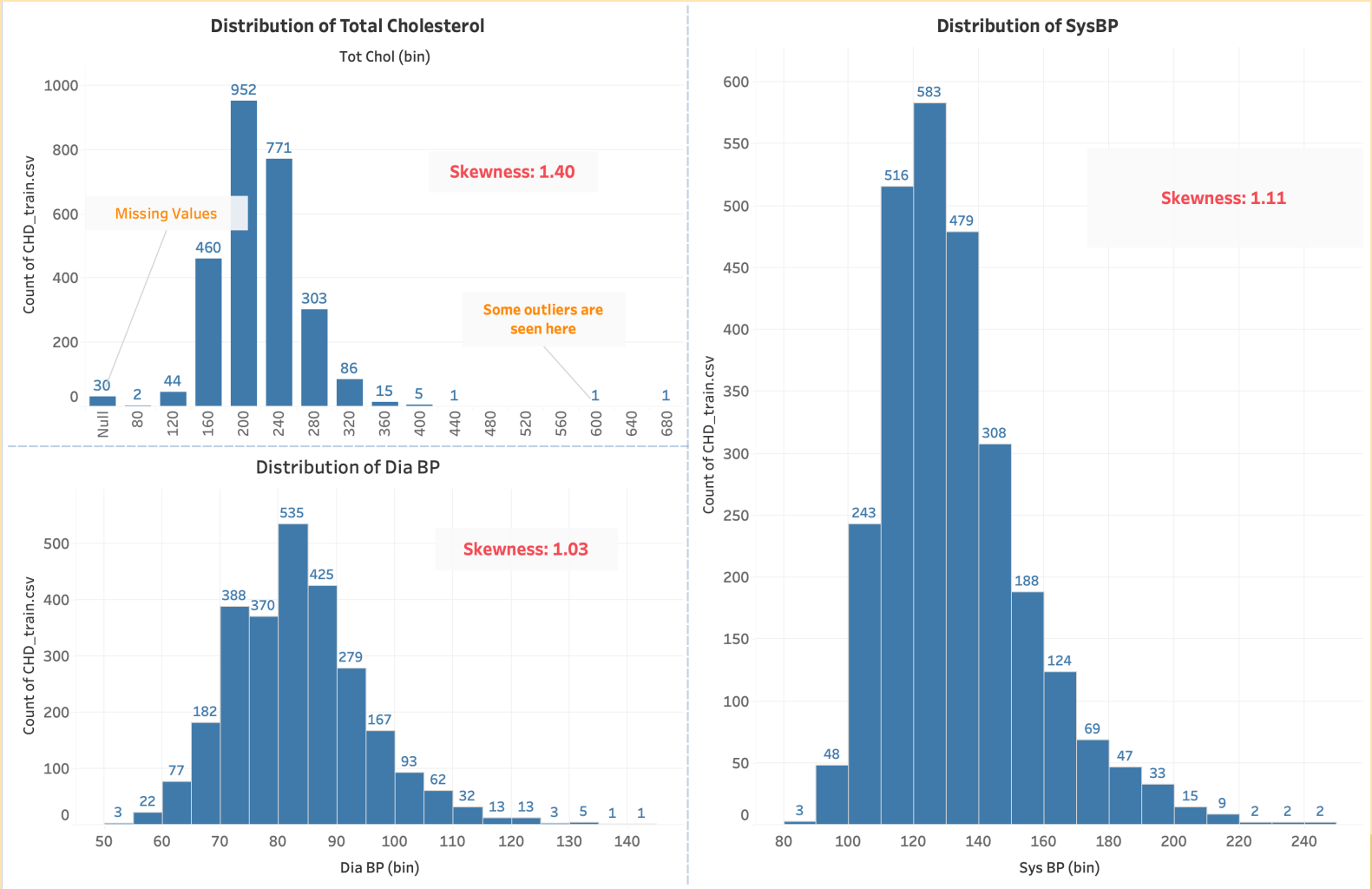
Univariate Analysis–Categorical Variables



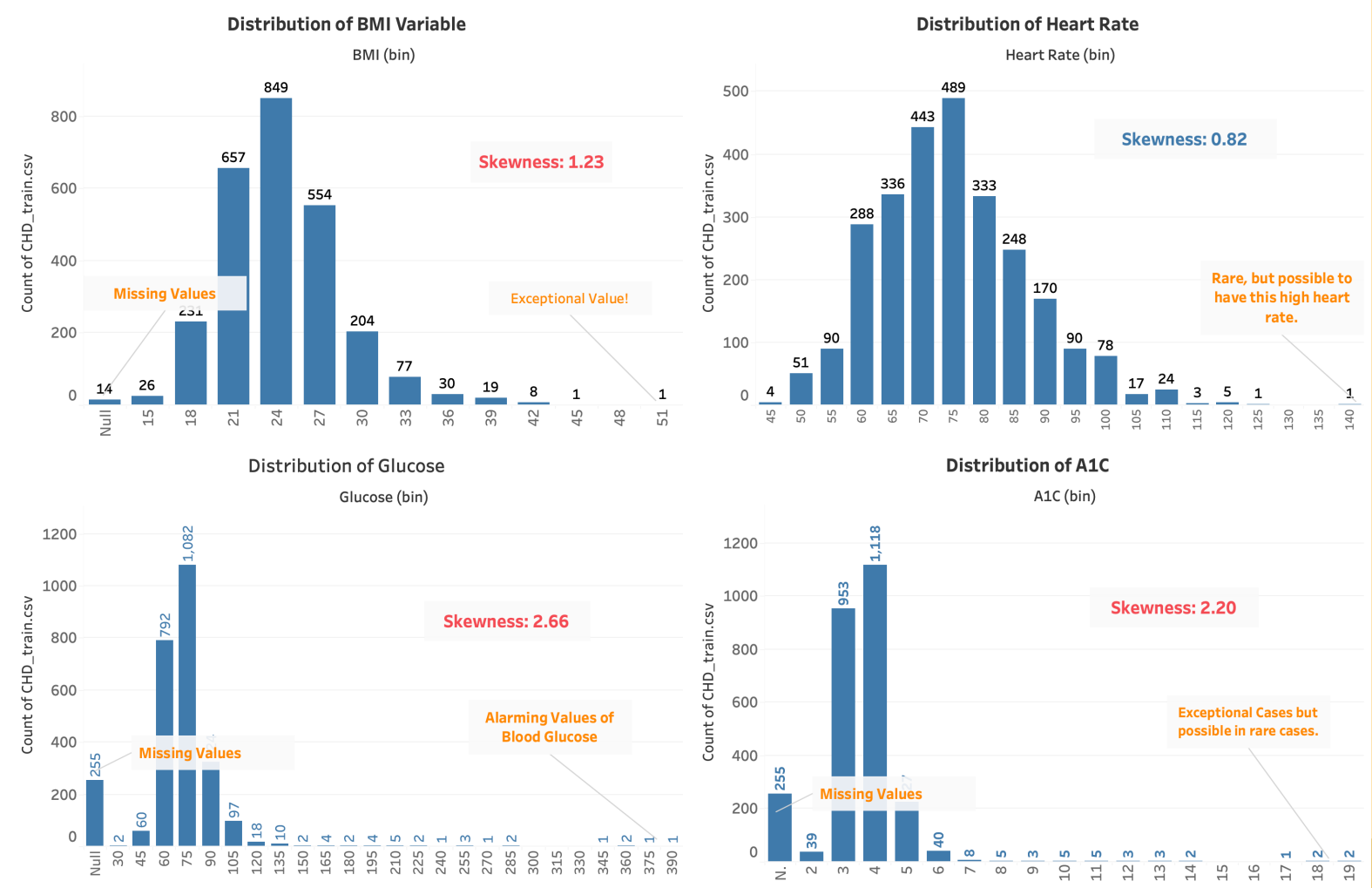
Univariate Analysis–Numeric Variables



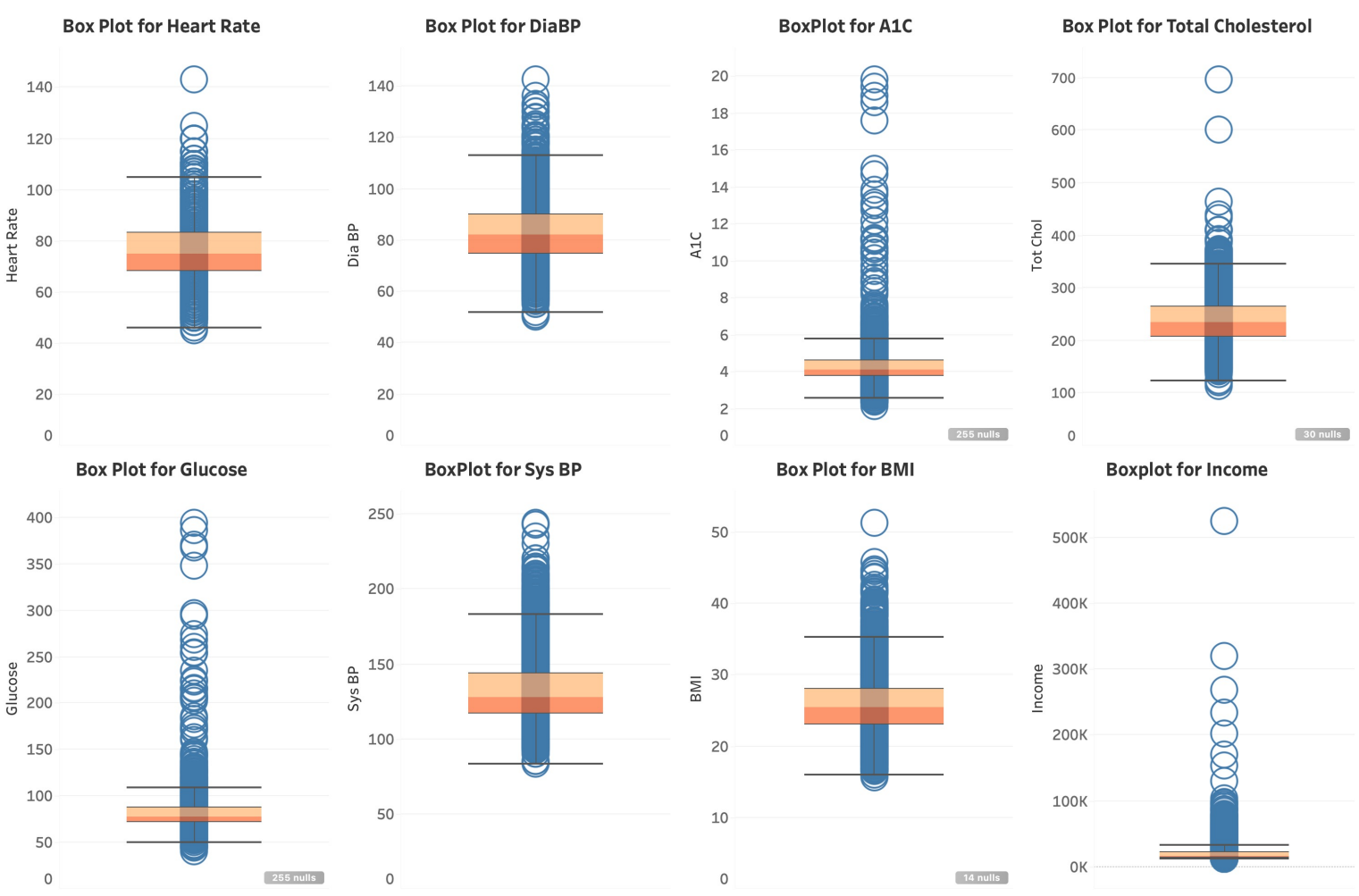
Univariate Analysis–Numeric Variables



Univariate Analysis–Numeric Variables

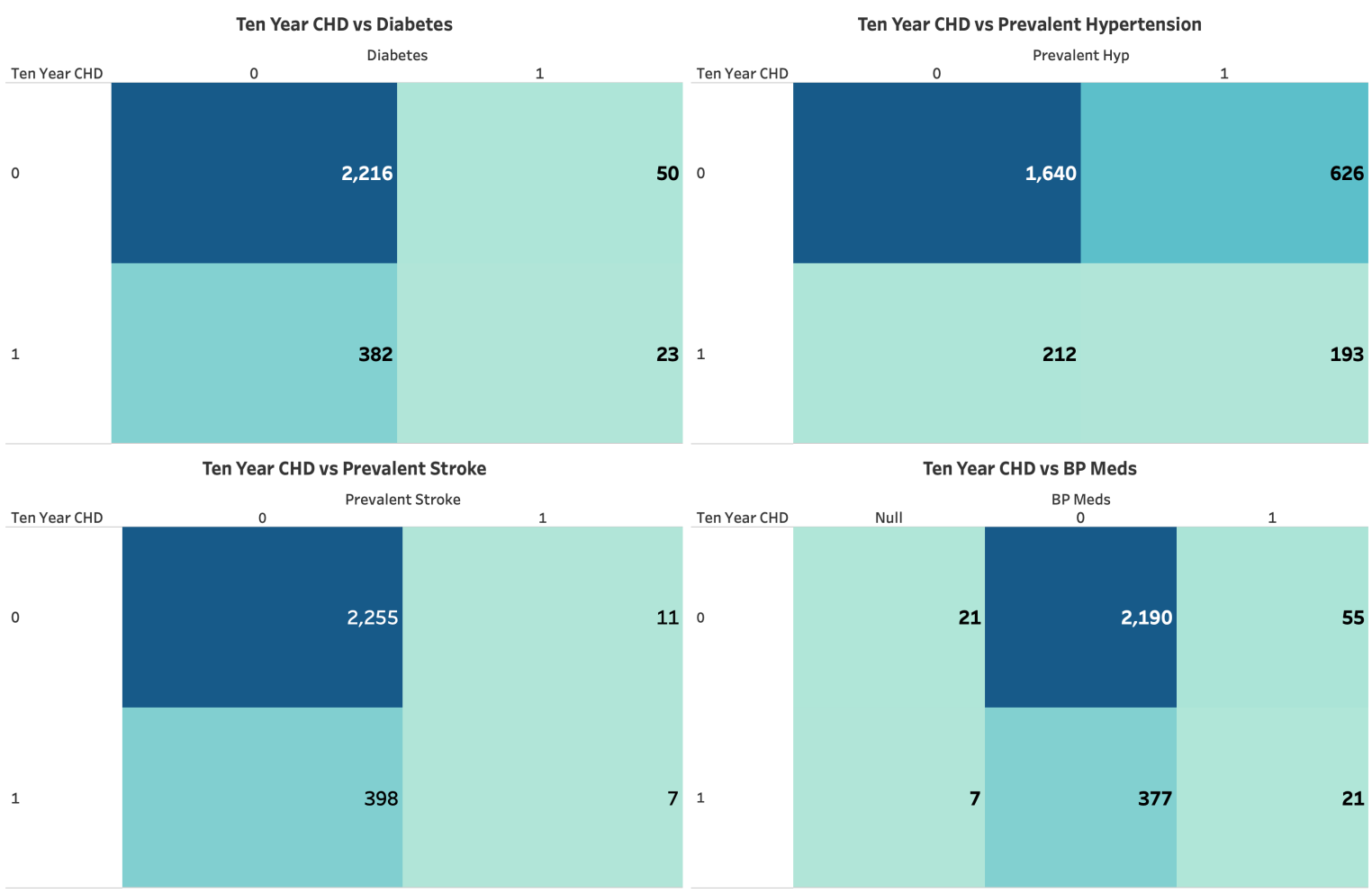


Univariate Analysis – Box Plots



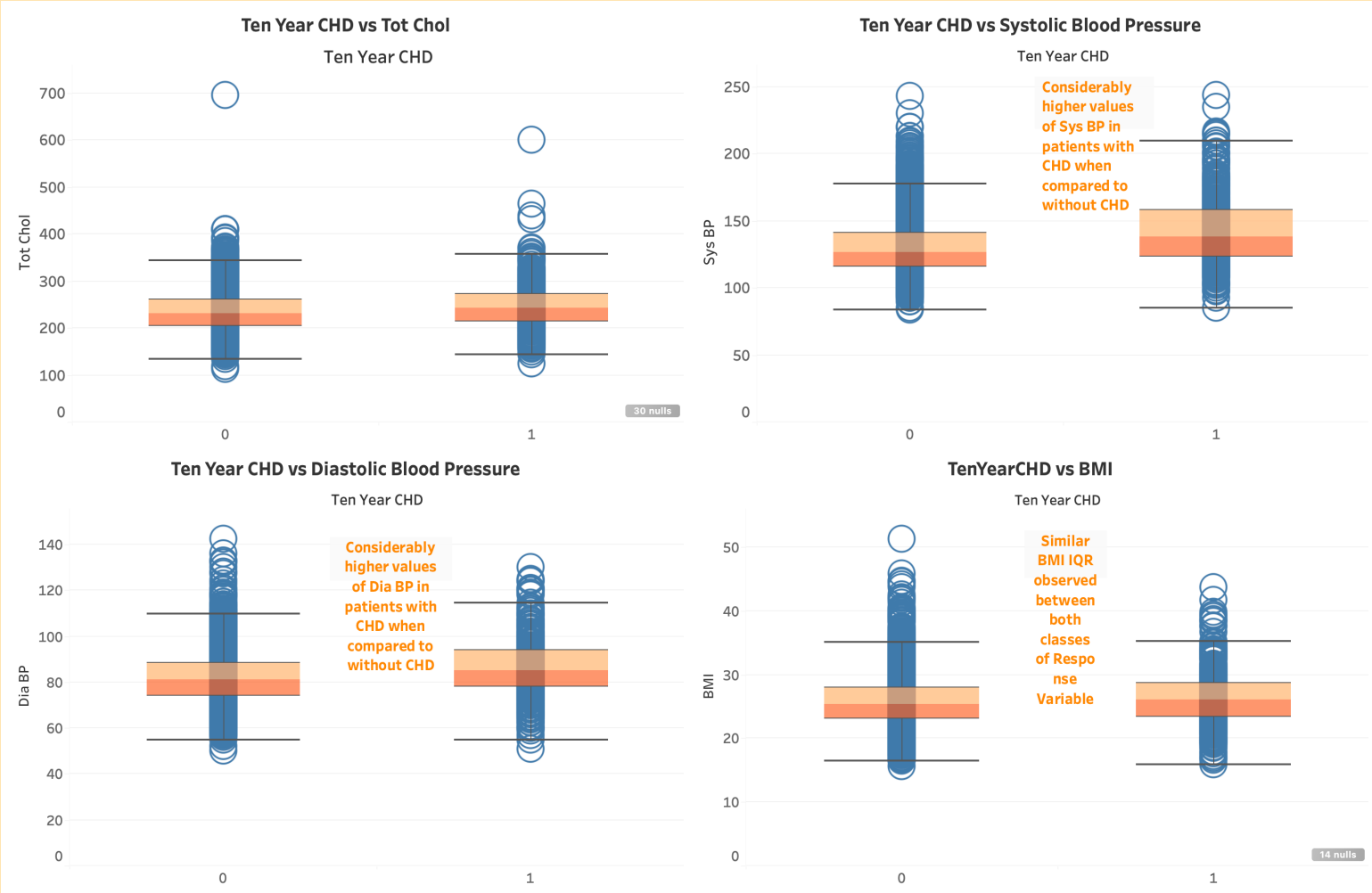
Bivariate Visualizations – Categories vs. Response

Bivariate Visualizations – Categories vs. Response



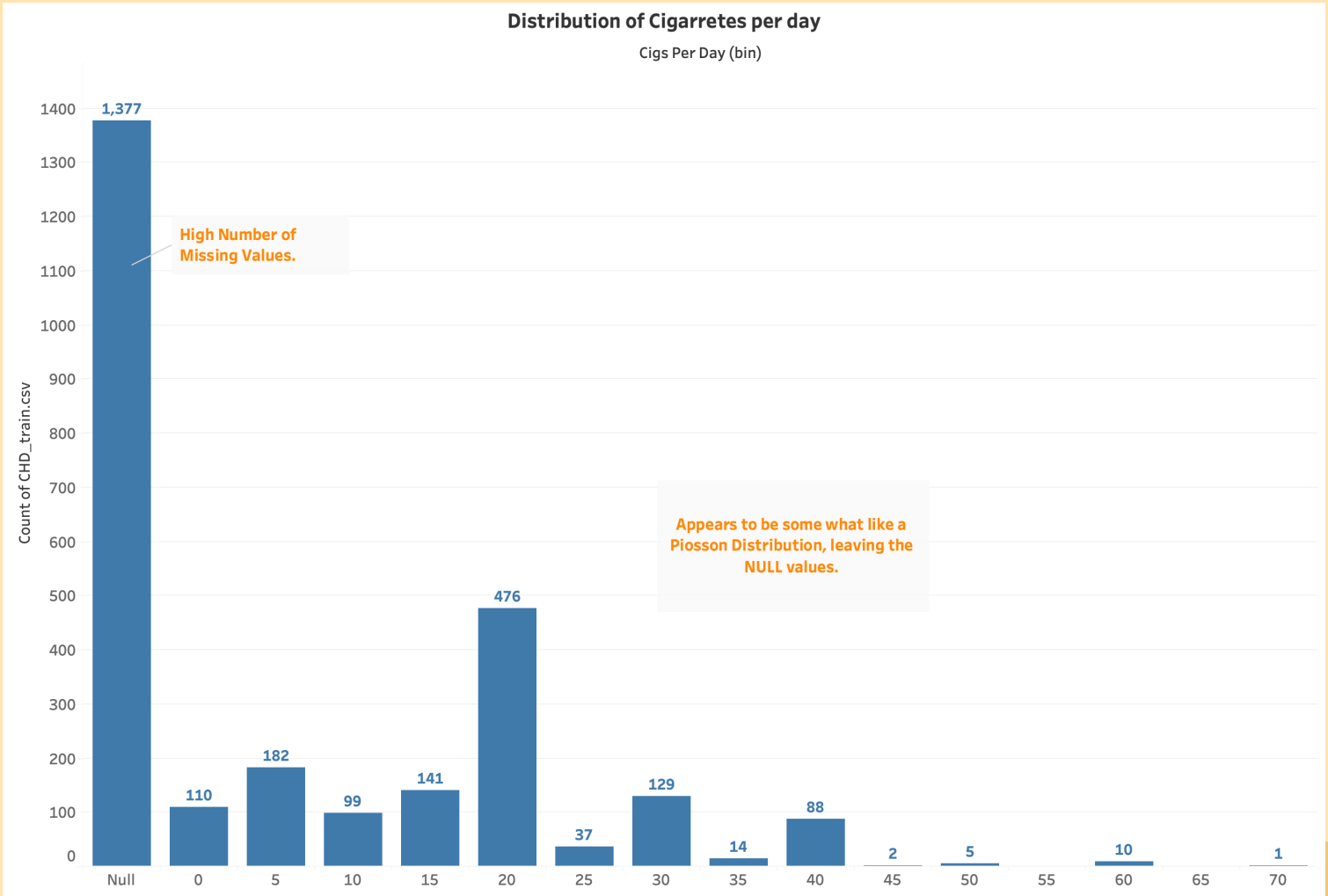
Bivariate Visualizations – Measures vs. Response

Bivariate Visualizations – Measures vs. Response

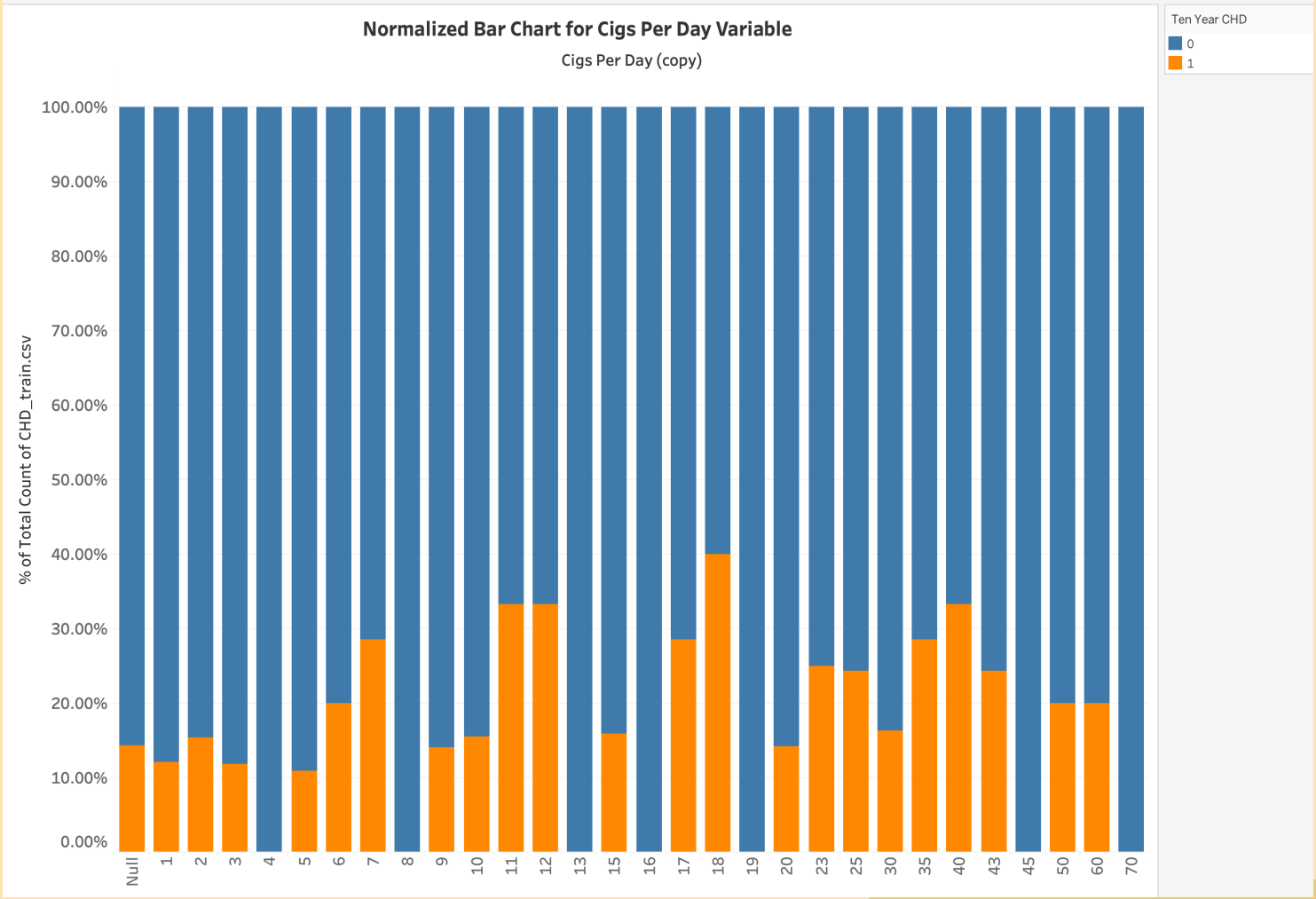


Bivariate Visualizations – Measures vs. Response

Univariate Analysis – Count Variable

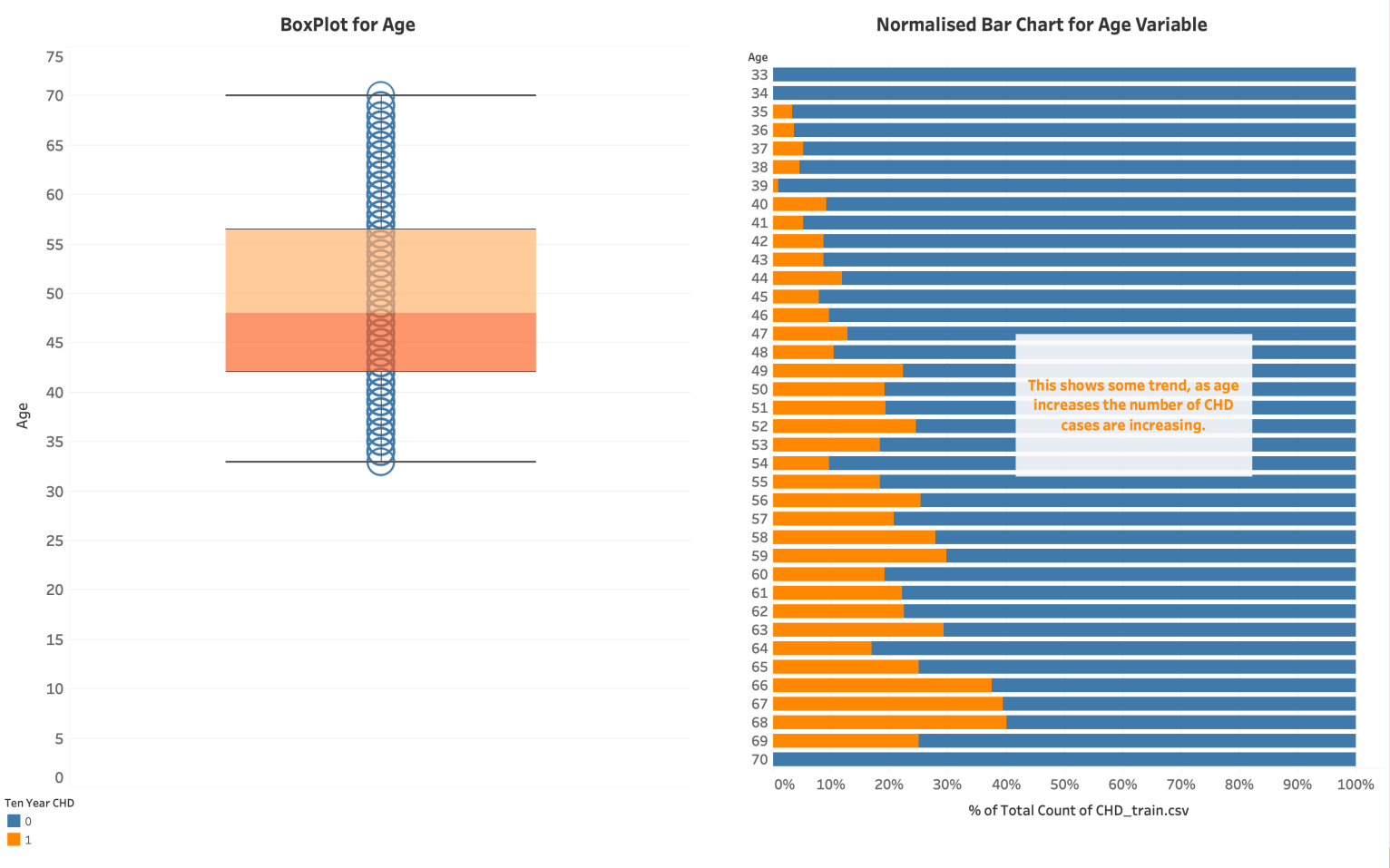


Bivariate Analysis – Count Variable vs Response



- The distribution is uneven and does not show any patterns.
- But there is high cardinality in the values for this feature.
- So we will treat the CigsPerDay variable as a numeric type only.

Univariate Analysis – Box Plot (Age)



Summary Statistics for numeric variables

```
[26] CHD_df.describe()
```

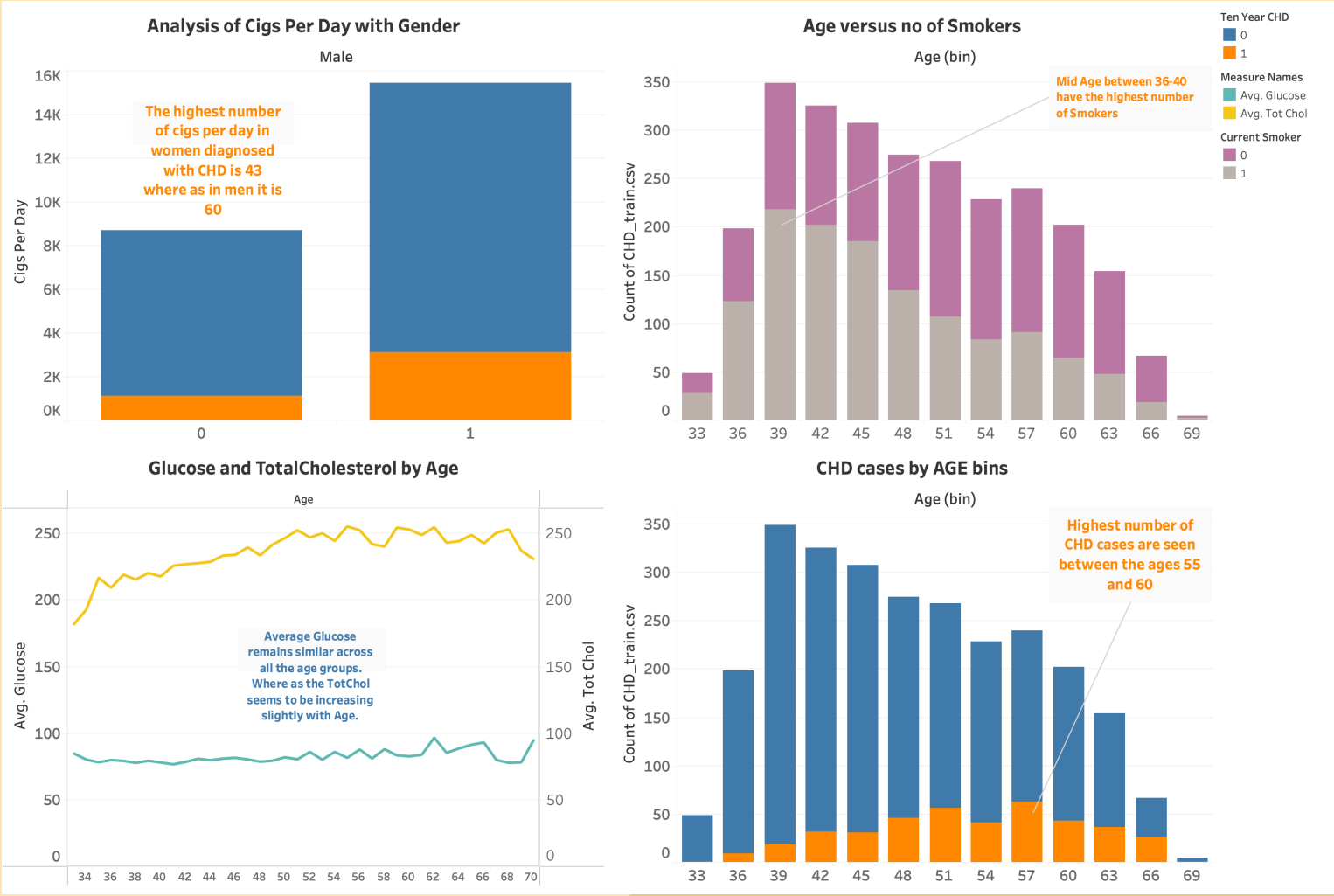
0s

	patientID	age	cigsPerDay	totChol	sysBP	diaBP	BMI	heartRate	glucose	a1c	income
count	2671.000000	2671.000000	1303.000000	2640.000000	2671.000000	2671.000000	2661.000000	2670.000000	2414.000000	2414.000000	2671.000000
mean	554797.277799	49.500936	18.526477	242.739773	132.472482	83.136653	25.787430	75.862921	81.586164	4.282015	20406.030326
std	255667.563662	8.566762	11.115330	223.897337	22.076228	11.968969	4.113838	12.253502	23.058886	1.168543	19254.845645
min	100002.000000	32.000000	1.000000	107.000000	83.500000	52.000000	15.540000	44.000000	40.000000	2.134769	12000.000000
25%	337162.000000	42.000000	10.000000	205.000000	117.000000	75.000000	23.050000	68.000000	71.000000	3.732105	13562.000000
50%	557875.000000	49.000000	20.000000	234.000000	128.000000	82.000000	25.380000	75.000000	78.000000	4.125298	15929.000000
75%	771492.500000	56.000000	20.000000	264.000000	144.000000	90.000000	28.020000	83.000000	87.000000	4.560329	21373.000000
max	999826.000000	70.000000	70.000000	9280.000000	295.000000	142.500000	56.800000	143.000000	394.000000	19.917371	524494.000000

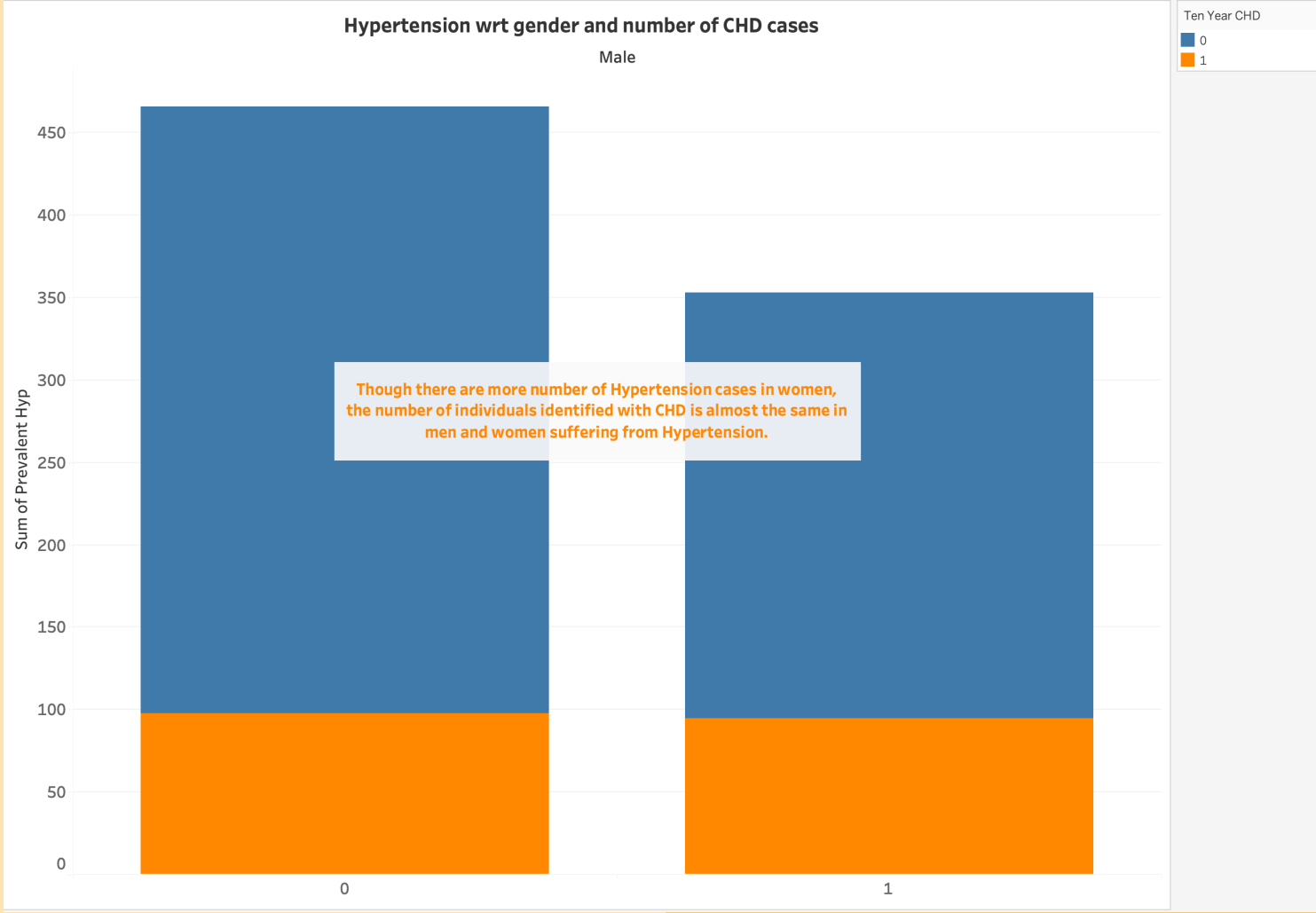
Distribution Summary for Categorical Variables

Characteristic	0, N = 2,269[1]	1, N = 402[1]
male		
0	1,321 (58%)	197 (49%)
1	948 (42%)	205 (51%)
education		
1	899 (41%)	197 (51%)
2	676 (30%)	88 (23%)
3	384 (17%)	57 (15%)
4	258 (12%)	48 (12%)
Unknown	52	12
currentSmoker		
0	1,159 (51%)	191 (48%)
1	1,110 (49%)	211 (52%)
BPMeds		
0	2,196 (98%)	371 (94%)
1	45 (2.0%)	25 (6.3%)
Unknown	28	6
prevalentStroke		
0	2,261 (100%)	392 (98%)
1	8 (0.4%)	10 (2.5%)
prevalentHyp		
0	1,638 (72%)	195 (49%)
1	631 (28%)	207 (51%)
diabetes		
0	2,223 (98%)	379 (94%)
1	46 (2.0%)	23 (5.7%)

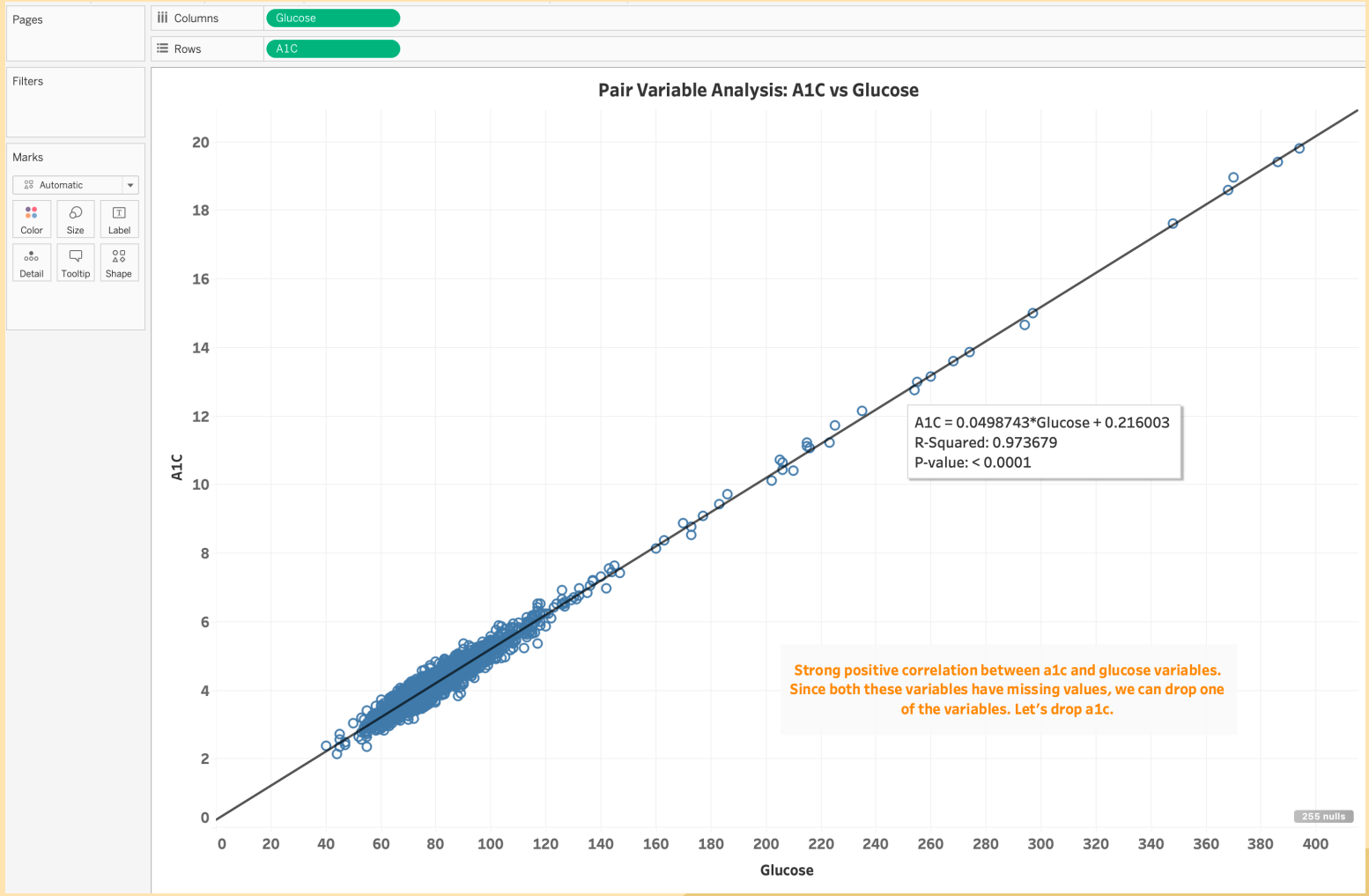
Bivariate Analysis – Other pairs of variables



Multivariate Analysis

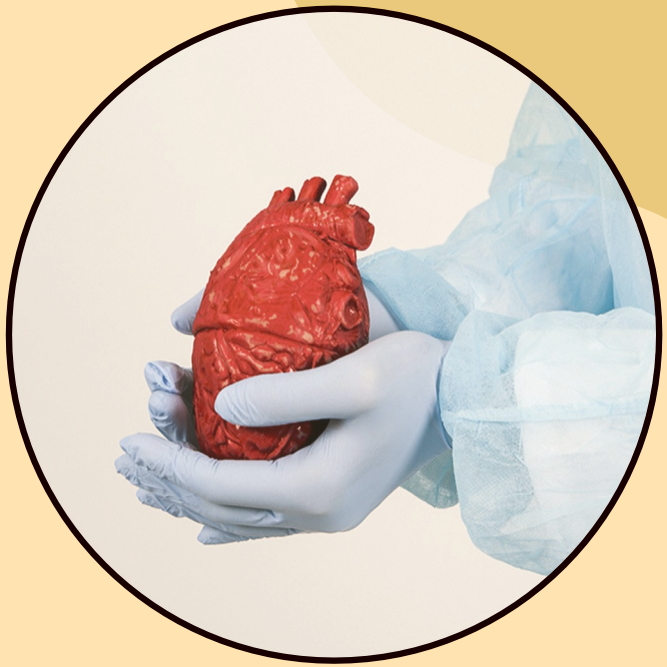


Correlation between glucose and a1c



Data Preparation Plan

- Data Quality Issues and Actions
- Feature Selection decisions
- Feature Engineering decisions
- Dataset partitioning decisions



Data Quality Issues and Actions

Outliers

The outliers were treated first, even before the missing data because of the following benefits:

- Stability of Imputation: Addressing the outliers first may lead to a more stable imputation process for missing values.
- Data Integrity: Outliers can sometimes represent erroneous data points or extreme values that are not reflective of the underlying population. By addressing outliers first, we can ensure that the imputed missing values are based on a more representative range of data.

All the extreme outliers for the numeric variables were treated using the Clipping technique.

Clipping

Clipping is a data pre-processing technique used to handle outliers in numerical data. Clipping involves setting a threshold beyond which the values of the data points are truncated or capped. Here we have used the 3-Sigma Clipping technique where the range is defined as ±3 standard deviations from the mean. Any value that falls outside this range is replaced by the corresponding threshold value.

Data Quality Issues and Actions

Missing Values

We have seen in the EDA that there are 8 variables with missing values. Each of these variables were addressed using an appropriate technique.

- totChol, BMI, glucose, a1c, heartRate (Numeric Variables):

Impute with respective median values.

- Education (Ordinal Categorical):

Assign a new category for the missing values.

- BPMeds (Nominal Categorical, Binary):

Imputed it with 1 when the sysBP > 120 and diaBP > 80 (the normal ranges of BP) and imputed with 0 otherwise.

- CigsPerDay (being treated as numeric variable):

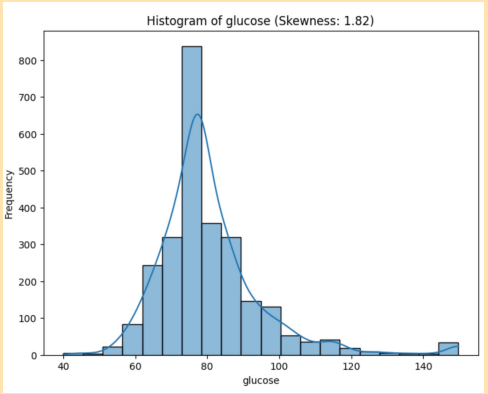
Imputed with 0, assuming the missing values means that they do not smoke .

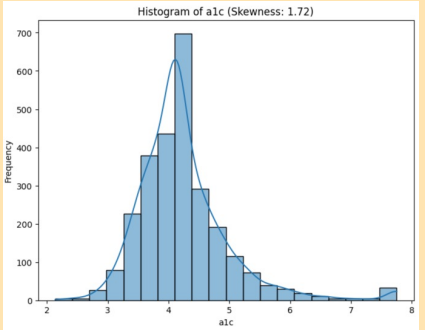
Data Quality Issues and Actions

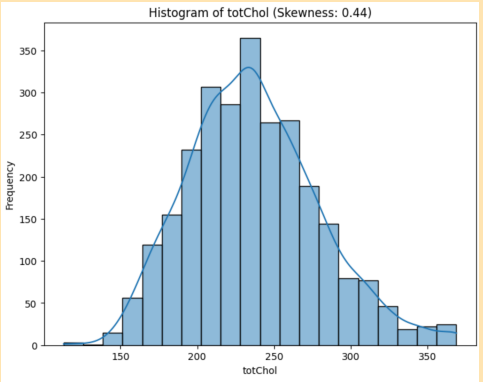
Skewness Analysis

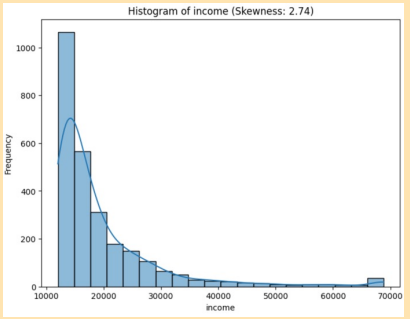
From the histogram analysis we can see that most of the measures have strong positive skew. But after the outliers are treated using 'Clipping', these are skewness values of the features. The ones in red are high (above 1.5) and need to be treated. We will perform **log transform** on these variables.

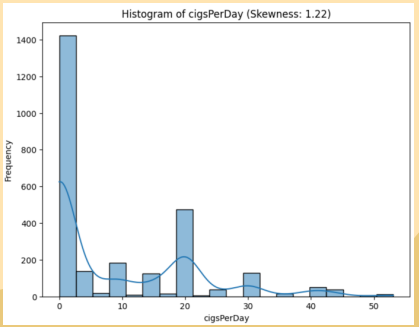
Variable	Skewness Statistic
Age	0.24
Total Chol	0.44
Heart Rate	0.47
Dia BP	0.49
BMI	0.61
Sys BP	0.87
cigsPerDay	1.22
Glucose	1.82
A1C	1.72
Income	2.74







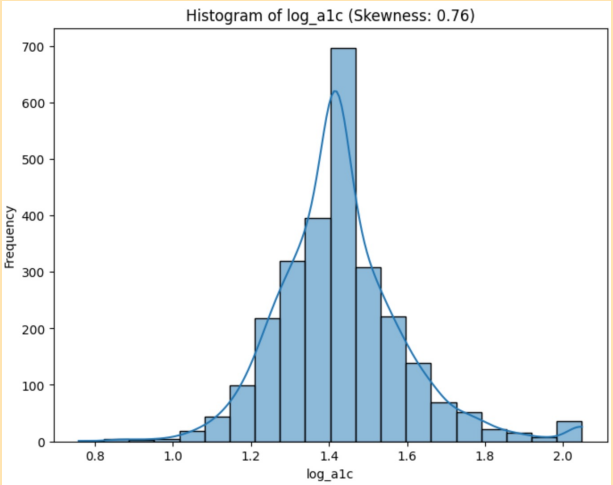


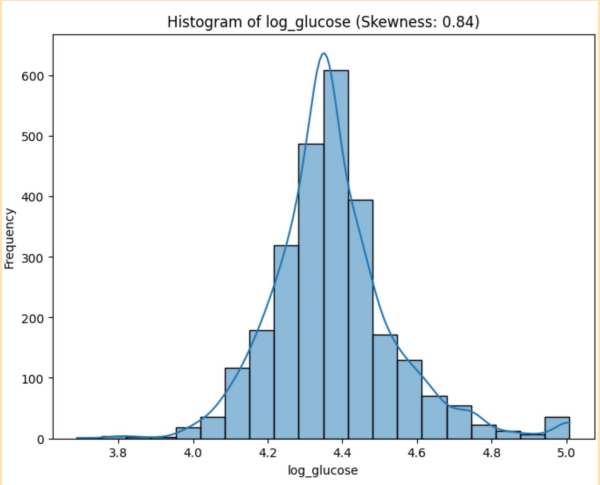


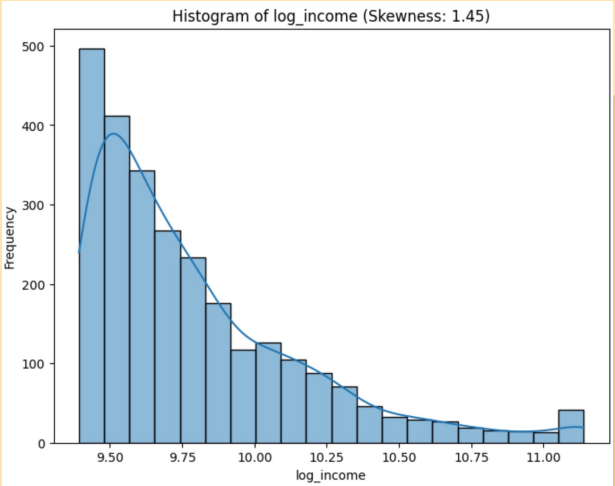
Data Quality Issues and Actions

Treating Skewness – Log Transforms

Applied log transforms on the high skewed variables, as seen previously. Now these are the new skew values of the transformed variables. All are below 1.5 skewness, which will be even reduced slightly when we perform normalization.





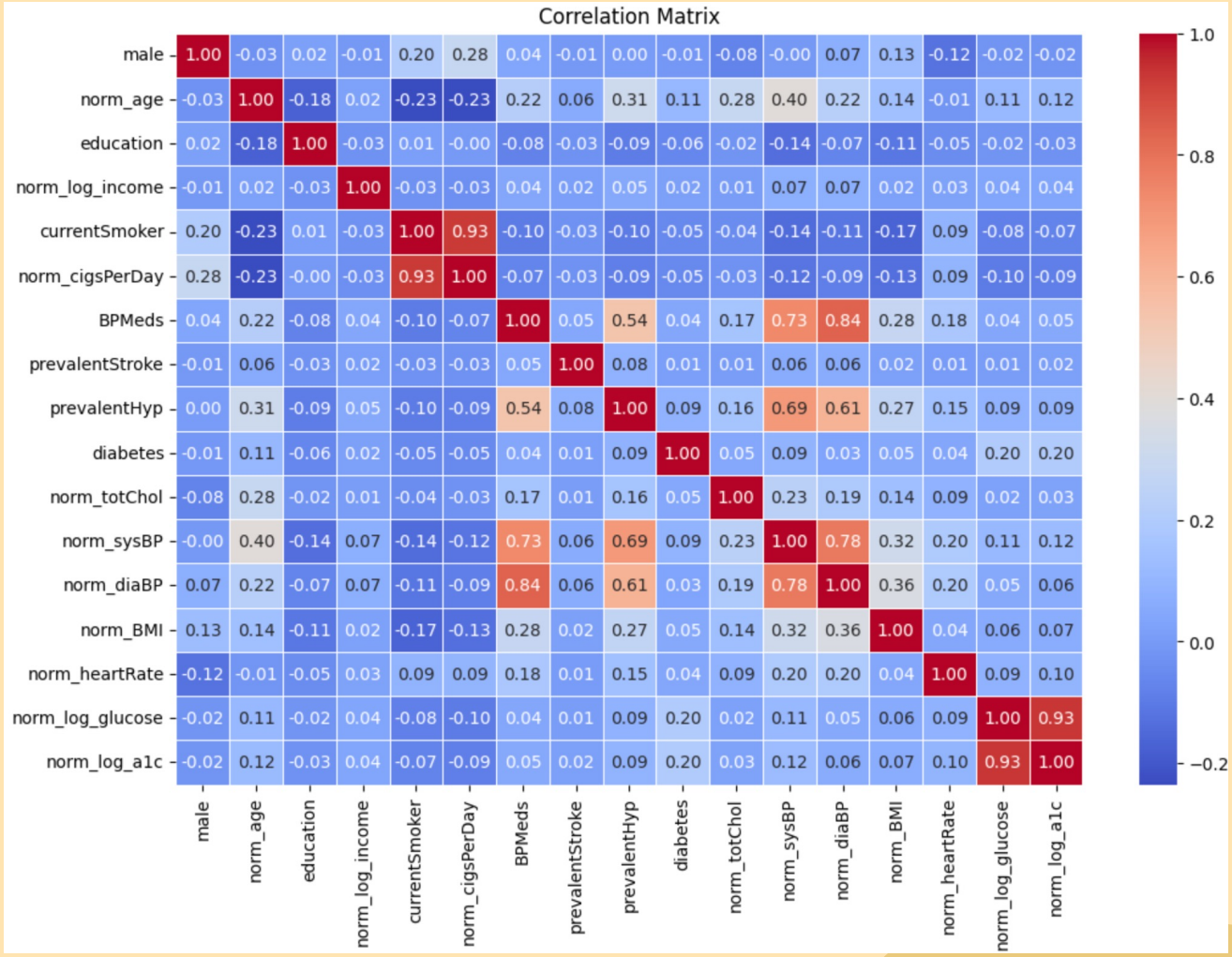


Data Quality Issues and Actions

Feature Scaling:
This becomes necessary when we have features in different units of measurement, ranges and magnitudes. For instance, here we have features like income, age, glucose. All these have different scales and can result in a bias which will further affect the performance of the model. We can use either Normalization or Standardization based on the distribution of our data. Both techniques were implemented and Normalization had a slightly better performance and was used in the final pipeline.

Normalization:
Also known as min-max scaling, applied to numeric columns where the values are shifted and rescaled so that they end up ranging between 0 and 1.

Correlation Analysis



- Correlation Analysis on the processed data, that is the features generated after all the actions taken above.
- Spearman Correlation was used as it can be performed on both numeric and categorical variables.

Handling Ordinal Categories

Education

The education variable represents the educational level of patients and is of the ordinal category type, where the order of categories holds significance for model interpretation.

Approaches:

1.Thermometer Scale Technique:

- Missing values were replaced with the most common class of the education variable.
- Categories were encoded using a thermometer scale approach, assigning numerical values based on the educational attainment level: 1 for less than high school, 2 for completed high school or equivalent, 3 for some college, and 4 for completed college or higher.

2.General Categorical Variable:

- Missing values were replaced with a new class labeled as '5: not known'.
- This approach simplifies handling missing values and results in five categories for the education variable.

Outcome: Both approaches yielded similar f1 scores. However, the second approach was selected due to its efficiency in managing missing values and its straightforward implementation within the pipeline.

Data Quality Issues and Actions

Data Preparation for Validation and Test Datasets

- All the Data Quality Issues and Actions seen above for the training datasets, were performed similarly for the Validation and Test Datasets.

Data Leakage

- To avoid Data Leakage, the imputations and normalization used the values from the training set as shown in the pipelines further.

Feature Selection Decisions

LASSO CV (cross validation) was used for feature selection and regularization in this classification task. This technique helps in feature selection by shrinking some coefficients to zero, preventing overfitting and ensures that the model generalizes well to new data in performing the task. These are the selected features by the LASSO CV.

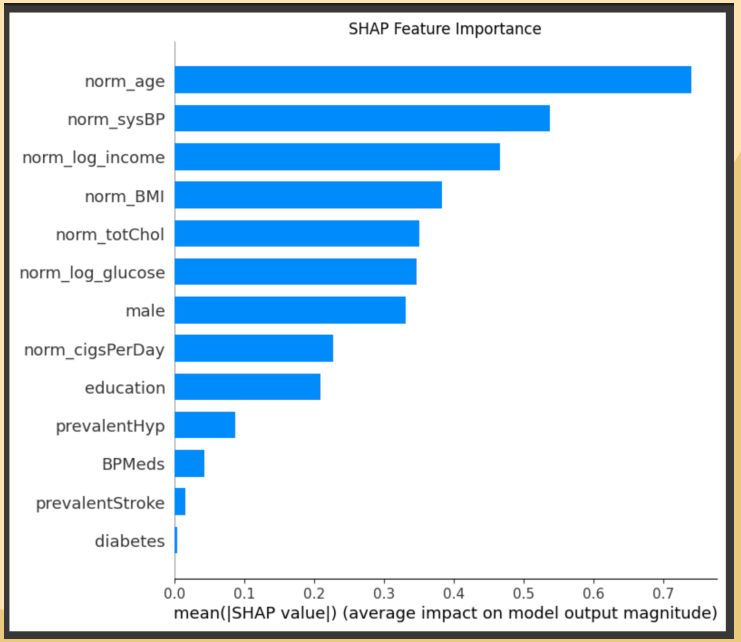
Selected Features: ['male', 'norm_age', 'education', 'norm_cigsPerDay', 'BPMeds', 'prevalentStroke', 'prevalentHyp', 'diabetes', 'norm_totChol', 'norm_sysBP', 'norm_BMI', 'norm_log_income', 'norm_log_glucose']

Feature Selection Decisions

SHAP

Following this, the features selected from LASSO CV were used to train different classifiers and the SHAP scores for the features were calculated for these models. The SHAP values for the model trained using the XGBoost Classifier is shown below.

Most have the classifiers showed that the features 'prevanlentStroke' and 'diabetes' had a very low SHAP value which means they are not so important in predicting the response.



Feature Selection Decisions

From the above analysis the following features were decided to be dropped before the model development.

- currentSmoker

The correlation matrix in the EDA shows very high correlation between 'currentSmoker' and 'cigsPerDay' variables. Also the LASSO CV has not shown any importance to the 'currentSmoker' feature.

- prevalentStroke, diabetes

Low SHAP scores, also model performance dropped very minutely when these variables were included.

- diaBP, heartRate

Not selected by LASSO CV, also model performance did not change by including these variables.

- a1c

From EDA, we know that a1c and glucose have very strong positive correlation, so we need to drop one of these variables.

Feature Selection Decisions

The final features which were selected to be used in the model are as follows:

- male
- norm_age
- education
- BPMeds
- norm_cigsPerDay
- prevalentHyp
- norm_log_income
- norm_totChol
- norm_sysBP
- norm_BMI
- norm_log_glucose

Feature Engineering Decisions

- Log-transforms on income, glucose, a1c variables due to strong positive skew.
- Education variable converted to categorical with an additional category for missing values.
- All numeric features were normalized, to be in a similar scale.
- Additional features generated

- Mean Arterial Pressure (MAP) = (2*(diaBP) + (sysBP))/ 3
- Avg_glucose_a1c = (glucose+a1c)/2 (to handle the high correlation)

But these features did not improve the model performance, and also the model performance dropped in classifiers like SVM. So these features were not used in the final model.

Dataset Partitioning Decisions

The dataset was split into 70% -30% split for training and validation. As we have seen in the EDA (performed on the train dataset), the dataset has high imbalance in the response variable. To solve this, the following 2 techniques were tried:

- **SMOTE – Synthetic Minority Over-sampling Technique**

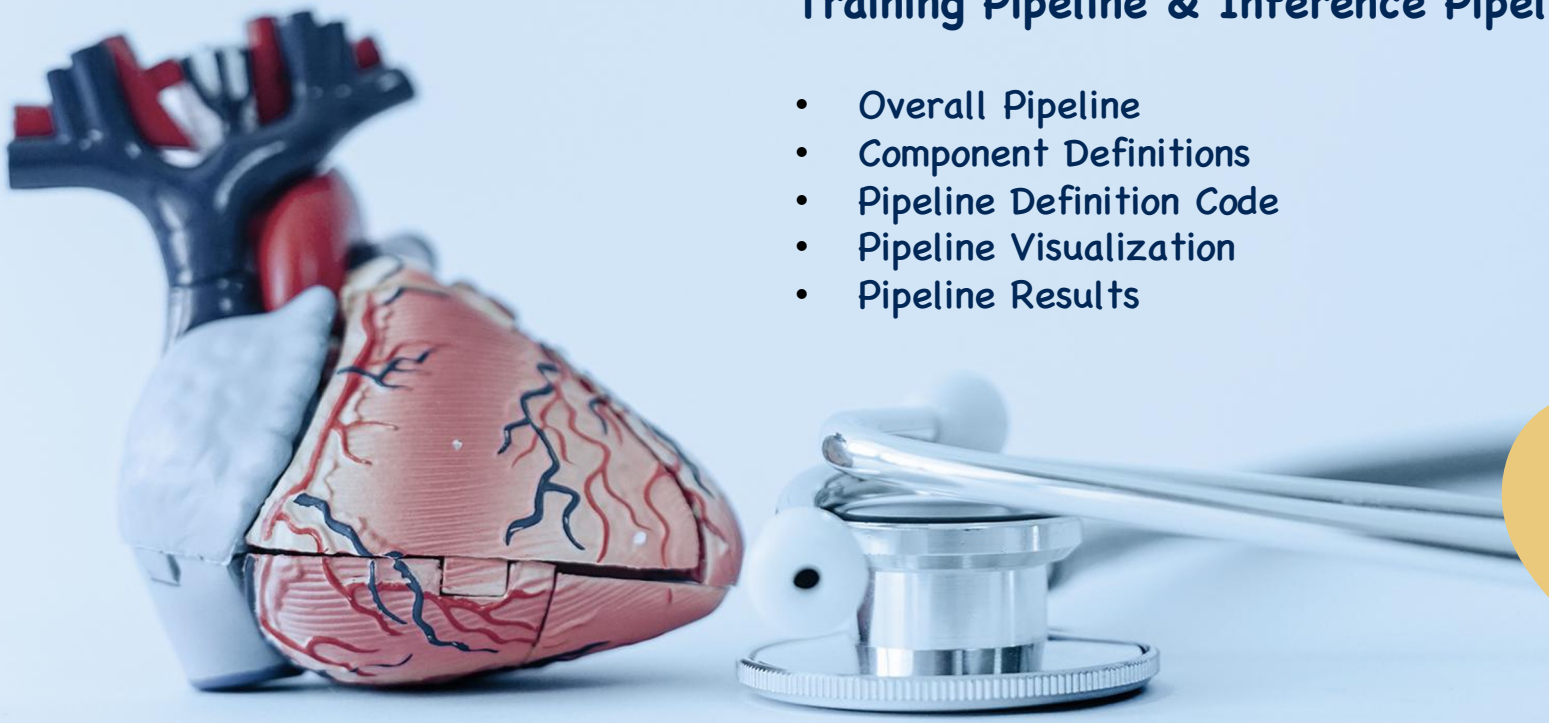
This works by generating synthetic samples for the minority class, and aims to balance the class distribution in the dataset. This helps to prevent the model from being biased towards the majority class and improves the model performance.

- **ROS – Random Over Sampling**

This technique, works similarly to SMOTE, here the minority class samples are randomly selected with replacement until the desired balance between the minority and majority classes is achieved.

ROS yielded a slightly higher f1-score compared to the SMOTE and hence it was used in the final model pipeline.

MODEL PIPELINE

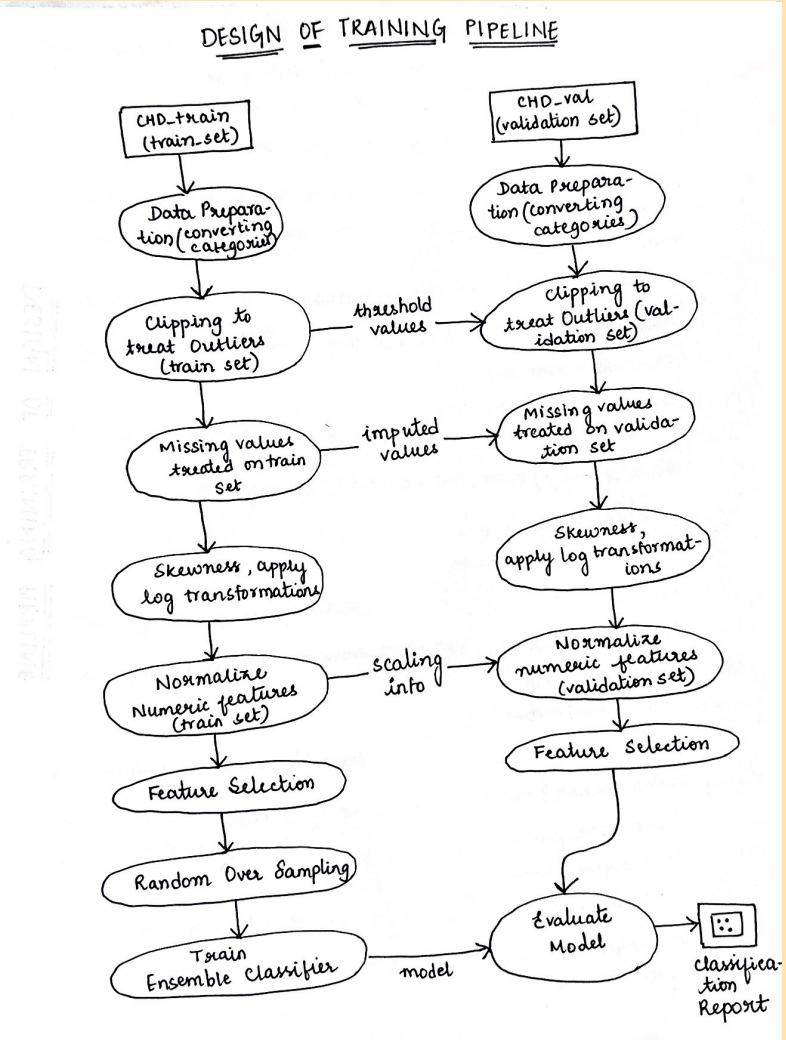


Training Pipeline & Inference Pipeline

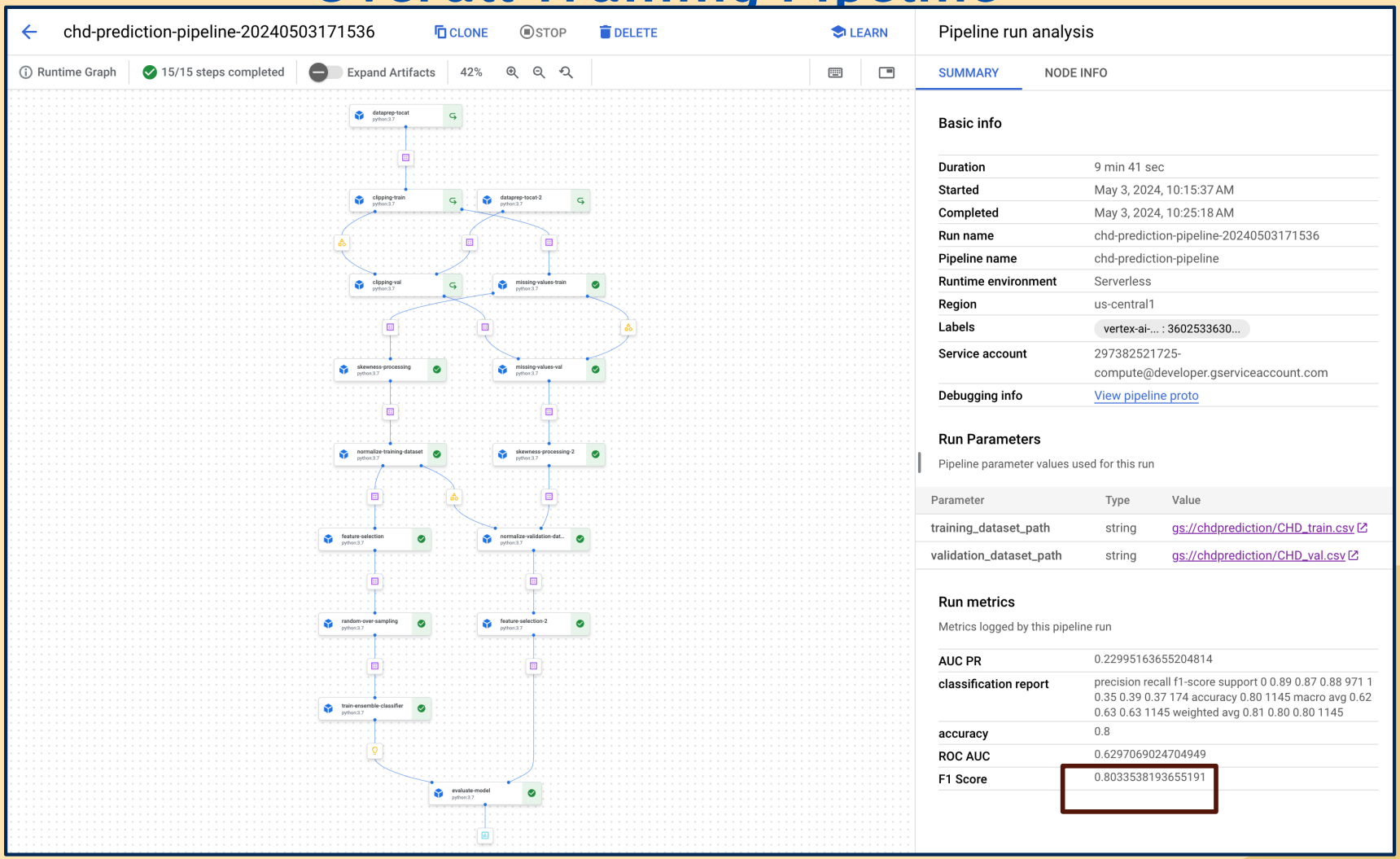
- Overall Pipeline
- Component Definitions
- Pipeline Definition Code
- Pipeline Visualization
- Pipeline Results

Training Pipeline

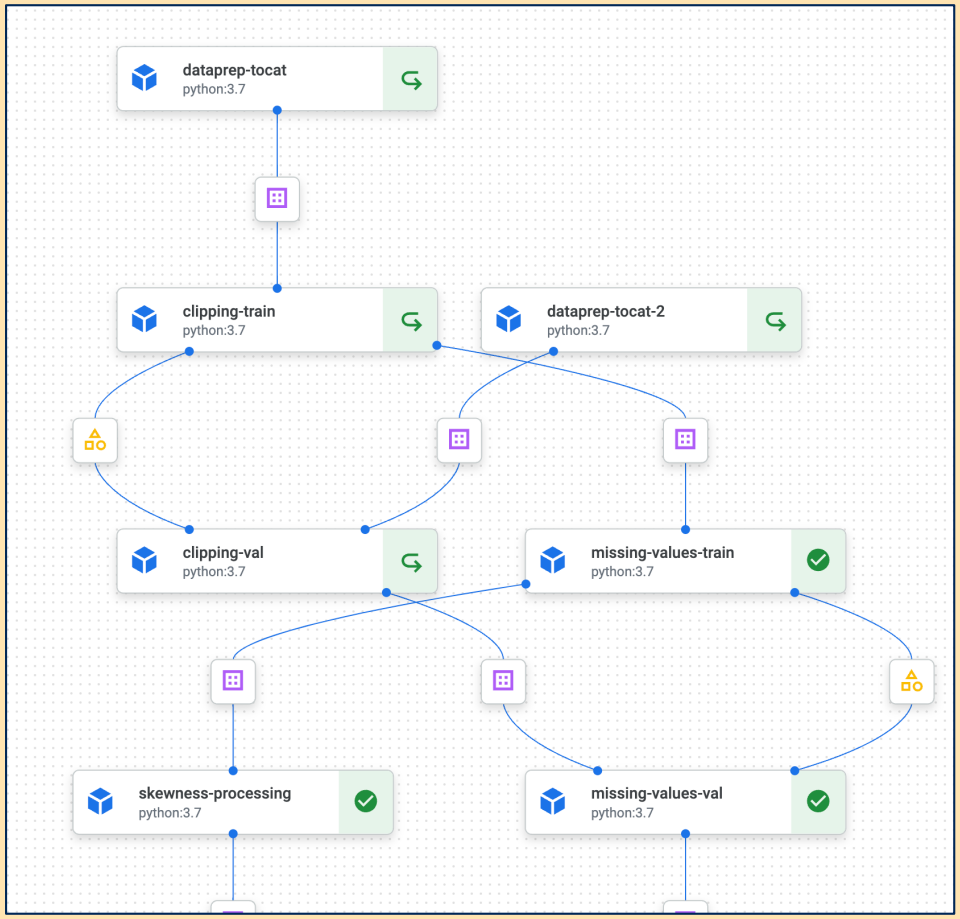
Design – Training Pipeline

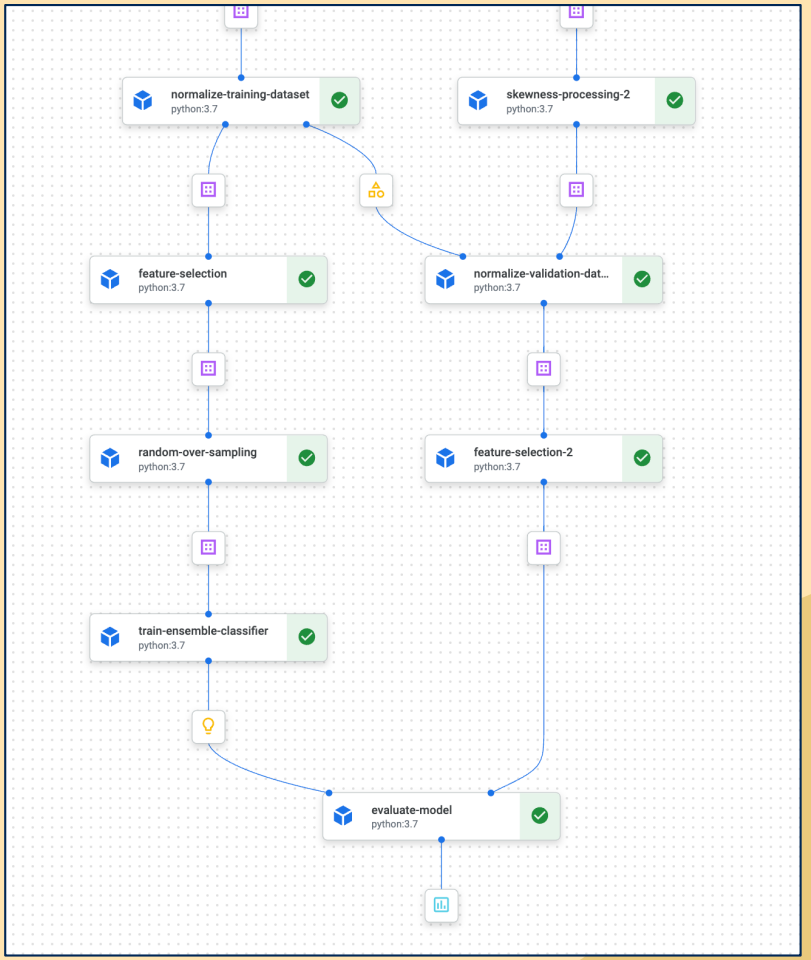


Overall Training Pipeline



Training Pipeline Visualization





Training Pipeline Evaluation Results

Run Parameters

Pipeline parameter values used for this run

Parameter	Type	Value
training_dataset_path	string	gs://chdprediction/CHD_train.csv
validation_dataset_path	string	gs://chdprediction/CHD_val.csv

Run metrics

Metrics logged by this pipeline run

AUC PR	0.22995163655204814
classification report	precision recall f1-score support 0 0.89 0.87 0.88 971 1 0.35 0.39 0.37 174 accuracy 0.80 1145 macro avg 0.62 0.63 0.63 1145 weighted avg 0.81 0.80 0.80 1145
accuracy	0.8
ROC AUC	0.6297069024704949
F1 Score	0.8033538193655191

Training Pipeline Definition Code

Define pipeline

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

@pipeline(name = 'CHD_prediction_pipeline')

def CHD_prediction_pipeline(training_dataset_path : str, validation_dataset_path : str):

  #Process training dataset preparation - Converting to Categorical
  training_data_preparation = dataprep_tocat(input_dataset_path = training_dataset_path)

  #Treating outliers - Clipping (Training dataset)
  training_data_outliers = clipping_train(training_dataset_path = training_data_preparation.outputs['output_dataset_path'])

  #Treating Missing Values - Imputations (training dataset)
  training_data_missingvalues = missing_values_train(training_dataset_path = training_data_outliers.outputs['output_dataset_path'])

  #Treating skewness - log transforms (training dataset)
  training_data_skewness = skewness_processing(input_dataset_path = training_data_missingvalues.outputs['output_dataset_path'])

  #Normalization (training dataset)
  training_data_normalized = normalize_training_dataset(training_dataset_path = training_data_skewness.outputs['output_dataset_path'])

  #Feature Selection (training dataset)
  training_data_final = feature_selection(input_dataset_path = training_data_normalized.outputs['output_dataset_path'])

  #Perform Random Oversampling
  oversampled_training_data = random_over_sampling(input_dataset_path = training_data_final.outputs['output_dataset_path'])

  #Train ensembled classifier
  trained_model = train_ensemble_classifier(training_dataset_path = oversampled_training_data.outputs['output_dataset_path'])
```

Training Pipeline Definition Code

```
#Process validation dataset preparation - Converting to Categorical
validation_data_preparation = dataprep_tocat(input_dataset_path = validation_dataset_path)


#Treating Outliers - Clipping (validation dataset)
validation_data_outliers = clipping_val(val_dataset_path = validation_data_preparation.outputs['output_dataset_path'],
                                        clipping_info = training_data_outliers.outputs['output_clipping_info'])

#Treating Missing Values - Imputation (validation dataset)
validation_data_missingvalues = missing_values_val(val_dataset_path = validation_data_outliers.outputs['output_dataset_path'],
                                                   missing_info = training_data_missingvalues.outputs['output_missing_info'])

#Treating Skewness log transforms (validation dataset)
validation_data_skewness = skewness_processing(input_dataset_path = validation_data_missingvalues.outputs['output_dataset_path'])

#Normalization (validation dataset)
validation_data_normalized = normalize_validation_dataset(val_dataset_path = validation_data_skewness.outputs['output_dataset_path'],
                                                          scaling_info_input = training_data_normalized.outputs['output_scaling_info'])

#Feature Selection (validation dataset)
validation_data_final = feature_selection(input_dataset_path = validation_data_normalized.outputs['output_dataset_path'])

#Evaluate the model
evaluate_model(validation_dataset_path = validation_data_final.outputs['output_dataset_path'],
               model = trained_model.output)
```

Compile and Run the training Pipeline

Compile and Run Pipeline

12m

```
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://chdprediction',
    parameter_values = {
        'training_dataset_path' : 'gs://chdprediction/CHD_train.csv',
        'validation_dataset_path' : 'gs://chdprediction/CHD_val.csv'
    },
    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/297382521725/locations/us-ce
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/297382521725/location
INFO:google.cloud.aiplatform.pipeline_jobs:View Pipeline Job:
```

Component Definition – 1

C1: Common dataset preparation (Converting Variable into categorical)

```
from kfp.v2.dsl import InputPath, OutputPath, Dataset

@component(packages_to_install = ["pandas","numpy","gcsfs","fsspec","scikit-learn"])

def dataprep_tocat(input_dataset_path: str, output_dataset_path: OutputPath('Dataset')):
    import pandas as pd
    import numpy as np

    CHD_df = pd.read_csv(input_dataset_path)
    #Converting features into categorical
    CHD_df['male'] = CHD_df['male'].astype('category')
    CHD_df['education'] = CHD_df['education'].astype('category')
    CHD_df['currentSmoker'] = CHD_df['currentSmoker'].astype('category')
    CHD_df['BPMeds'] = CHD_df['BPMeds'].astype('category')
    CHD_df['prevalentStroke'] = CHD_df['prevalentStroke'].astype('category')
    CHD_df['prevalentHyp'] = CHD_df['prevalentHyp'].astype('category')
    CHD_df['diabetes'] = CHD_df['diabetes'].astype('category')
    CHD_df['TenYearCHD'] = CHD_df['TenYearCHD'].astype('category')


    # Save the DataFrame to a CSV file with specified datatypes
    CHD_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 en
  return component_factory.create_component_from_func(
```

Component Definition – 2

C2: Treating Outliers Component - Clipping (Train set)

```
from kfp.v2.dsl import Output
from kfp.v2.dsl import Artifact
from kfp.v2.dsl import Model
from kfp.v2.dsl import InputPath, OutputPath
@component(packages_to_install=["pandas", "numpy", "scipy", "gcsfs"])
def clipping_train(training_dataset_path: InputPath('Dataset'),
                        output_dataset_path: OutputPath('Dataset'),
                        output_clipping_info: Output[Artifact]):
    import pandas as pd
    import numpy as np
    import json
    # Loading training dataset
    CHD_df = pd.read_csv(training_dataset_path)
    # List of numeric variables
    numeric_cols = ['age', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose', 'a1c', 'income','cigsPerDay'
    # Create a dictionary to store the winsorized values
    outlier_imputed_values = {}
    # Iterate over numeric variables
    for var in numeric_cols:
        # Calculate median and standard deviation
        median_value = CHD_df[var].median()
        std_value = CHD_df[var].std()
        # Calculate upper and lower thresholds
        upper_threshold = median_value + 3 * std_value
        lower_threshold = median_value - 3 * std_value
        # Clip outliers
        outlier_imputed_values[var] = {'upper_threshold': upper_threshold, 'lower_threshold': lower_threshold}
        CHD_df[var] = np.clip(CHD_df[var], lower_threshold, upper_threshold)
    # Save the clipped dataset
    CHD_df.to_csv(output_dataset_path, index=False)
    # Save the clipping values
    outlier_imputed_values_json = json.dumps(outlier_imputed_values)
    output_clipping_info.metadata['value'] = outlier_imputed_values_json
```

Component Definition – 3

C3: Treating Outliers Component - Clipping (Validation set)

```
from kfp.v2.dsl import Input
from kfp.v2.dsl import Model
from kfp.v2.dsl import InputPath, OutputPath

@component(packages_to_install = ["pandas","numpy","gcsfs"])
def clipping_val(val_dataset_path: InputPath('Dataset'),
                        output_dataset_path: OutputPath('Dataset'),
                        clipping_info: Input[Artifact]):
    import pandas as pd
    import numpy as np
    import json
    # Load Validation dataset
    val_df = pd.read_csv(val_dataset_path)
    # Load clipping values
    clipping_info_str = clipping_info.metadata['value']
    clipping_values = json.loads(clipping_info_str)
    # List of numeric variables
    numeric_cols = ['age', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose', 'a1c', 'income', 'cigsPerDay
    # Iterate over numeric variables
    for var in numeric_cols:
        # Extract clipping thresholds for the variable
        upper_threshold = clipping_values[var]['upper_threshold']
        lower_threshold = clipping_values[var]['lower_threshold']
        # Clip outliers in the validation data
        val_df[var] = np.clip(val_df[var], lower_threshold, upper_threshold)
    # Save the clipped validation dataset
    val_df.to_csv(output_dataset_path, index=False)
```

Component Definition – 4

C4: Treating Missing Values Component (Train set)

```
from kfp.v2.dsl import Output,InputPath, OutputPath, Artifact
@component(packages_to_install = ["pandas","numpy","gcsfs"])
def missing_values_train(training_dataset_path: InputPath('Dataset'), output_dataset_path: OutputPath('Dataset'), o
    import pandas as pd
    import numpy as np
    import json
    CHD_df = pd.read_csv(training_dataset_path)
    # Create a dictionary to store information at each step
    missing_values_info = {}
    # Step 1: Replace missing values in numeric variables with respective medians
    numeric_variables = ['totChol', 'BMI', 'glucose', 'a1c','heartRate']
    for var in numeric_variables:
       median_value = CHD_df[var].median()
       CHD_df[var].fillna(median_value, inplace=True)
    missing_values_info['numeric_variables_medians'] = {var: median_value for var, median_value in zip(numeric_vari
    # Step 2: Create a new category for missing values in the Education variable
    CHD_df['education'] = CHD_df['education'].astype('category')
    education_categories = CHD_df['education'].cat.categories.tolist()
    education_categories.append(5)  # Add 'Missing' category
    CHD_df['education'] = CHD_df['education'].cat.add_categories([5])
    CHD_df['education'].fillna(5, inplace=True)
    missing_values_info['education_missing_category'] = 5
    # Step 3: Impute missing values in BPMeds
    CHD_df['BPMeds'] = np.where((CHD_df['sysBP'] > 120) & (CHD_df['diaBP'] > 80), 1, 0)
    # Step 4: Treat CigsPerDay as continuous and impute with median value
    median_cigsPerDay = 0
    CHD_df['cigsPerDay'].fillna(median_cigsPerDay, inplace=True)
    missing_values_info['med_cigs'] = median_cigsPerDay
     # Save the modified dataset
    CHD_df.to_csv(output_dataset_path, index = False)
    # Save the missing values information
    missing_values_info_json = json.dumps(missing_values_info)
    output_missing_info.metadata['value'] = missing_values_info_json
```

Component Definition – 5

C5: Treating Missing Values Component (Validation set)

```
from kfp.v2.dsl import Input
from kfp.v2.dsl import Model
from kfp.v2.dsl import InputPath, OutputPath

@component(packages_to_install = ["pandas","numpy","gcsfs"])
def missing_values_val(val_dataset_path: InputPath('Dataset'),
                       output_dataset_path: OutputPath('Dataset'),
                       missing_info: Input[Artifact]):
  import pandas as pd
  import numpy as np
  import json
  # Load Validation dataset
  val_df = pd.read_csv(val_dataset_path)
  # Load missing values
  missing_info_str = missing_info.metadata['value']
  missing_values_info = json.loads(missing_info_str)
  numeric_variables = ['totChol', 'BMI', 'glucose', 'a1c','heartRate']
  # Step 1: Replace missing values in numeric variables with respective medians
  for var in numeric_variables:
     median_value = missing_values_info['numeric_variables_medians'][var]
     val_df[var].fillna(median_value, inplace=True)
  # Step 2: Create a new category for missing values in the Education variable
  val_df['education'] = val_df['education'].astype('category')
  val_df['education'] = val_df['education'].cat.add_categories([missing_values_info['education_missing_category']])
  val_df['education'].fillna(missing_values_info['education_missing_category'], inplace=True)
  # Step 3: Impute missing values in BPMeds
  val_df['BPMeds'] = np.where((val_df['sysBP'] > 120) & (val_df['diaBP'] > 80), 1, 0)
  # Step 4: Treat CigsPerDay as continuous and impute with median value
  val_df['cigsPerDay'].fillna(missing_values_info['med_cigs'], inplace=True)
  # Save the imputed dataset
  val_df.to_csv(output_dataset_path, index = False)
```

Component Definition – 6

C6: Common dataset processing(Applying log transform to solve skewness)

```
from kfp.v2.dsl import InputPath,OutputPath,Dataset

@component(packages_to_install = ["pandas","numpy","gcsfs"])

def skewness_processing(input_dataset_path: InputPath('Dataset'), output_dataset_path: OutputPath(Dataset)):
  import pandas as pd
  import numpy as np

  CHD_df = pd.read_csv(input_dataset_path)

  CHD_df['log_a1c'] = np.log(CHD_df['a1c'])

  CHD_df['log_glucose'] = np.log(CHD_df['glucose'])

  CHD_df['log_income'] = np.log(CHD_df['income'])


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

Component Definition – 7

C7: Normalizing numeric features (Train Dataset)

```
from kfp.v2.dsl import Output
from kfp.v2.dsl import Artifact
from kfp.v2.dsl import Model
from kfp.v2.dsl import InputPath, OutputPath
@component(packages_to_install = ["pandas"])
def normalize_training_dataset(training_dataset_path: InputPath('Dataset'),
                               output_dataset_path: OutputPath('Dataset'),
                               output_scaling_info: Output[Artifact]):
  import pandas as pd
  import numpy as np
  import json
  #Loading training dataset
  CHD_df = pd.read_csv(training_dataset_path)
  numeric_cols = ['age','log_income','totChol','sysBP','diaBP','BMI','heartRate','log_glucose','log_a1c','cigsPerDa
  min_max_values = {}
  # Normalize log-transformed columns and store min-max values
  for col in numeric_cols:
     # Calculate min and max values
     col_min = int(CHD_df[col].min())
     col_max = int(CHD_df[col].max())
     # Normalize column using the formula
     normalized_col = (CHD_df[col] - col_min) / (col_max - col_min)
     # Store min-max values in the dictionary
     min_max_values[col] = {'min': col_min, 'max': col_max}
    # Replace the original column with the normalized values
     CHD_df['norm_'+col] = normalized_col
  # Save normalized dataset
  CHD_df.to_csv(output_dataset_path, index = False)
  # Save min max values
  scaling_info_json = json.dumps(min_max_values)
  output_scaling_info.metadata['value'] = scaling_info_json
```

Component Definition – 8

C8: Normalizing numeric features (Validation set)

```
from kfp.v2.dsl import Input
from kfp.v2.dsl import Model
from kfp.v2.dsl import InputPath, OutputPath
@component(packages_to_install = ["pandas"])
def normalize_validation_dataset(val_dataset_path : InputPath('Dataset'),
                                 output_dataset_path : OutputPath('Dataset'),
                                 scaling_info_input: Input[Artifact]):
  import pandas as pd
  import numpy as np
  import json
  # Load Validation dataset
  val_df = pd.read_csv(val_dataset_path)
  # Load scaling info from training
  scaling_info_str = scaling_info_input.metadata['value']
  min_max_values = json.loads(scaling_info_str)
  # Normalizing the features
  numeric_cols = ['age','log_income','totChol','sysBP','diaBP','BMI','heartRate','log_glucose','log_a1c','cigsPerDa
  for col in numeric_cols:
    # Retrieve min-max values for the column
    col_min = min_max_values[col]['min']
    col_max = min_max_values[col]['max']
    # Normalize the column using the formula
    val_df['norm_'+col] = (val_df[col] - col_min) / (col_max - col_min)
  # Display the normalized test data
  print("Normalized Test Data:")
  print(val_df)
  # Save normalized dataset
  val_df.to_csv(output_dataset_path, index = False)
```

Component Definition – 9

C9: Common feature selection (Selecting features from decisions taken from EDA)

```
from kfp.v2.dsl import InputPath, OutputPath, Dataset

@component(packages_to_install = ["pandas","numpy","gcsfs"])

def feature_selection(input_dataset_path: InputPath('Dataset'), output_dataset_path:OutputPath(Dataset)):
  import pandas as pd
  import numpy as np

  CHD_df = pd.read_csv(input_dataset_path)
  CHD_df = CHD_df[['patientID','male','norm_age','education','BPMeds','norm_cigsPerDay',
                   'prevalentHyp','norm_log_income','norm_totChol','norm_sysBP','norm_BMI',
                   'norm_log_glucose','TenYearCHD']]

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

Component Definition – 10

C10: Random Over Sampling to treat class imbalance (Training Dataset)

```
from kfp.v2.dsl import InputPath, OutputPath, Dataset

@component(packages_to_install = ["pandas","numpy","scikit-learn","gcsfs", "imbalanced-learn == 0.11.0"])

def random_over_sampling(input_dataset_path: InputPath('Dataset'), output_dataset_path: OutputPath('Dataset')):
  import pandas as pd
  import numpy as np
  from imblearn.over_sampling import RandomOverSampler

  # Load the input dataset
  CHD_df = pd.read_csv(input_dataset_path)
  CHD_df['TenYearCHD'] = CHD_df['TenYearCHD'].astype('category')

  X = CHD_df.drop(['TenYearCHD','patientID'], axis=1)
  y = CHD_df['TenYearCHD']

  # Random Over Sampling
  oversampler = RandomOverSampler()
  X_train_resampled, y_train_resampled = oversampler.fit_resample(X, y)

  X_train_df = pd.DataFrame(X_train_resampled, columns = X.columns)
  y_train_df = pd.DataFrame(y_train_resampled, columns = ['TenYearCHD'])

  # Concatenate the features to a dataframe
  oversampled_df = pd.concat([X_train_df, y_train_df], axis = 1)

  # Save the re-joined, oversampled dataset
  oversampled_df.to_csv(output_dataset_path, index = False)
```

Component Definition – 11

C11: Train an ensemble classifier

```
@component(packages_to_install = ["pandas","scikit-learn","joblib","xgboost == 1.5.0"])

def train_ensemble_classifier(training_dataset_path: InputPath('Dataset'),Output_model: Output[Model]):
  import pandas as pd
  import numpy as np
  from sklearn.ensemble import GradientBoostingClassifier
  from sklearn.linear_model import LogisticRegression
  from sklearn.ensemble import RandomForestClassifier
  from sklearn.linear_model import Perceptron
  from sklearn.neighbors import KNeighborsClassifier
  from sklearn.ensemble import VotingClassifier
  from xgboost import XGBClassifier
  import joblib
  # Load training dataset
  CHD_df = pd.read_csv(training_dataset_path)
  X_train = CHD_df.drop('TenYearCHD', axis=1)
  y_train = CHD_df['TenYearCHD']
  # Instantiate Classifiers
  knn_classifier = KNeighborsClassifier(algorithm ='auto', n_neighbors = 3, weights = 'distance')
  rf_classifier = RandomForestClassifier(max_depth=12 ,n_estimators =200, random_state=42)
  logistic_classifier = LogisticRegression()
  gb_classifier = GradientBoostingClassifier()
  #  Create Ensemble Classifier
  ensemble_classifier = VotingClassifier(estimators=[
    ('knn', knn_classifier),
    ('rf', rf_classifier),
    ('logistic', logistic_classifier),
    ('gb', gb_classifier)
   ], voting='hard')  # Use 'hard' for majority voting
  #Fit the Ensemble Classifier
  ensemble_classifier.fit(X_train, y_train)
  # Save the model
  joblib.dump(ensemble_classifier, Output_model.path)
```

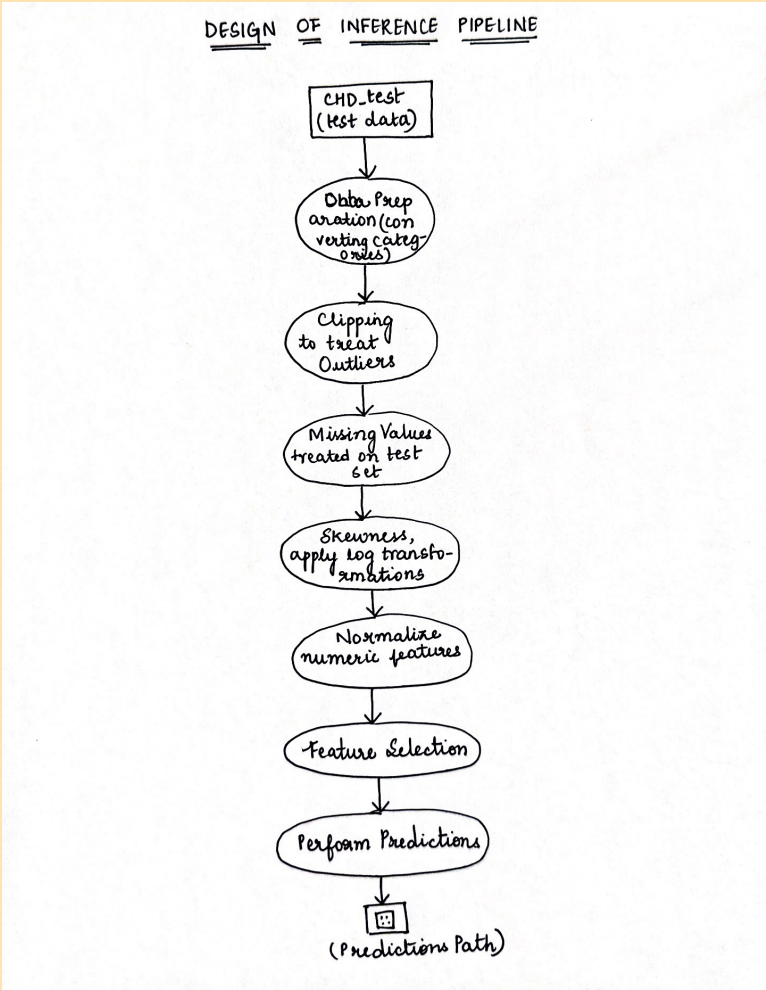
Component Definition – 12

C12: Evaluate Model

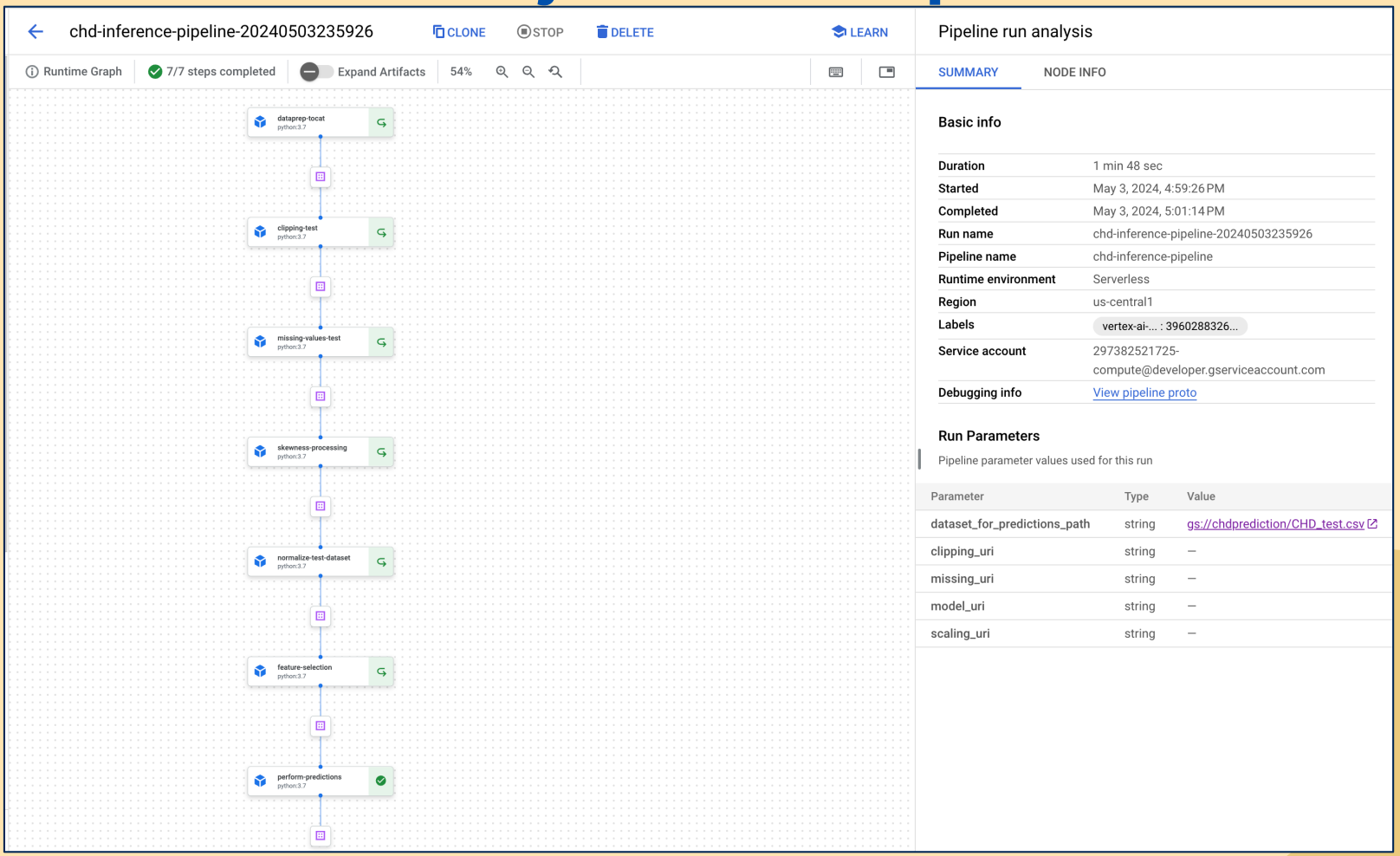
```
from kfp.v2.dsl import Metrics
@component(packages_to_install = ["pandas","scikit-learn","joblib"])
def evaluate_model(validation_dataset_path: InputPath('Dataset'),
                   model: Input[Model],
                   metrics: Output[Metrics]):
  import pandas as pd
  import numpy as np
  from sklearn.metrics import roc_auc_score
  from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
  from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score,average_precision_score
  import joblib
  #Load the validation dataset
  val_df = pd.read_csv(validation_dataset_path)
  X_val = val_df.drop(['patientID','TenYearCHD'], axis =1)
  y_val = val_df['TenYearCHD']
  # Load the model
  model = joblib.load(model.path)
  # Make predictions
  y_pred = model.predict(X_val)
  # Evaluate Model(Calculate metrics)
  report = classification_report(y_val, y_pred)
  accuracy = accuracy_score(y_val, y_pred)
  f1_score = f1_score(y_val, y_pred, average = 'weighted')
  roc_auc = roc_auc_score(y_val, y_pred)
  auc_pr = average_precision_score(y_val, y_pred)
  # Output Metrics
  metrics.log_metric('classification report',report)
  metrics.log_metric('accuracy',accuracy)
  metrics.log_metric('ROC AUC',roc_auc)
  metrics.log_metric('AUC PR', auc_pr)
  metrics.log_metric('F1 Score', f1_score)
```

Inference Pipeline

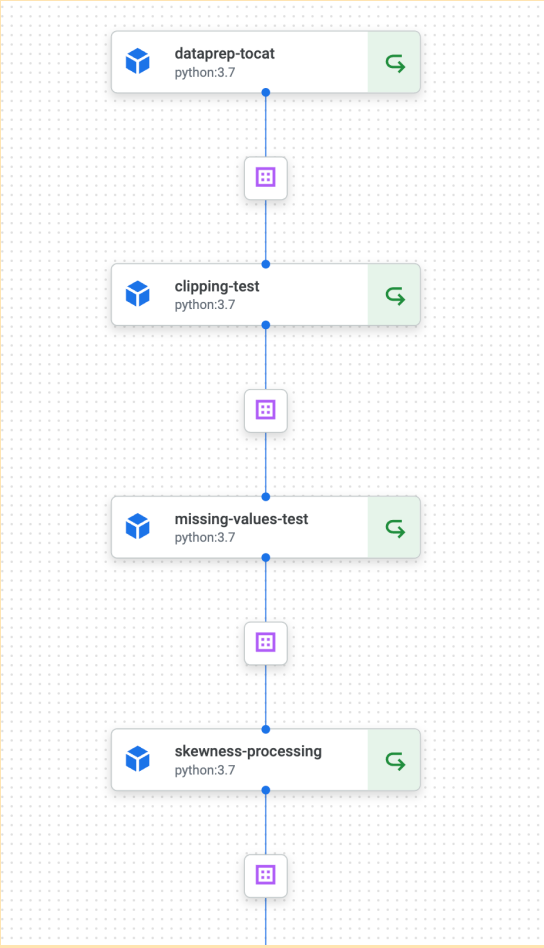
Design- Inference Pipeline

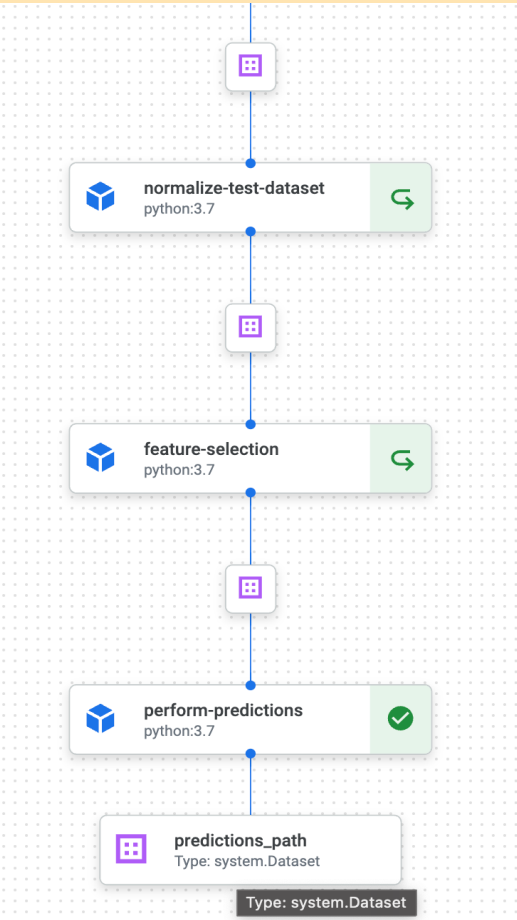


Overall Inference Pipeline

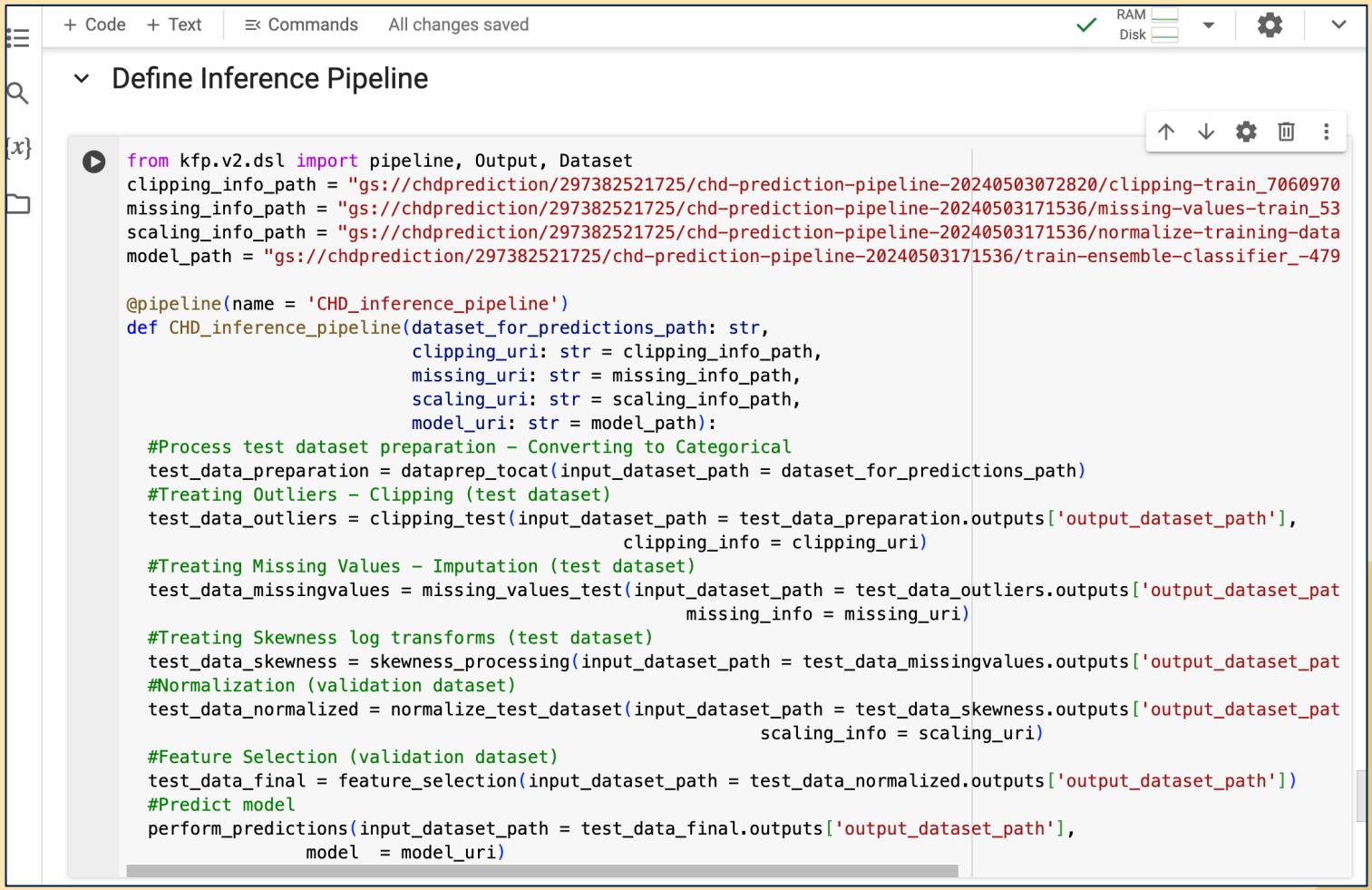


Inference Pipeline Visualization





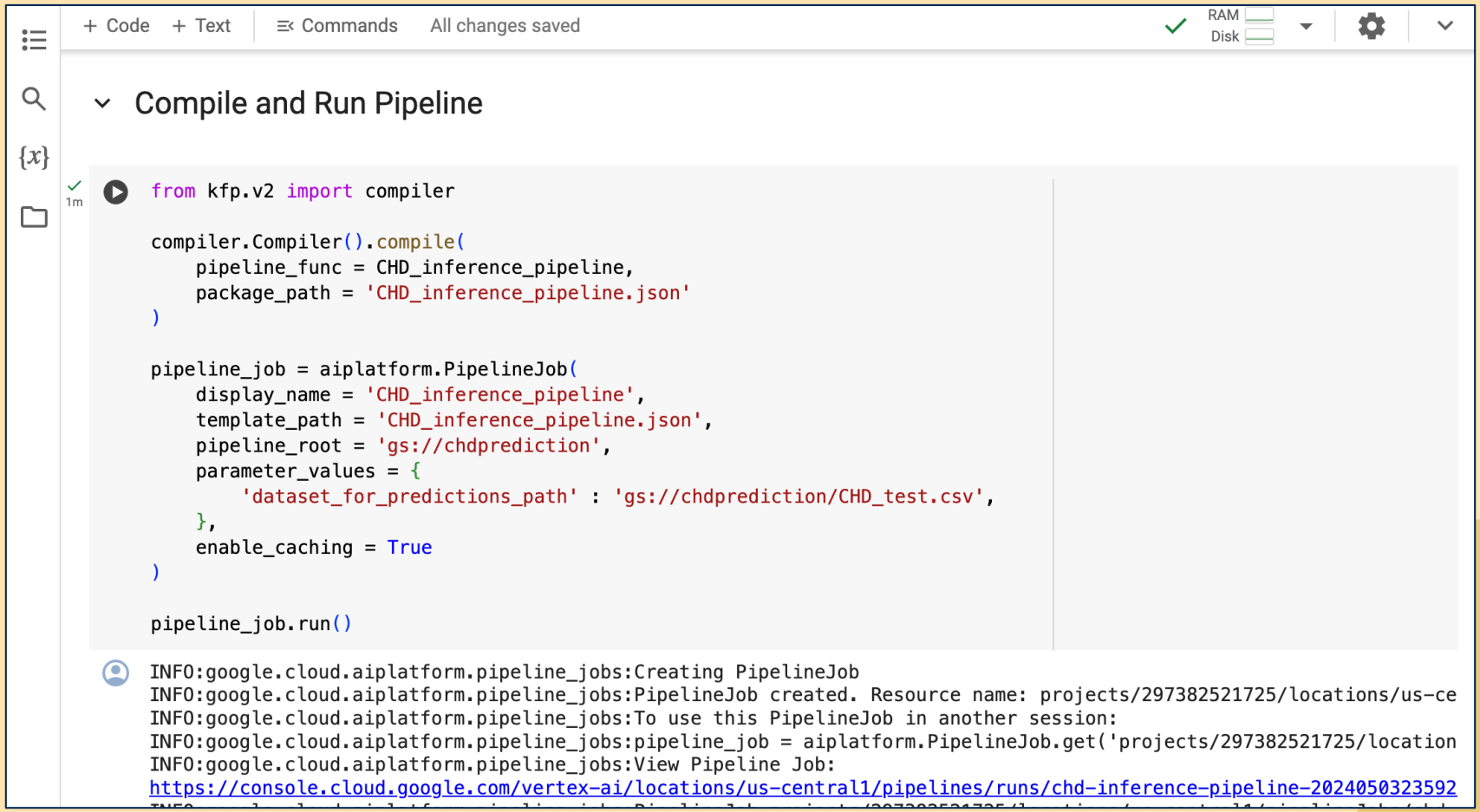
Inference Pipeline Definition



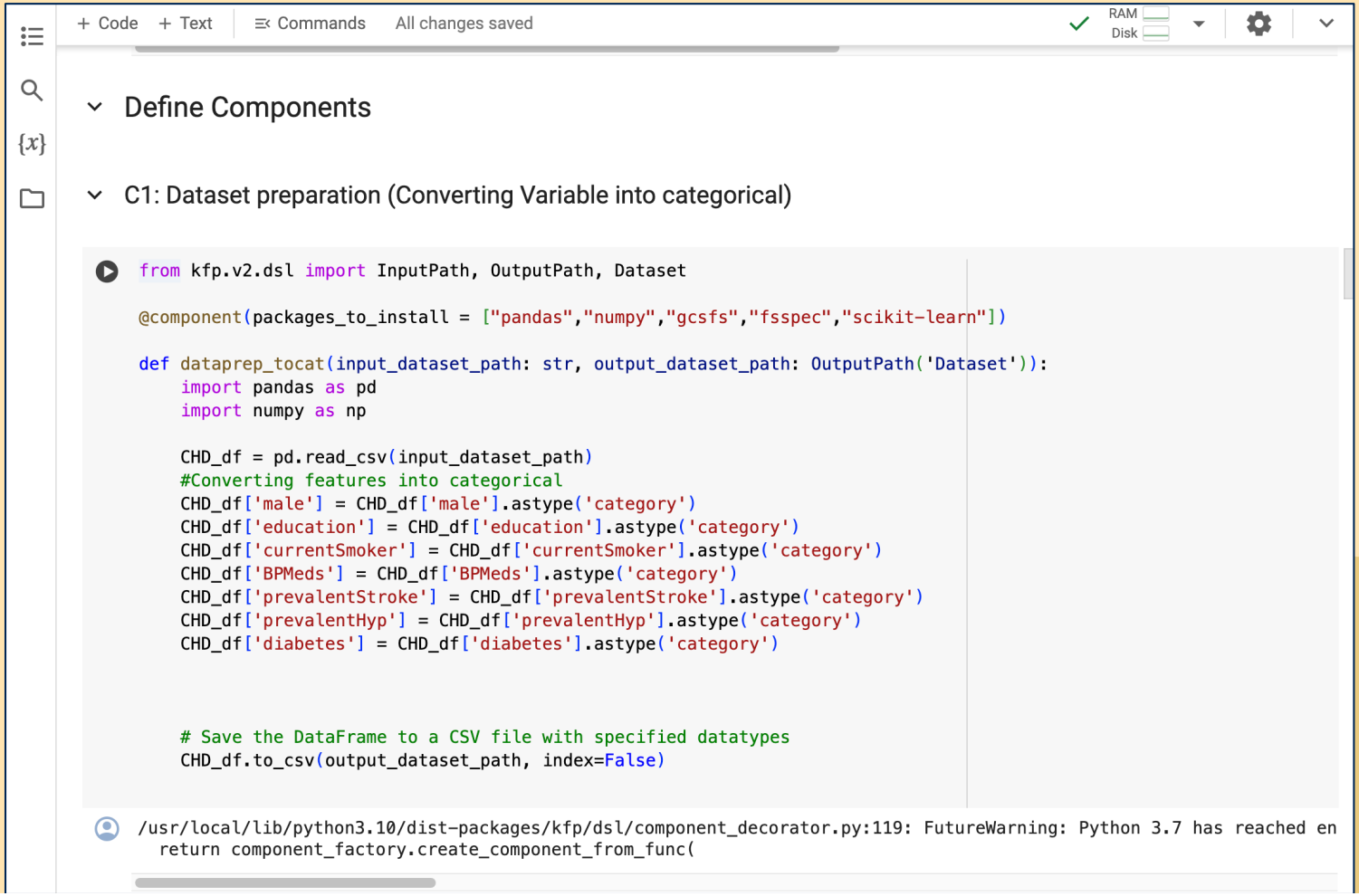

```
from kfp.v2.dsl import pipeline, Output, Dataset
clipping_info_path = "gs://chdprediction/297382521725/chd-prediction-pipeline-20240503072820/clipping-train_7060970
missing_info_path = "gs://chdprediction/297382521725/chd-prediction-pipeline-20240503171536/missing-values-train_53
scaling_info_path = "gs://chdprediction/297382521725/chd-prediction-pipeline-20240503171536/normalize-training-data
model_path = "gs://chdprediction/297382521725/chd-prediction-pipeline-20240503171536/train-ensemble-classifier_-479

@pipeline(name = 'CHD_inference_pipeline')
def CHD_inference_pipeline(dataset_for_predictions_path: str,
                           clipping_uri: str = clipping_info_path,
                           missing_uri: str = missing_info_path,
                           scaling_uri: str = scaling_info_path,
                           model_uri: str = model_path):
  #Process test dataset preparation - Converting to Categorical
  test_data_preparation = dataprep_tocat(input_dataset_path = dataset_for_predictions_path)
  #Treating Outliers - Clipping (test dataset)
  test_data_outliers = clipping_test(input_dataset_path = test_data_preparation.outputs['output_dataset_path'],
                                     clipping_info = clipping_uri)
  #Treating Missing Values - Imputation (test dataset)
  test_data_missingvalues = missing_values_test(input_dataset_path = test_data_outliers.outputs['output_dataset_pat
                                                missing_info = missing_uri)
  #Treating Skewness log transforms (test dataset)
  test_data_skewness = skewness_processing(input_dataset_path = test_data_missingvalues.outputs['output_dataset_pat
  #Normalization (validation dataset)
  test_data_normalized = normalize_test_dataset(input_dataset_path = test_data_skewness.outputs['output_dataset_pat
                                                scaling_info = scaling_uri)
  #Feature Selection (validation dataset)
  test_data_final = feature_selection(input_dataset_path = test_data_normalized.outputs['output_dataset_path'])
  #Predict model
  perform_predictions(input_dataset_path = test_data_final.outputs['output_dataset_path'],
                      model  = model_uri)
```

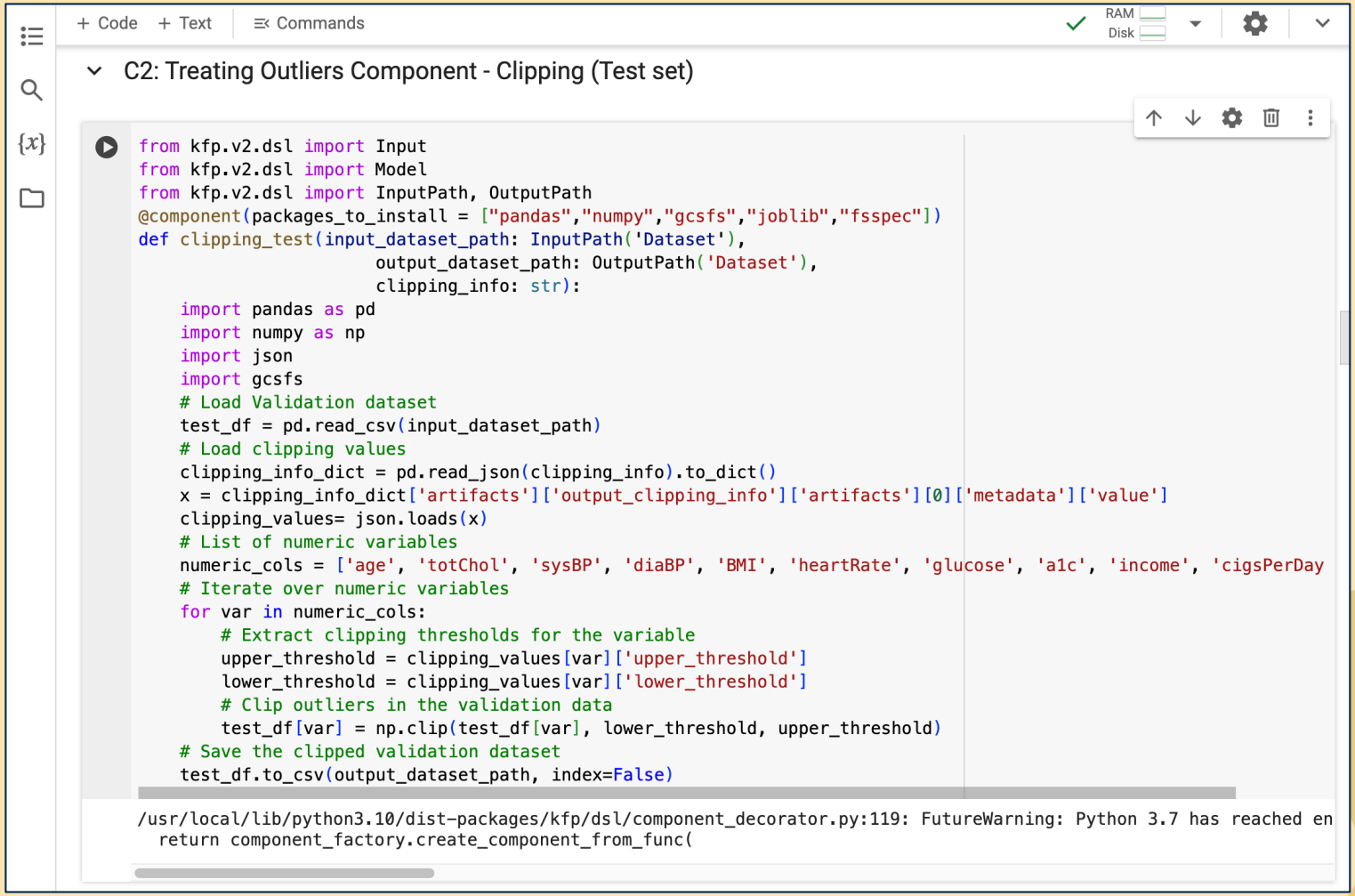
Compile and Run the Inference Pipeline



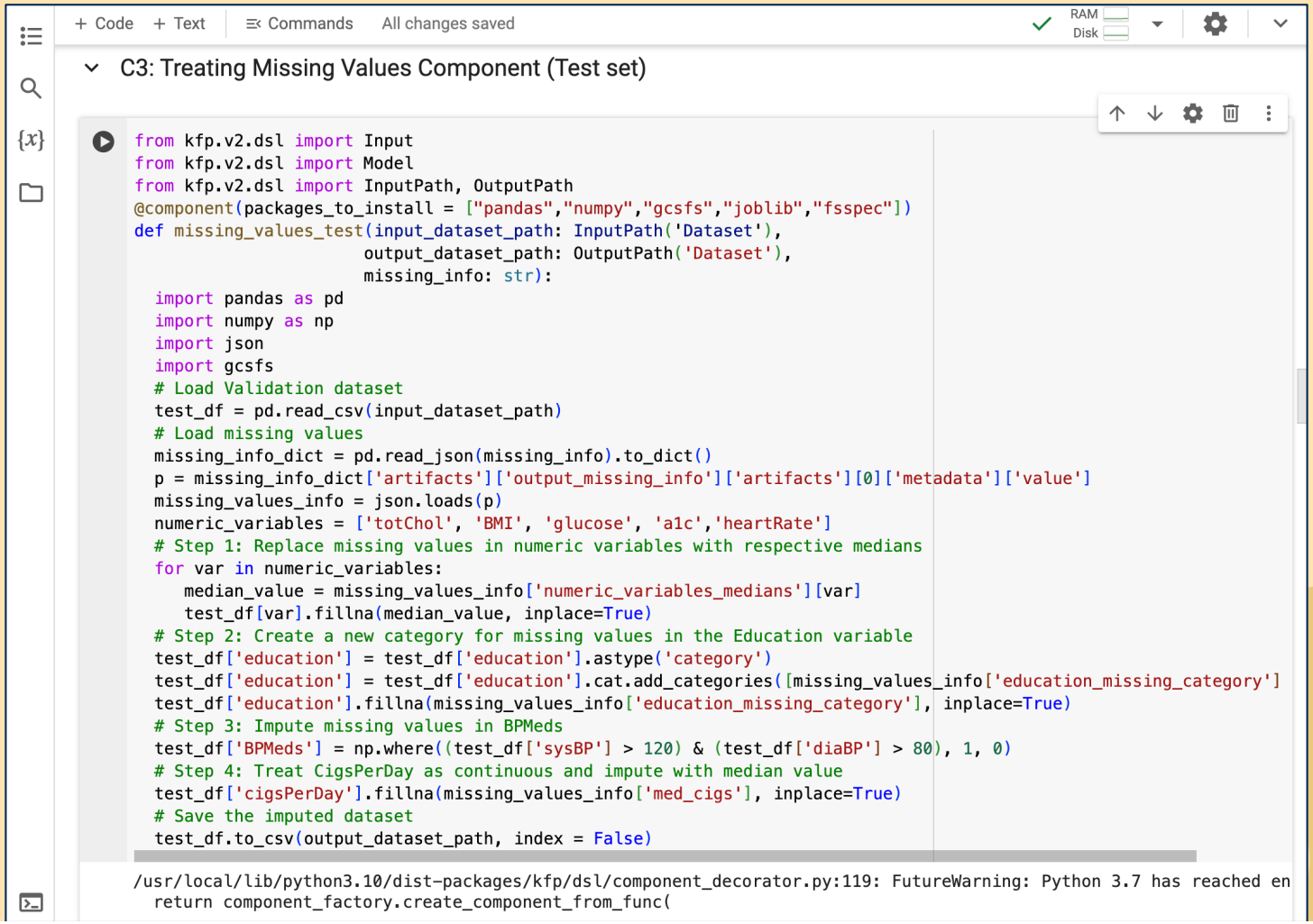
Component Definition –1



Component Definition -2

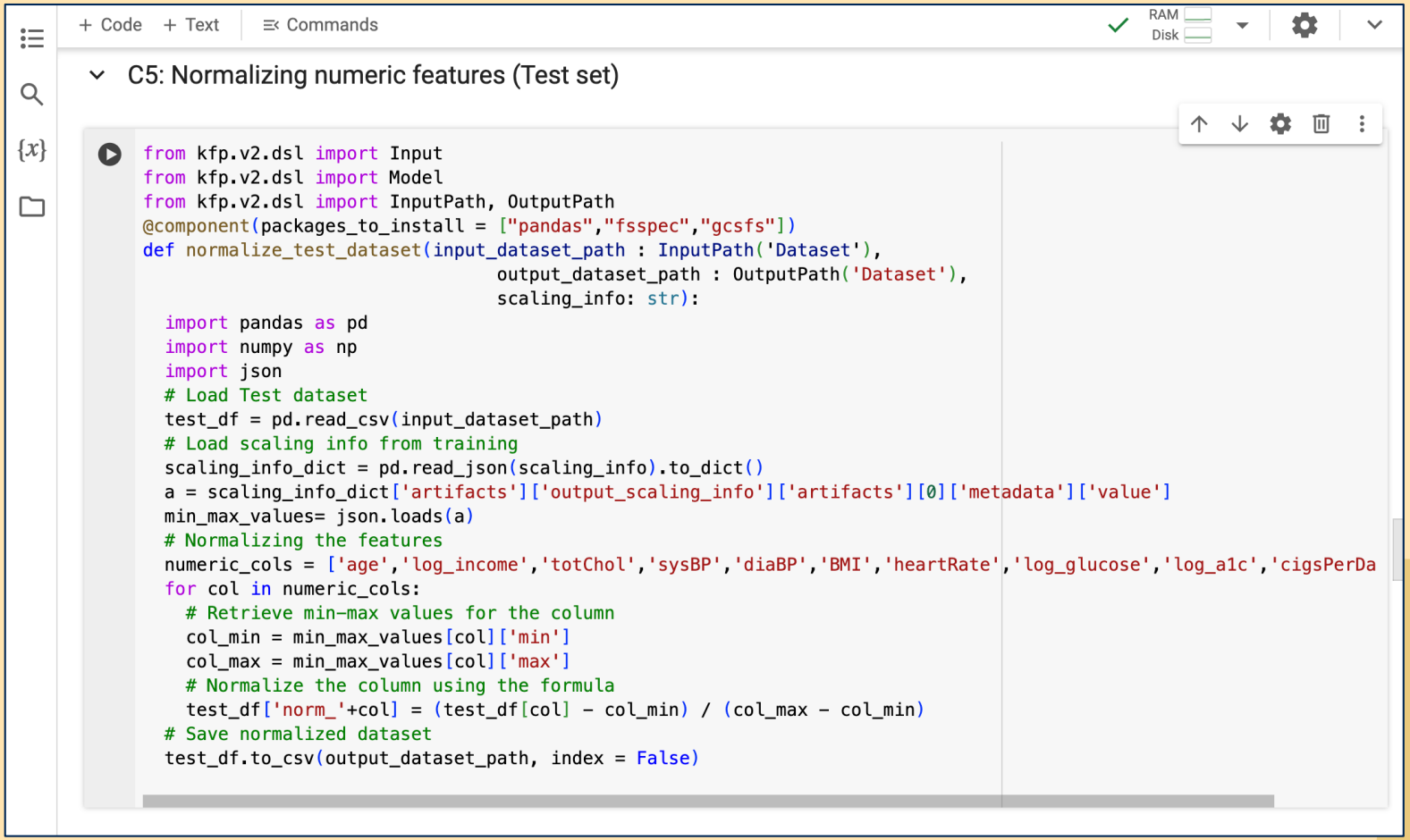


Component Definition -3



Component Definition -4

Component Definition –5



Component Definition -6

Component Definition –7

C7: Predict Model

```
@component(packages_to_install = ["pandas","scikit-learn","joblib","gcsfs","fsspec","numpy"])

def perform_predictions(input_dataset_path: InputPath('Dataset'),
                        model: str,
                        predictions_path: OutputPath('Dataset')):
  import pandas as pd
  import numpy as np
  import joblib
  import gcsfs
  #Load the validation dataset
  test_df = pd.read_csv(input_dataset_path)
  X_test = test_df.drop('patientID', axis =1)

  # Load the model
  fs = gcsfs.GCSFileSystem()

  with fs.open(model,'rb') as f:
    trained_model = joblib.load(f)

  # Make predictions
  y_pred = trained_model.predict(X_test)
  test_df['pred_CHD'] = y_pred
  test_df = test_df[['patientID','pred_CHD']]

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

Inference Pipeline
Screenshot of Predictions (csv file)

CHD_preds

patientID	pred_CHD
110399	0
189047	0
957019	1
208967	0
230935	0
216024	1
368834	1
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

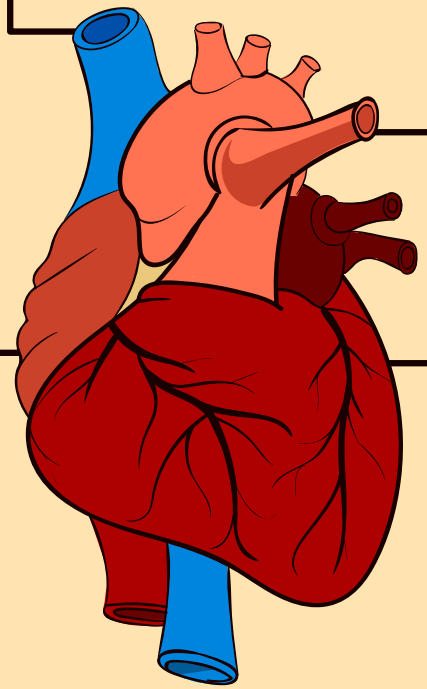
Summary

Feature Engineering

Log transformations on skewed variables and normalization scaling of numeric variables improved the model performance. But new variables like the MAP (calculated from sysBP, diaBP) did not show any improvement in the model.

Feature Selection

Techniques like LASSO CV and other variable importance algorithms helped select the variables that were important in getting the predictions. Also some variables that were highly correlated were dropped.



Random Over Sampling

Random Over Sampling gave a better f1 score compared to the SMOTE and hence was used in the final model. This technique was used to solve the high imbalance in the response variable.

Model Development

Different Classifiers were tried, but finally an ensemble model made up of 4 classifiers gave the best performance and was used in the pipeline.

Summary Discussion

1.Data Preparation:

- Dataset split into 70% training and 30% validation sets to prevent data leakage.
- Exploratory Data Analysis (EDA) conducted on the training set to identify and address data quality issues such as missing values, outliers, skewness, and inappropriate data types.

2.Data Processing Pipeline:

- Implemented a data processing pipeline to handle missing values, outliers, skewness, and scale variations in numeric features, following decisions outlined in the data preparation plan.
- Ensured consistency by applying the same pre-processing steps to the validation and test sets to prevent data leakage.

3.Feature Selection and Engineering:

- Utilized LASSO CV and SHAP algorithms to identify important features for model training.
- Conducted feature engineering techniques such as log-transforms and normalization to improve model performance.
- Addressed multicollinearity by dropping highly correlated variables.

4.Handling Class Imbalance:

- Applied Random Over Sampling to address class imbalance in the training dataset.

Summary Discussion

5.**Model Selection and Hyperparameter Tuning:**

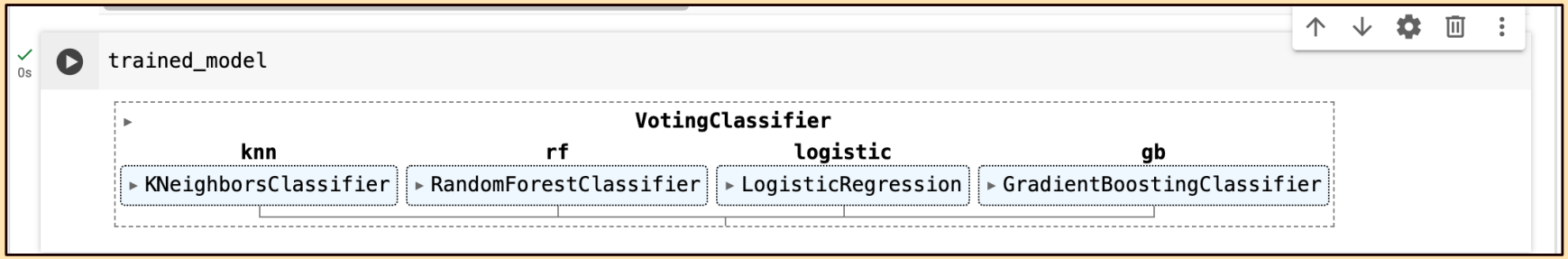
- Explored different classification models and fine-tuned them using Grid Search Cross Validation to optimize hyperparameters.
- Constructed an ensemble classifier using the best-performing models as shown below, resulting in improved performance.

6.**Model Evaluation:**

- Evaluated the ensemble classifier on the validation dataset, achieving an F1-score of **0.8033**.

7.**Inference Pipeline:**

- Implemented an inference pipeline to generate predictions on the test dataset using the trained ensemble classifier.



F1-Score Summary for different classifiers

Model	Hyperparameters	F1 Score
KNearestNeighbors	n_neighbors = 3, weights ='distance', algorithm = 'auto'	0.72
Random Forest Classifier	max_depth = 12, n_estimators = 200	0.71
Logistic Regression	Default threshold = 0.5	0.71
Gradient Boosting	learning_rate = 0.1, n_estimators = 100, max_depth =3	0.73

Thank You!