George Washington University

Heart Disease Risk Analysis and Visualization using Python and Dash

Sagar Sanjay Shah
GWID: G26436634

DATS6401: Visualization of Complex Data

Prof. Reza Jafari

04/30/2025

Dash Link: https://dashapp-605581485690.southamerica-west1.run.app/

Table of Contents

List of Figures and Tables

Abstract

Final Term Project (FTP) emphasizes the application of cutting-edge Python-based data visualization combined with interactive dashboard design to investigate, analyze, and portray risk indicators of heart disease in detail. The project, developed from a real-world dataset made available by the Centers for Disease Control and Prevention (CDC), applies a series of static and dynamic data visualization methods to reveal hidden patterns, correlations, and outliers within the data. Heavy preprocessing to handle missing values, outliers, and normalization was performed to preprocess the data before detailed statistical analysis. Principal Component Analysis (PCA) was used to perform dimension reduction for enhancing interpretability and computational efficiency.

A web-based dashboard was developed using the Dash framework, providing an interactive and intuitive interface for users to explore different aspects of the dataset dynamically without re-running code. The dashboard was successfully deployed on Google Cloud Platform (GCP) for public access. This report comprehensively documents the entire workflow from data cleaning and exploration to dashboard development and deployment showing how effective visualization techniques and interactive apps can contribute to data-driven decision-making in healthcare.

Introduction

Heart disease is one of the most significant public health problems in the world and remains the leading cause of death among men and women in the United States. With millions of individuals being impacted annually, the ability to effectively review and predict the determinants of heart disease is essential to reducing morbidity and mortality rates. The intricacy of heart-related diseases is due to the broad spectrum of causative factors, such as lifestyle habits, genetic dispositions, and socioeconomic status.

This project is focused on the examination of a real data set from the Centers for Disease Control and Prevention (CDC) Behavioral Risk Factor Surveillance System (BRFSS). The data set comprises over 400,000 responses across a range of health indicators including physical and mental health days, cigarette smoking and alcohol use, sleep, BMI, and self-reported general health. These features offer a solid foundation for examining trends, identifying risk factors, and inferring information about population-level heart health.

The final purpose of the project is to implement Python data visualization techniques and interactive dashboard development for analyzing the data and extracting respective patterns of heart disease. Preprocessing tasks of data cleaning, missing value, and outliers' elimination form the initiation of the project. Normal testing and transformation along with dimension reduction through Principal Component Analysis (PCA) are done subsequently. A variety of static visualizations are utilized to explore variable relationships, including boxplots, dist-plots, strip plots, violin plots, heatmaps, 3D plots, and more. These visualizations are not only exploration tools but also narrative tools that help in translating raw data into actionable information.

Dataset Description

The dataset used in this project is derived from the Behavioral Risk Factor Surveillance System (BRFSS), a nationwide health-related telephone survey system managed by the Centers for Disease Control and Prevention (CDC). Initiated in 1984 with participation from only 15 states, the BRFSS has grown significantly and now collects data from all 50 U.S. states, the District of Columbia, and three U.S. territories, making it the largest continuously conducted health survey system in the world.

The BRFSS conducts over 400,000 interviews annually, capturing detailed information about U.S. residents' health-related risk behaviors, chronic health conditions, and use of preventive services. This vast and diverse dataset serves as a cornerstone for epidemiological research and policy planning in public health. In the context of heart disease, a leading cause of death and disability in the U.S. this dataset provides a valuable opportunity to understand the influence of various behavioral and demographic factors on cardiovascular health outcomes.

The dataset includes numerous self-reported variables related to general health perception, physical and mental health days, lifestyle habits (such as smoking, alcohol consumption, and physical activity), as well as biometric indicators like BMI, height, weight, and sleep hours. By applying data analytics and visualization techniques to this rich dataset, we can uncover meaningful patterns and correlations that may help predict the likelihood of heart disease and assist in developing targeted interventions.

Variables in the Dataset

- **Dependent Variable**:
  - Had Heart Attack: Indicates whether an individual has suffered a heart attack (binary: Yes/No).

- **Independent Variables** (subset):
  - State: U.S. state of residence
  - Sex: Gender (Male/Female)
  - Age Category: Categorical age groups
  - General Health: Self-reported general health status
  - Physical Health Days: Number of days physical health was not good in past 30 days
  - Mental Health Days: Number of days mental health was not good in past 30 days
  - Physical Activities: Engagement in physical exercise
  - Sleep Hours: Average number of hours of sleep per night
  - Smoker Status: Current smoking status
  - E-Cigarette Usage: Whether the individual uses e-cigarettes
  - Alcohol Drinkers: Whether the individual consumes alcohol
  - Height In Meters, Weight in Kilograms, BMI: Biometric indicators
  - Race Ethnicity Category: Race or ethnicity classification

Data Preprocessing

1. Before Cleaning

Snapshot Before Cleaning:

HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealth	SleepTime	Asthma	KidneyDisease	SkinCancer
No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Yes	Very good	5.0	Yes	No	Yes
No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Yes	Very good	7.0	No	No	No
No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Yes	Fair	8.0	Yes	No	No
No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	No	Good	6.0	No	No	Yes
No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Yes	Very good	8.0	No	No	No
Yes	28.87	Yes	No	No	6.0	0.0	Yes	Female	75-79	Black	No	No	Fair	12.0	No	No	No
No	21.63	No	No	No	15.0	0.0	No	Female	70-74	White	No	Yes	Fair	4.0	Yes	No	Yes
No	31.64	Yes	No	No	5.0	0.0	Yes	Female	80 or older	White	Yes	No	Good	9.0	Yes	No	No
No	26.45	No	No	No	0.0	0.0	No	Female	80 or older	White	No, borderline diabetes	No	Fair	5.0	No	Yes	No
No	40.69	No	No	No	0.0	0.0	Yes	Male	65-69	White	No	Yes	Good	10.0	No	No	No

No missing values found in the dataset

(Fig 1. Before Cleaning Console Output)

The first photo is the raw image of the dataset prior to cleaning. As seen, the dataset has categorical and numerical variables relating to health indicators such as Heart Disease, BMI, Smoking, Alcohol Drinking, Stroke, Physical Health, Mental Health, Diff Walking, Sleep Time, and others.

Observation:

- Variables are held in heterogeneous data types: numerical values (e.g., BMI, Sleep Time), binary flags (e.g., Smoking, Stroke), and category strings (e.g., Sex, Age Category, Race).
- Most entries have extreme values, i.e., 30 days of poor mental or physical health, which will influence distribution.
- Fortunately, the message confirms that there were no missing values found in the dataset, and imputation was not required.

2. After Cleaning

Snapshot After Cleaning:

HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealth	SleepTime	Asthma	KidneyDisease	SkinCancer
0	16.60	1	0	0	3.0	30.0	0	Female	55-59	White	Yes	1	Very good	5.0	1	0	1
0	20.34	0	0	1	0.0	0.0	0	Female	80 or older	White	No	1	Very good	7.0	0	0	0
0	26.58	1	0	0	20.0	30.0	0	Male	65-69	White	Yes	1	Fair	8.0	1	0	0
0	24.21	0	0	0	0.0	0.0	0	Female	75-79	White	No	0	Good	6.0	0	0	1
0	23.71	0	0	0	28.0	0.0	1	Female	40-44	White	No	1	Very good	8.0	0	0	0
1	28.87	1	0	0	6.0	0.0	1	Female	75-79	Black	No	0	Fair	12.0	0	0	0
0	21.63	0	0	0	15.0	0.0	0	Female	70-74	White	No	1	Fair	4.0	1	0	1
0	31.64	1	0	0	5.0	0.0	1	Female	80 or older	White	Yes	0	Good	9.0	1	0	0
0	26.45	0	0	0	0.0	0.0	0	Female	80 or older	White	No, borderline diabetes	0	Fair	5.0	0	1	0
0	40.69	0	0	0	0.0	0.0	1	Male	65-69	White	No	1	Good	10.0	0	0	0

(Fig 2. After Cleaning Console Output)

After confirming the absence of missing values, the next step was to remove duplicate records to eliminate bias and over-representation in analysis. The system identified and removed 18,078 duplicate rows, leaving a cleaned dataset of 301,717 records within 18 features.

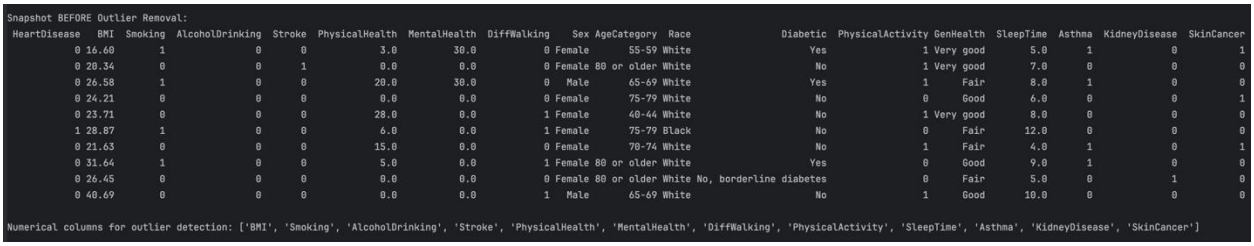
Sanitized snapshot illustrating the mapping of categorical values into binary flags (Smoking now shown as 0/1) to keep up with numerical processing and modeling.

Observation:

- The purified dataset appears more structured and well-formatted.
- Every feature is coded uniformly, preparing the data for machine learning models and statistical analysis.
- Reduction from ~320k to ~301k rows indicates significant redundancy in the raw dataset, possibly from duplicate responses or survey mistakes.

Outlier Detection and Removal

3. Before Outlier Removal



Snapshot BEFORE Outlier Removal:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealth	SleepTime	Asthma	KidneyDisease	SkinCancer
0	0	16.60	1	0	0	3.0	30.0	0	Female	55-59	White	Yes	1	Very good	5.0	1	0	1
1	0	20.34	0	0	1	0.0	0.0	0	Female	80 or older	White	No	1	Very good	7.0	0	0	0
2	0	26.58	1	0	0	20.0	30.0	0	Male	65-69	White	Yes	1	Fair	8.0	1	0	0
3	0	24.21	0	0	0	0.0	0.0	0	Female	75-79	White	No	0	Good	6.0	0	0	1
4	0	23.71	0	0	0	28.0	0.0	1	Female	40-44	White	No	1	Very good	8.0	0	0	0
5	1	28.87	1	0	0	6.0	0.0	1	Female	75-79	Black	No	0	Fair	12.0	0	0	0
6	0	21.63	0	0	0	15.0	0.0	0	Female	70-74	White	No	1	Fair	4.0	1	0	1
7	0	31.64	1	0	0	5.0	0.0	1	Female	80 or older	White	Yes	0	Good	9.0	1	0	0
8	0	26.45	0	0	0	0.0	0.0	0	Female	80 or older	White	No, borderline diabetes	0	Fair	5.0	0	1	0
9	0	40.69	0	0	0	0.0	0.0	1	Male	65-69	White	No	1	Good	10.0	0	0	0

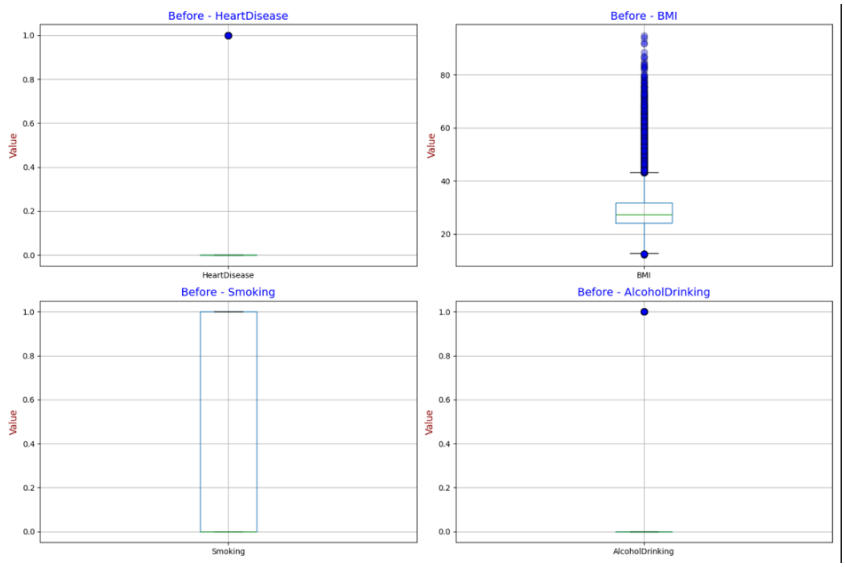
Numerical columns for outlier detection: ['BMI', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'PhysicalActivity', 'SleepTime', 'Asthma', 'KidneyDisease', 'SkinCancer']

(Fig 3. Before Outlier Removal Console Output)

The pre-outlier removal snapshot is the dataset in cleaned numeric format but still containing all the extreme values. The variables BMI, Physical Health, Mental Health, and Sleep Time contain potential outliers that can distort mean-based statistics and modeling results.

Observation:

- Only a few respondents report 28+ days of sickness or 12+ hours of sleep daily, which - while conceivable, are statistical exceptions.
- The data set also includes features such as Skin Cancer, Kidney Disease, Asthma, and Diabetic, which are inherently binary but may also have low variance or be imbalanced.
- The listed "numerical columns for outlier detection" are appropriate and comprehensive for boxplot or IQR-based screening.



(Fig 4. Boxplot Before Outlier Removal)

- This figure presents four individual boxplots for Heart Disease, BMI, Smoking, and Alcohol Drinking before any outlier filtering was performed.

4. After Outlier Removal – Outlier Summary

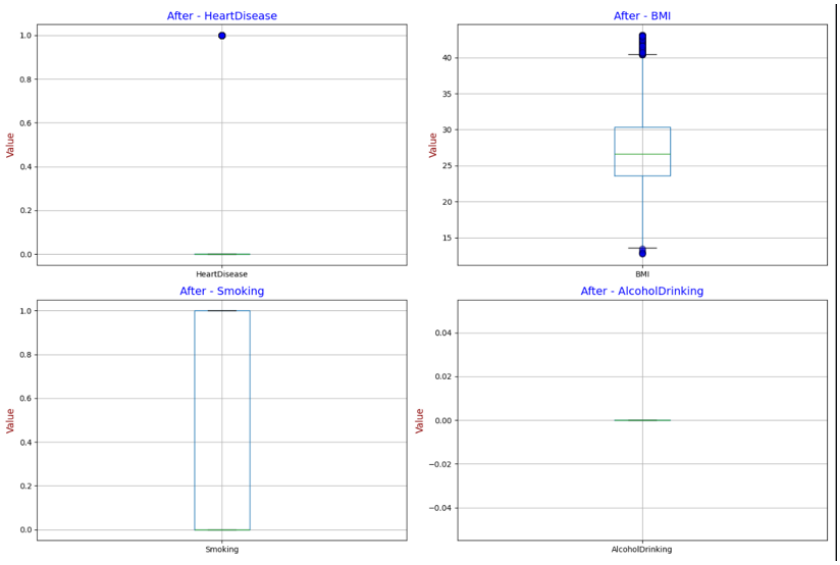
```
Final shape after outlier removal: (120364, 18)

Snapshot AFTER Outlier Removal:
 HeartDisease   BMI  Smoking  AlcoholDrinking  Stroke  PhysicalHealth  MentalHealth  DiffWalking     Sex AgeCategory  Race Diabetic  PhysicalActivity GenHealth  SleepTime  Asthma  KidneyDisease  SkinCancer
            0 28.71        1                0       0             0.0           0.0            0  Female       55-59 White       No                 1 Very good        5.0       0              0           0
            0 29.18        0                0       0             1.0           0.0            0  Female       50-54 White       No                 1 Very good        6.0       0              0           0
            0 31.66        1                0       0             5.0           0.0            0    Male       60-64 White       No                 1 Very good        5.0       0              0           0
            0 24.89        0                0       0             1.0           0.0            0  Female       55-59 White       No                 1 Very good        7.0       0              0           0
            0 25.84        1                0       0             5.0           0.0            0    Male       70-74 Black       No                 1      Good        8.0       0              0           0
            0 19.02        1                0       0             0.0           5.0            0  Female       60-64 White       No                 1 Very good        9.0       0              0           0
            0 24.62        0                0       0             5.0           0.0            0  Female 80 or older White       No                 1      Good        6.0       0              0           0
            0 28.13        0                0       0             0.0           0.0            0    Male       60-64 White       No                 1 Excellent        8.0       0              0           0
            0 33.23        0                0       0             0.0           0.0            0    Male       65-69 White      Yes                 1 Very good        8.0       0              0           0
            0 25.11        0                0       0             5.0           5.0            0  Female       65-69 Black       No                 1      Good        7.0       0              0           0
```

(Fig 5. After Outlier Removal Console Output)

The outlier removed snapshot shows the dataset cleaned, having discarded statistically outlying values based on the Interquartile Range (IQR) method. Columns such as BMI, Physical Health, Mental Health, and Sleep Time now have more realistic and meaningful boundaries, leading to a statistically cleaner and model-friendlier dataset.

Observation:

- The BMI histogram has become increasingly more highly centralized, without the top outliers (the most extreme obesity cases) and therefore much easier to trace trends.
- Duration health variables like Physical Health and Mental Health are now within realistic and stipulated bounds, improving summary statistics and graphical displays.
- Binary features (i.e., Smoking, Alcohol Drinking) are largely as expected but are known to have no incorrect or invalid values after filtering.
- The dataset is now minimized from more than 300,000 row to about 120,364, representing the large number of outliers eliminated from many features.



(Fig 6. Boxplot After Outlier Removal)

- This figure shows the same four variables (Heart Disease, BMI, Smoking, and Alcohol Drinking) after outlier filtering was applied.

5. Key Highlights:

Outlier Summary:

Column Name	Outliers Count	Lower Bound	Upper Bound
BMI	8905	12.6	43.08
Smoking	0	-1.5	2.5
AlcoholDrinking	21581	0.0	0.0
Stroke	12064	0.0	0.0
PhysicalHealth	47136	-3.0	5.0
MentalHealth	39713	-6.0	10.0
DiffWalking	44355	0.0	0.0
PhysicalActivity	71305	1.0	1.0
SleepTime	4542	3.0	11.0
Asthma	42651	0.0	0.0
KidneyDisease	11776	0.0	0.0
SkinCancer	29292	0.0	0.0

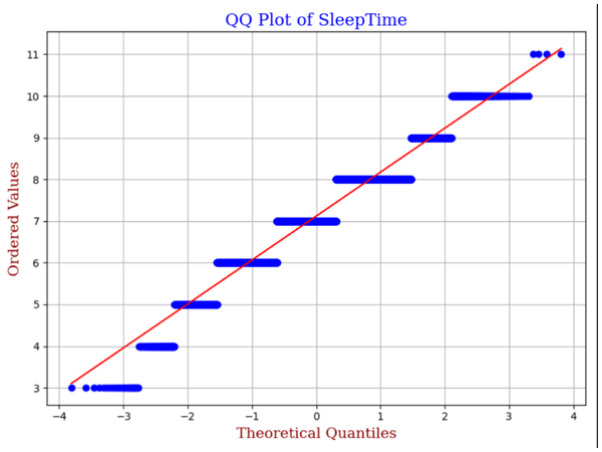
Final shape after outlier removal: (120364, 18)

(Table 1. Outlier Summary Table)

- The filtering helped to remove statistical noise, especially from variables with long tails like Mental Health and Physical Health.
- Categorical binary features like Smoking had no outliers, as expected.
- The radical removal of rows (almost 60% data removed) suggests the presence of overall inconsistencies or extremes but increases the integrity of the following statistical modeling.

Normality Test using QQ Plots

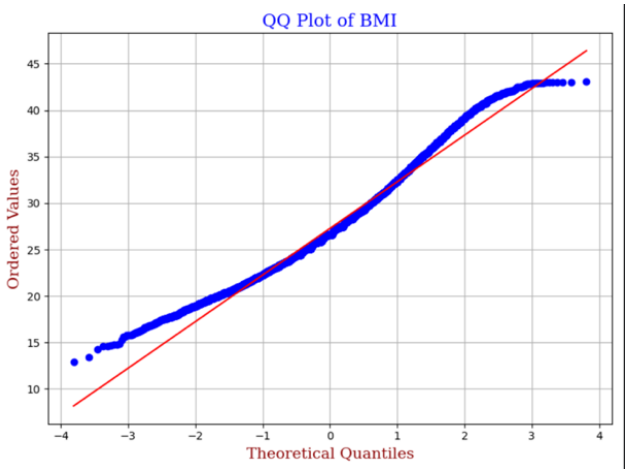
1. **QQ Plot of Sleep Time:**



(Fig 7. QQ Plot – Sleep Time)

- The points largely follow the reference line, indicating that **Sleep Time** approximates a normal distribution after outlier removal. There are slight deviations in the tails, but overall, it is close to Gaussian behavior, making it suitable for linear models.

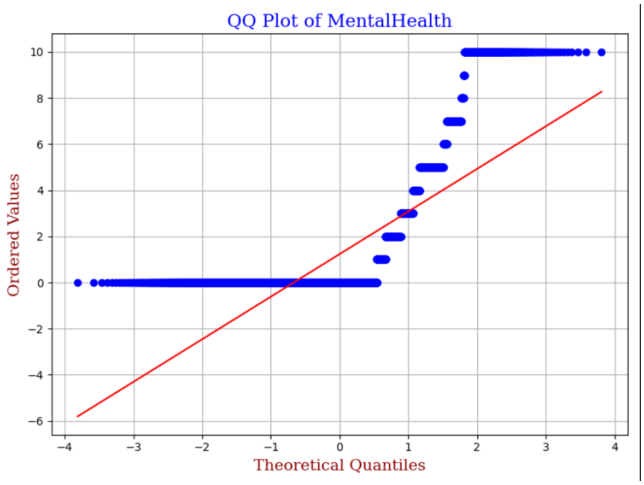
2. **QQ Plot of BMI:**



(Fig 8. QQ Plot – BMI)

- The curve in the plot suggests that **BMI** has a right-skewed distribution, even after removing outliers. A heavier tail is present on the upper end, which is common in biological indicators such as weight or body fat.

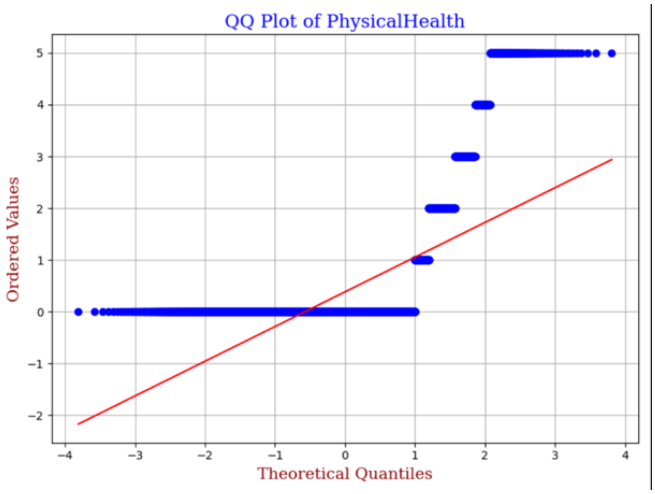
3. QQ Plot of Mental Health:



(Fig 9. QQ Plot – Mental Health)

- The steep upward turn at the high end shows that **Mental Health Days** is **highly right-skewed**, even after cleaning. This skewness could affect models sensitive to normality and may require a transformation like Box-Cox or log.

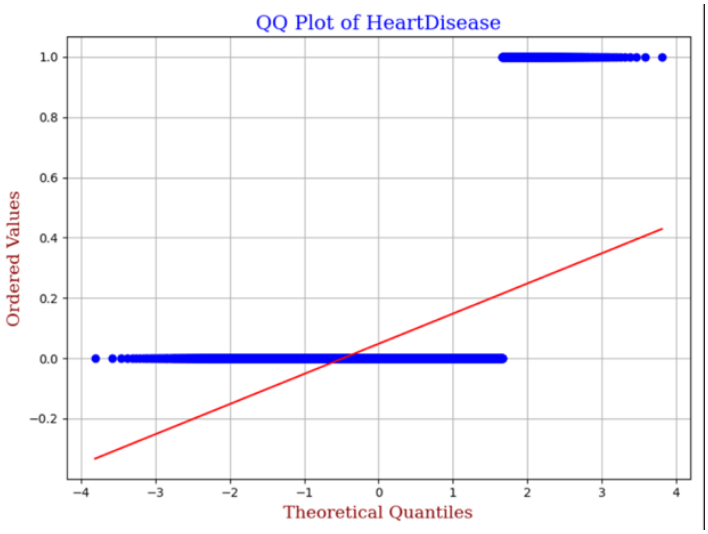
4. QQ Plot of Physical Health:



(Fig 10. QQ Plot – Physical Health)

- Like mental health, **Physical Health Days** also displays right skewness, with values piling at the lower range (0) and tapering sharply after 5 days.

5. QQ Plot of Heart Disease:



(Fig 11. QQ Plot – heart diseases)

- Being a **binary variable**, the QQ plot of **heart disease** shows two flat bands around 0 and 1, which is expected. This confirms it doesn't follow a continuous distribution and should not be tested for normality.

Statistical Normality Test Results:

Apart from QQ plots, various statistical tests were employed to validate the normality of key variables such as:

- **D'Agostino's K² Test** (test for skewness and kurtosis)
- **Shapiro-Wilk Test** (best for small-to-medium-sized datasets)
- **Anderson-Darling Test** (places more weight in the tails of the distribution)

All. tests were applied to the continuous and binary-coded features of the data set. For all tests, the null hypothesis (H_0) is that the data should be normally distributed. A **p-value less than 0.05** leads to rejection of H_0, which corresponds to non-normality.

```
Normality Check Results:

Column: HeartDisease
D'Agostino's K^2 Test: Statistics=8339.91, p-value=0.0000
Shapiro-Wilk Test: Statistics=0.21, p-value=0.0000
Anderson-Darling Test: Statistics=3623.23
The data is not normally distributed (reject H0)

Column: BMI
D'Agostino's K^2 Test: Statistics=439.07, p-value=0.0000
Shapiro-Wilk Test: Statistics=0.98, p-value=0.0000
Anderson-Darling Test: Statistics=57.46
The data is not normally distributed (reject H0)

Column: Smoking
D'Agostino's K^2 Test: Statistics=51899.04, p-value=0.0000
Shapiro-Wilk Test: Statistics=0.60, p-value=0.0000
Anderson-Darling Test: Statistics=2025.12
The data is not normally distributed (reject H0)

Column: AlcoholDrinking
D'Agostino's K^2 Test: Statistics=nan, p-value=nan
Shapiro-Wilk Test: Statistics=1.00, p-value=1.0000
Anderson-Darling Test: Statistics=nan
The data is not normally distributed (reject H0)

Column: Stroke
D'Agostino's K^2 Test: Statistics=nan, p-value=nan
Shapiro-Wilk Test: Statistics=1.00, p-value=1.0000
Anderson-Darling Test: Statistics=nan
The data is not normally distributed (reject H0)
```

(Fig 12.1. Statistical Normality Test Results)

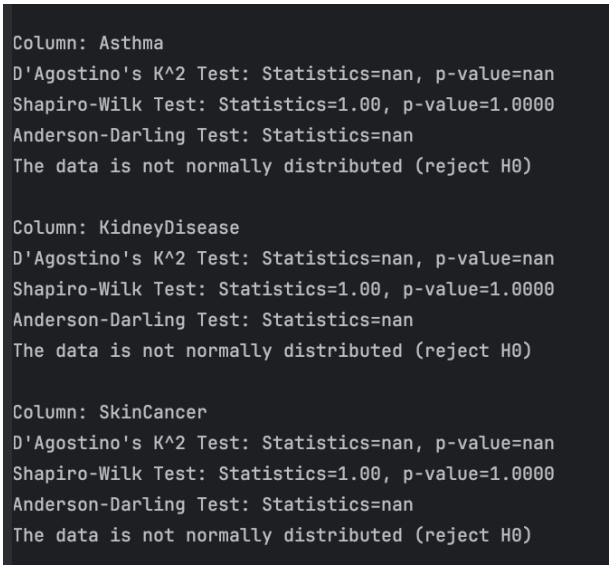
- This table presents the results of three different normality tests D'Agostino's K^2**,** Shapiro-Wilk**,** and Anderson-Darling for the variables Heart Disease, BMI, Smoking, Alcohol Drinking, and Stroke. All tests return extremely low p-values (≈ 0.0000) for continuous variables like BMI, confirming strong deviation from normal distribution. For binary variables such as heart disease and Smoking, although the p-values are low, their categorical nature makes these

results expected and less impactful for modeling. The Alcohol Drinking and Stroke columns show Nan for some test statistics, suggesting invalid input format for continuous normality assumptions.

```
Column: PhysicalHealth
D'Agostino's K^2 Test: Statistics=6044.85, p-value=0.0000
Shapiro-Wilk Test: Statistics=0.43, p-value=0.0000
Anderson-Darling Test: Statistics=2578.79
The data is not normally distributed (reject H0)

Column: MentalHealth
D'Agostino's K^2 Test: Statistics=4288.29, p-value=0.0000
Shapiro-Wilk Test: Statistics=0.58, p-value=0.0000
Anderson-Darling Test: Statistics=1798.22
The data is not normally distributed (reject H0)

Column: DiffWalking
D'Agostino's K^2 Test: Statistics=nan, p-value=nan
Shapiro-Wilk Test: Statistics=1.00, p-value=1.0000
Anderson-Darling Test: Statistics=nan
The data is not normally distributed (reject H0)

Column: PhysicalActivity
D'Agostino's K^2 Test: Statistics=nan, p-value=nan
Shapiro-Wilk Test: Statistics=1.00, p-value=1.0000
Anderson-Darling Test: Statistics=nan
The data is not normally distributed (reject H0)

Column: SleepTime
D'Agostino's K^2 Test: Statistics=163.96, p-value=0.0000
Shapiro-Wilk Test: Statistics=0.92, p-value=0.0000
Anderson-Darling Test: Statistics=370.09
The data is not normally distributed (reject H0)
```

(Fig 12.2. Statistical Normality Test Results)

- This subsection extends the statistical analysis to other health indicators like Physical Health, Mental Health, Diff Walking, Physical Activity, and Sleep Time.
- Physical Health and Mental Health both have extremely high-test statistics and p-values of 0.0000 for all tests, signaling strong non-normality due to right-skewed distributions.
- Sleep Time, while graphically closer to normality in the QQ plot, still fails the statistical tests indicating that minor tail deviations can lead to null hypothesis rejection.



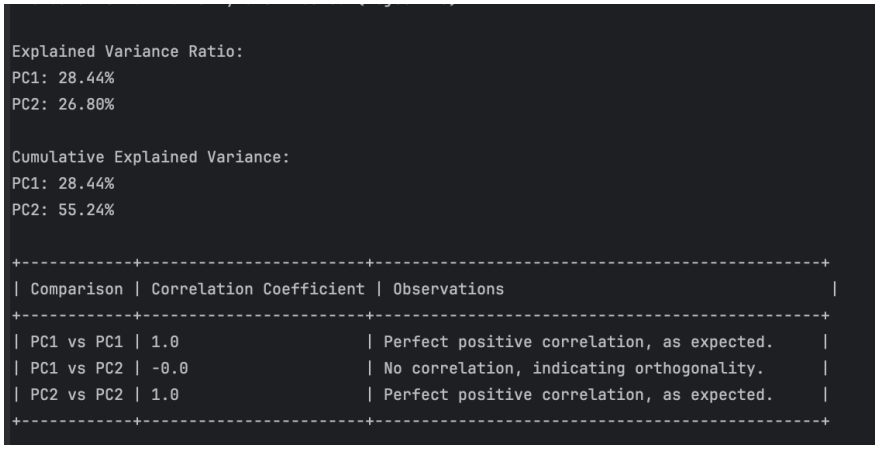
(Fig 12.3. Statistical Normality Test Results)

- This is the summary of Asthma, Kidney Disease, and Skin Cancer results. These three are binary variables (only 0 or 1 values), which produces invalid statistical findings in the D'Agostino and Anderson-Darling tests (Nan) and a perfect test score (1.00) in the Shapiro-Wilk test. These results confirm that normality testing has no application on these variables and needs to be handled by categorical encoding in modeling rather than assuming or imposing Gaussian distribution.

Principal Component Analysis (PCA)

To reduce dimensionality and eliminate redundant or correlated variables**, Principal Component Analysis (PCA)** was applied to the preprocessed dataset. PCA transforms the current variables into a new set of uncorrelated components (Principal Components), ranked according to the variance explained. It is especially handy for visualization, noise removal, and improving the performance of models by retaining only the most informative features.

Variance and Component Correlation:



```
Explained Variance Ratio:
PC1: 28.44%
PC2: 26.80%

Cumulative Explained Variance:
PC1: 28.44%
PC2: 55.24%

+------------+-----------------------+-----------------------------------------------+
| Comparison | Correlation Coefficient | Observations                                 |
+------------+-----------------------+-----------------------------------------------+
| PC1 vs PC1 | 1.0                   | Perfect positive correlation, as expected.    |
| PC1 vs PC2 | -0.0                  | No correlation, indicating orthogonality.     |
| PC2 vs PC2 | 1.0                   | Perfect positive correlation, as expected.    |
+------------+-----------------------+-----------------------------------------------+
```

(Fig 13. PCA Explained Variance and Component Correlation)

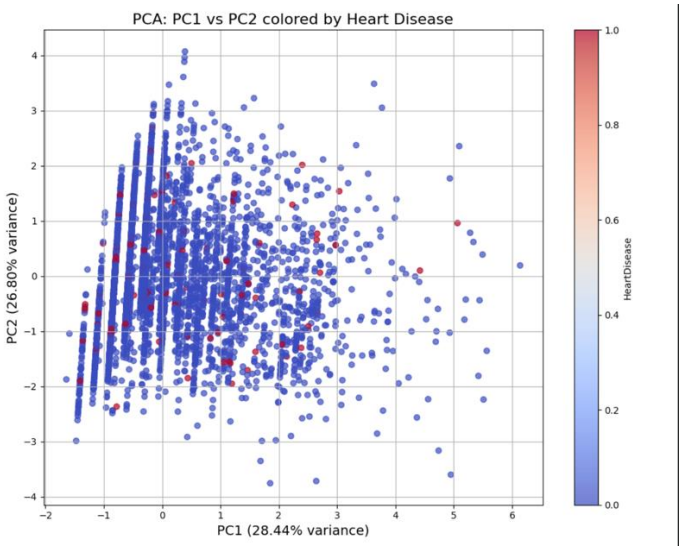
The PCA transformation extracted several components, of which PC1 and PC2 were the initial two. The proportion of explained variance indicates the amount of information (variance) each principal component is extracting from the initial data:

- **PC1:** 28.44%
- **PC2:** 26.80%
- **Cumulative Variance (PC1 + PC2):** 55.24%

A correlation matrix confirms that:

- PC1 vs PC1 and PC2 vs PC2 correlate ideally **(r = 1.0),** as expected.
- PC1 vs PC2 is almost uncorrelated **(r = -0.0),** which confirms that the components are uncorrelated and orthogonal.
- The two initial components alone explain over half of the total variance, sufficient for 2D explorations and noise removal from less informative variables.

Scatter Plot – PC1 vs PC2 Colored by Heart Disease:



(Fig 14. PCA Scatterplot (PC1 vs PC2))

This scatter plot is plotting the transformed dataset versus the first two principal components, each point colored based on the heart disease variable. Red points refer to individuals having heart disease and blue points represent no reported heart disease.

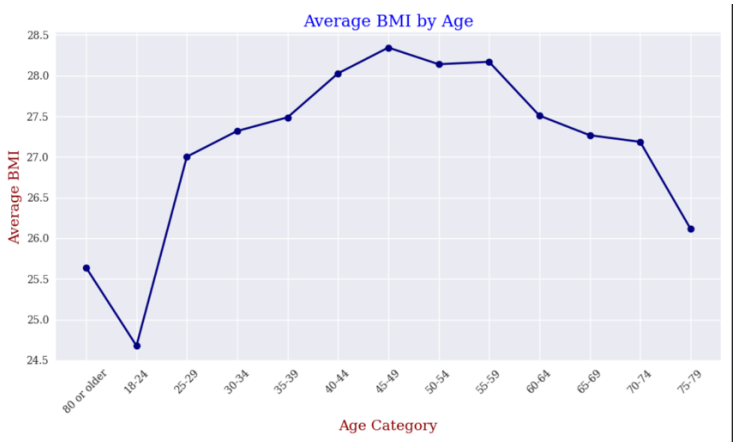
- Tightly banded vertical lines of data points show that while PCA has extracted dimensions, the data remain tightly clustered along PC1.
- Individuals with heart disease (underlined in red) are scattered all over the chart but appear more concentrated in areas, meaning tendencies to be investigated.
- A color gradient bar is a handy visual marker for identifying trends in the distribution of the target variable in PCA space.

PCA was able to project the feature space successfully down to two orthogonal axes without losing over **55% of the original variance.** While it doesn't largely distinguish classes (what occurs in unsupervised PCA), it's a good starting point for viewing and clustering inspection. The extra components (PC3, PC4, etc.) can be used for even higher-dimensional modeling, although PC1 and PC2 are suitable for dashboard inclusion and summary plotting.

Statistics - (Phase 1)

1.1 Data Visualization

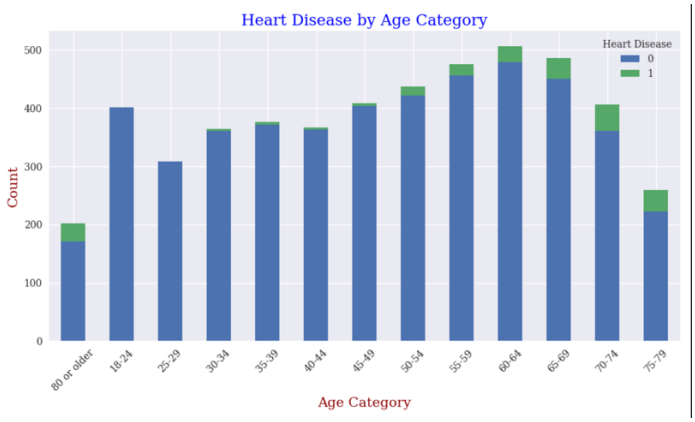
1. **Line Plot – Average BMI by Age**



(Fig 15. Line Plot – Average BMI by Age)

- This line graph shows the pattern of change in mean BMI in different age groups. BMI increases steadily from 18–24 years, reaches a peak at 45–54 years, and falls gradually in higher age groups. It may be because of lifestyle and metabolic changes during life with increased mean BMI in middle-aged people.

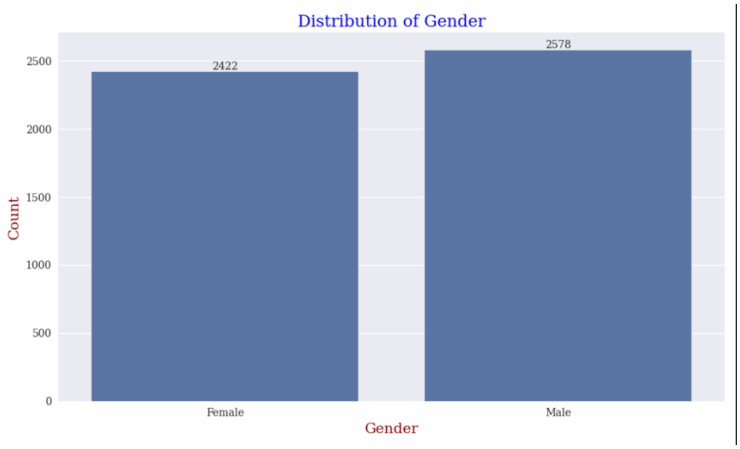
2. **Stacked Bar Plot – heart disease by Age Category**



(Fig 16. Stacked Bar – heart disease by Age Category)

- This stacked bar plot shows the number of people with and without heart disease in various age groups. The green section indicates those with heart disease, rising in older age groups, especially from the age of 50 onwards. This is consistent with established medical trends where age is one of the significant risk factors for cardiovascular illness.

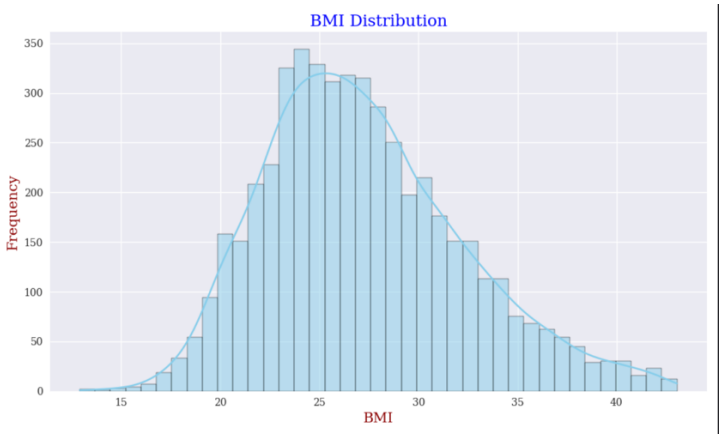
3. **Bar Plot – Distribution of Gender**



(Fig 17. Bar Plot – Distribution of Gender)

- This bar plot is a comparison of gender allocation in the data set. The data set has nearly balanced sex distribution with minimal male surplus (2,578 males and 2,422 females). Synchronized gender composition allows for the unbiased statistical handling in gender-specific comparisons.

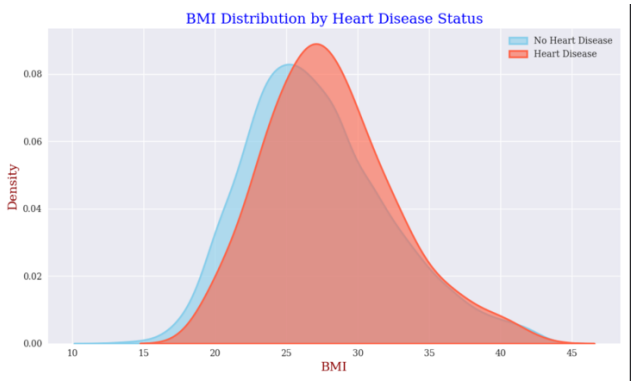
4. **Histogram with KDE – BMI Distribution**



(Fig 18. Histogram with KDE – BMI Distribution)

- This histogram depicts the distribution of BMI in the data. The right-skewed histogram indicates that the majority are in a normal to overweight range. The heavier tail for larger values of BMI is seen by the smooth KDE line, and this indicates this skewness.

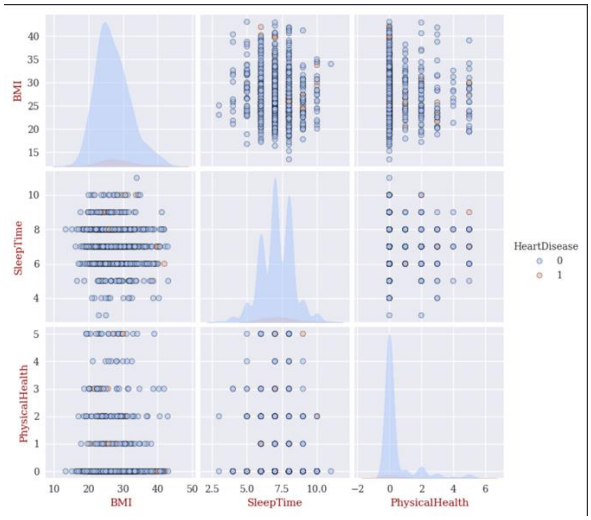
5. **Density Plot – BMI Distribution by Heart Disease Status**



(Fig 19. Density Plot – BMI by Heart Disease Status)

- This graph overlays BMI distributions of heart disease and no heart disease. The red curve (heart disease) is to the right of the blue (no heart disease), which shows higher values of BMI are more common in cardiovascular disease.

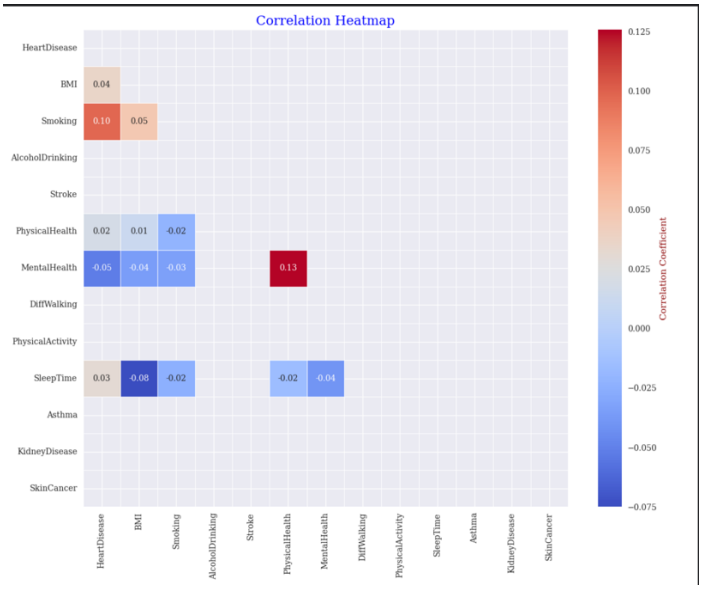
6. **Pair Plot – BMI, Sleep Time, Physical Health vs heart disease**



(Fig 20. Pair Plot – BMI, Sleep Time, Physical Health vs hear diseases)

- This plot demonstrates relationships between Sleep Time, BMI, and Physical Health, with points colored based on heart disease status. It enables us to see how the continuous variables interact and change with heart disease outcomes. No obvious clustering patterns but density indicates central trends.

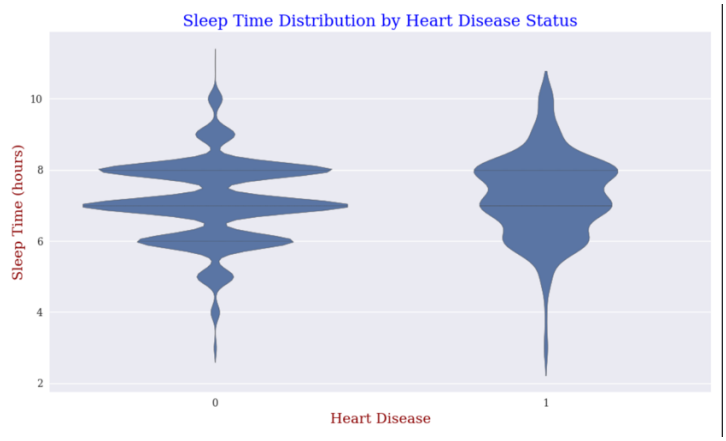
7. **Heatmap – Correlation Matrix**



(Fig 21. Heatmap – Correlation Matrix)

- This heatmap graphs correlation coefficients between features. Most of the correlations are weak, the strongest of which are Mental Health and Heart Disease (0.13). The overall low magnitudes of correlation suggest that features act independently towards the outcome, a best-case scenario for multivariate analysis.

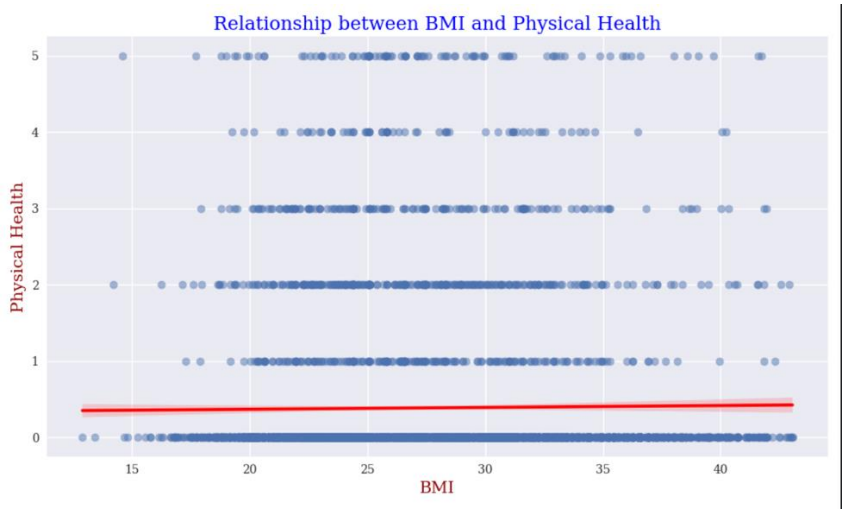
8. **Violin Plot – Sleep Time Distribution by Heart Disease Status**



(Fig 22. Violin Plot – Sleep Time by Heart Disease)

- This violin plot graphs sleep time distribution of those with and without heart disease. The width of the plot is an indication of the density of observations. The heart disease group has tighter and lower sleep time distribution, while the non-disease group has more spread out and small right skew.

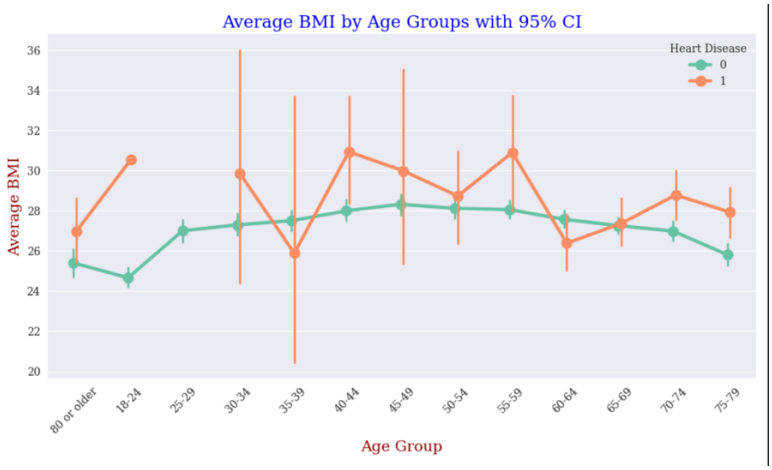
9. **Scatter Plot with Regression Line: Relationship between BMI and Physical Health**



(Fig 23. Scatter Plot with regression line – Relationship of BMI vs Physical health)

- The above scatter plot indicates evidence of correlation between BMI and physical health rating. On the x-axis, each point plots one subject's BMI whereas on the y-axis physical health rating for all individuals is plotted. Red line of regression marks that the curve, if drawn little positive, then a weak, poor association was observed wherein physical health complaint maybe correlates with higher BMI though it reflects weak effect based on extreme clump at low-rated scores.

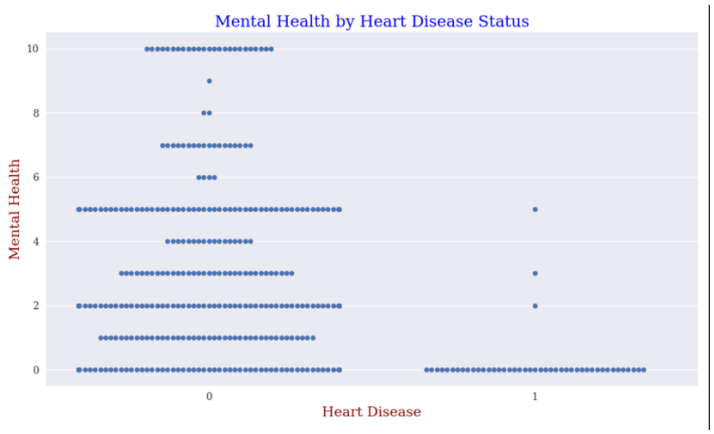
10. Line Plot with Confidence Intervals: Average BMI by Age Groups with 95% CI



(Fig 24. Line Plot with confidence interval – Avg BMI by age group)

- This line plot presents the average BMI by age category, stratified by heart disease status **(0 = No, 1 = Yes).** Shaded bands around each line show 95% confidence intervals. Individuals with heart disease have a higher average BMI than those without, especially in middle-age ranges, though there is a great deal of variability because of smaller sample sizes in some categories.

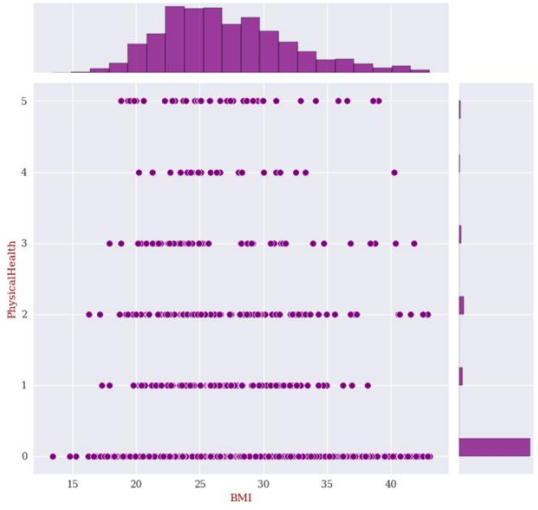
11. **Strip Plot: Mental Health by Heart Disease Status**



(Fig 25. Strip Plot – Mental Health by Heart Diseases Status)

- This strip plot shows the distribution of mental health scores (amount of poor mental health days) for those with and without heart disease. Values tend to cluster around 0 in all instances, but there is more dispersion among those without heart disease. It suggests that mental health variation is more extensive in the healthy population, even though extreme symptoms are also present in heart disease patients.

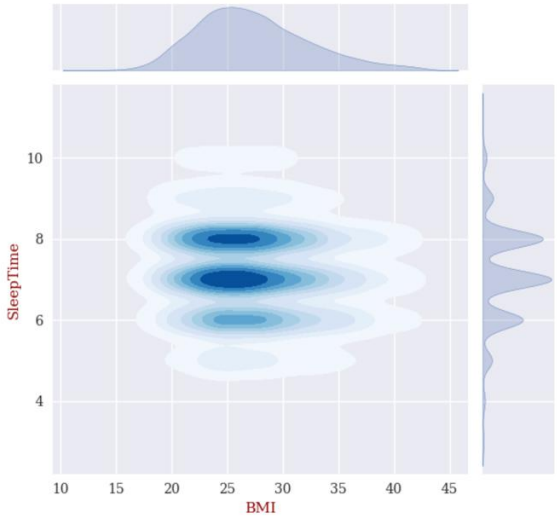
12. **Joint Plot – BMI vs Physical Health**



(Fig 26. Joint Plot – BMI vs Physical Health)

- This joint plot displays the relationship between BMI and Physical Health. Data points are clustered near zero physical health days, with BMI spanning a large range. While no linear relationship is robust, individuals with more unhealthy days continue to have high BMI.

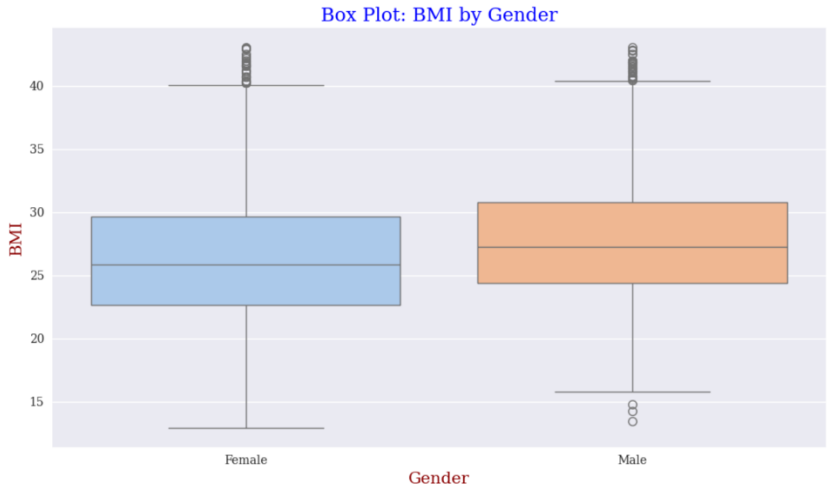
13. **Joint KDE Plot – BMI vs Sleep Time**



(Fig 27. Joint KDE Plot – BMI vs Sleep Time)

- This graph shows the density of combination of BMI and Sleep Time using contour shading. The densest is for BMI 25–30 and sleep time 6–8 hours, meaning these are the most common patterns of health behavior among participants.

14. **Box Plot: BMI by Gender**

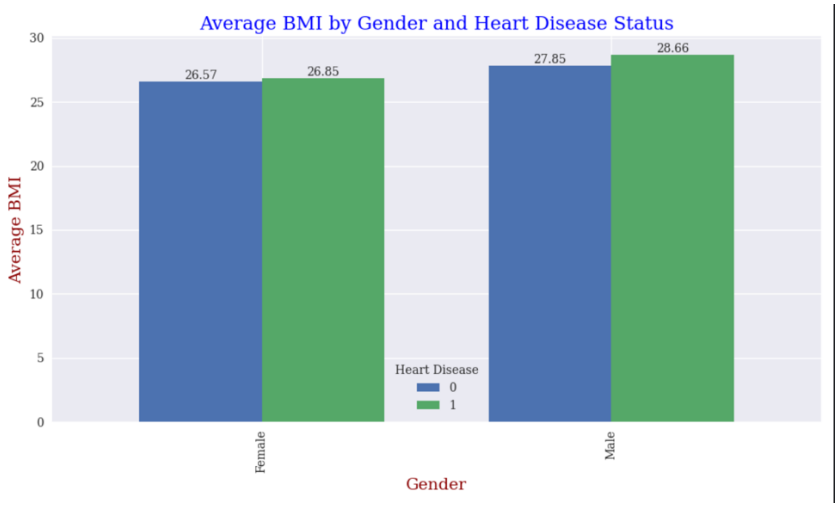


(Fig 28. Box Plot – BMI by Gender)

- The box plot indicates comparison of the distribution of BMI between males and females. It displays the median, quartiles and potential outliers. The males have a slightly higher median BMI and more upper-end outliers, and the females have a narrower interquartile range. It indicates gender-specific variation in BMI distribution in the data.

1.2 Story Telling Plots:

1. Bar Plot: Average BMI by Gender and Heart Disease Status



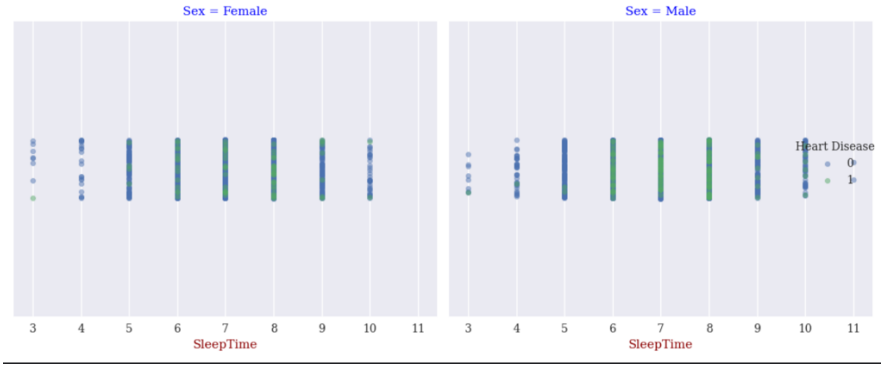
(Fig 29. Storytelling Bar Plot – BMI by Gender and Heart Disease)

- This bar chart illustrates the mean Body Mass Index (BMI) by male and female groups, categorized by heart disease status. In both genders, patients with heart disease have a slightly greater mean BMI than those without. Specifically, mean BMI in females with heart disease is 26.85 compared to 26.57 without heart disease; in males, it is 28.66 with heart disease and 27.85 without.

Interpretation:

- This graph indicates a modest but sustained rise in BMI in heart-disease patients in both men and women. It supports the hypothesis that increased BMI may be a causative risk factor for heart disease, as consistent with known clinical experience.

2. **Strip Plot: Sleep Time Distribution by Gender and Heart Disease**



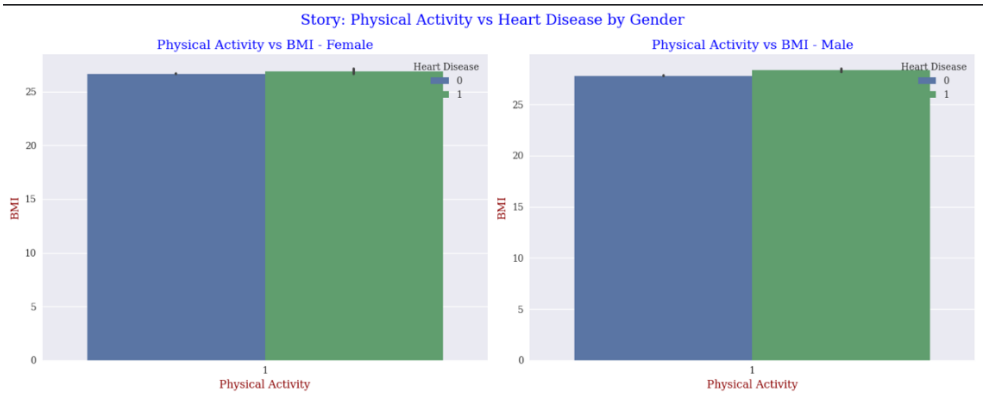
(Fig 30. Strip Plot – Sleep Time by Gender and Heart Disease)

- This is a two-strip plot showing reported sleep time for men and women, split by heart disease status. The dots are all one subject, and the minimal color difference among dots represents presence or absence of heart disease.

Interpretation:

- Sleeping habits appear to be similar regardless of gender or heart disease status, with most sleeping 6 to 9 hours. There is no dramatic visual distinction here between the two heart disease groups, however. This may indicate that sleep time per se is not a robust independent predictor for heart disease in this data.

3. **Bar Plots: Physical Activity vs BMI by Gender and Heart Disease**



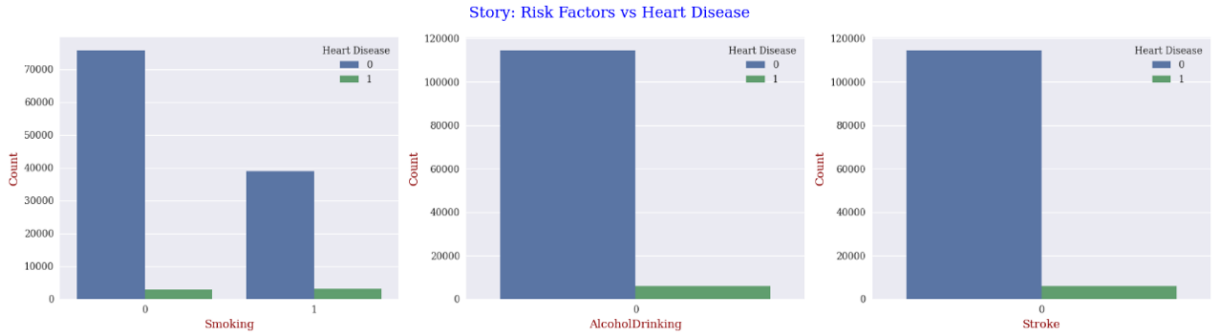
(Fig 31. Storytelling Bar Plot – Physical Activity vs BMI)

- This double bar graph is a comparison of the BMI of physically active men and women, stratified by whether they have heart disease. The bars show that within each gender, individuals who have heart disease have a slightly higher BMI even though they are physically active.

Interpretation:

- Despite exercise, BMI is also greater in individuals who have heart disease. This suggests that exercise alone might not be sufficient to negate BMI-associated heart disease risk suggesting that a multifactorial approach to health must be taken.

4. Clustered Bar Plots: Risk Factors (Smoking, Alcohol, Stroke) vs heart disease



(Fig 32. Storytelling Clustered Bars – Smoking, Alcohol, Stroke vs heart disease)

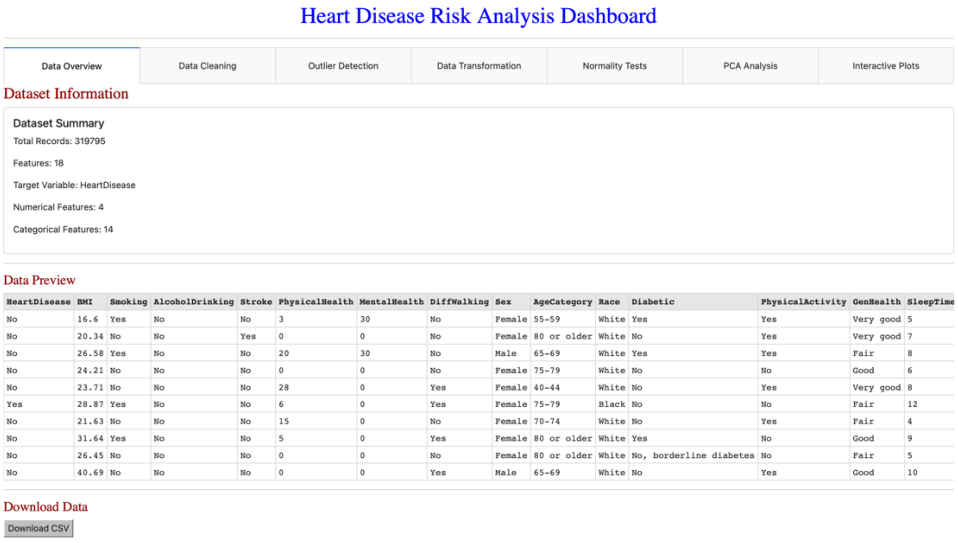
- This series of clustered bar charts demonstrates the prevalence of the presence of heart disease across three categorical risk factors: Smoking, Alcohol Drinking, and Stroke. In both cases, the green bars (heart disease present) are obviously more prominent among individuals with the risk factor (value = 1).

Interpretation:

- Such plots heavily underscore higher rates of heart disease in the case of smokers, drinkers, or stroke sufferers. These are clear and action-oriented indicators, reinforcing known cardiovascular risk correlations.

Interactive web-based dashboard (Phase -2)

Tab 1 - Data Overview:

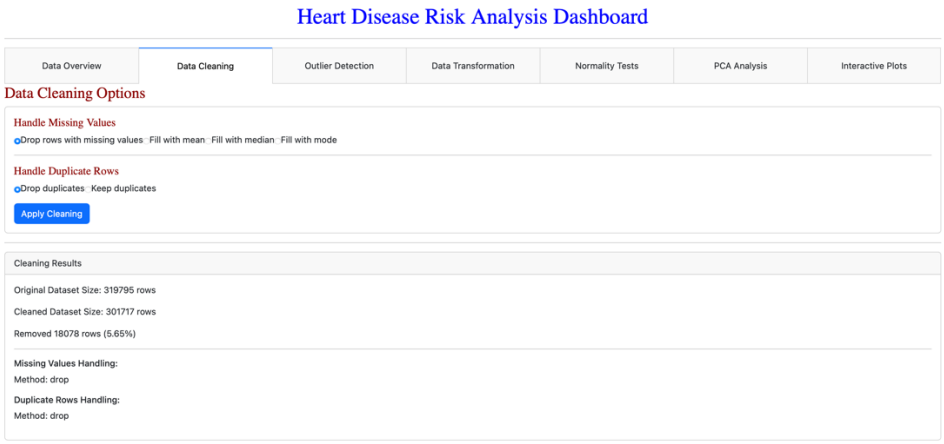


HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealth	SleepTime
No	16.6	Yes	No	No	3	30	No	Female	55-59	White	Yes	Yes	Very good	5
No	20.34	No	No	Yes	0	0	No	Female	80 or older	White	No	Yes	Very good	7
No	26.58	Yes	No	No	20	30	No	Male	65-69	White	Yes	Yes	Fair	8
No	24.21	No	No	No	0	0	No	Female	75-79	White	No	No	Good	6
No	23.71	No	No	No	28	0	Yes	Female	40-44	White	No	Yes	Very good	8
Yes	28.87	Yes	No	No	6	0	Yes	Female	75-79	Black	No	No	Fair	12
No	21.63	No	No	No	15	0	No	Female	70-74	White	No	Yes	Fair	4
No	31.64	Yes	No	No	5	0	Yes	Female	80 or older	White	Yes	No	Good	9
No	26.45	No	No	No	0	0	No	Female	80 or older	White	No, borderline diabetes	No	Fair	5
No	40.69	No	No	No	0	0	Yes	Male	65-69	White	No	Yes	Good	10

(Fig 33. Dashboard Tab 1 – Data Overview)

- This tab provides an instant overview of the dataset that was used to analyze heart disease risk. The Dataset Summary pane explains the data structure that has 319,795 records and 18 features overall.
- Under the summary, the Data Preview section displays a snapshot of the dataset giving users an instant glance at how variables like BMI, age, gender, lifestyle factors (e.g., smoking, alcohol consumption, exercise), and health status look like in their raw state. This helps users familiarize themselves with the structure, categories, and likely values in the dataset.
- A Download CSV button is included so that users can download the data for independent replication or analysis.
- This overview tab serves as the foundation of the dashboard by offering context to the data being addressed in subsequent steps like transformation, cleaning, and modeling.

Tab 2 - Data Cleaning:



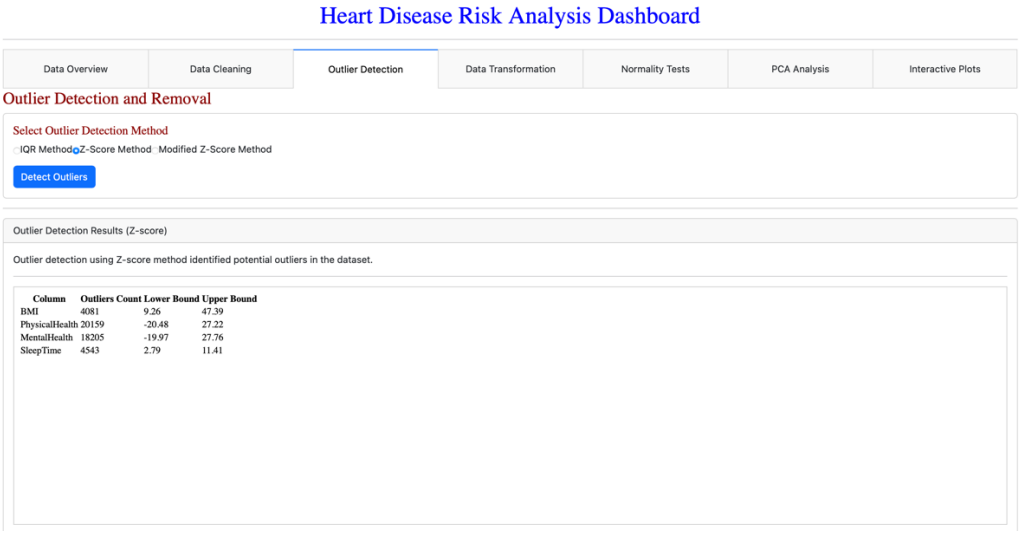
(Fig 34. Dashboard Tab 2 – Data Cleaning)

- The Data Cleaning tab is interested in preparing the dataset for analysis by addressing missing and duplicate values. There is a variety of interactive cleaning options, where the user can choose from a variety of imputation methods for missing values (mean, median, or mode) or just drop rows containing them. Similarly, users can choose whether to retain or delete duplicate records.

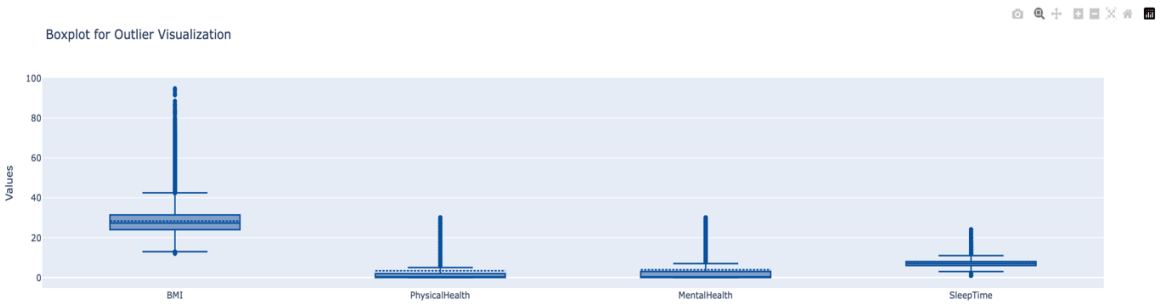
    1. **Original dataset size:** 319,795 rows
    2. **Final clean dataset size:** 301,717 rows
    3. **Total deleted:** 18,078 rows (5.65%)

- This step is crucial as it avoids biased insights and reduces noise in subsequent processes like modeling and visualization.

Tab 3 - Outlier Detection:



(Fig 35. Dashboard Tab 3 – Z-Score Outlier Detection)



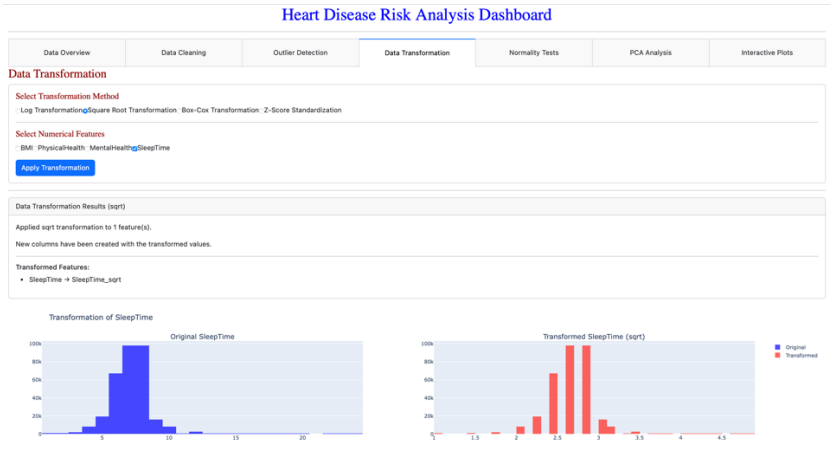
(Fig 36. Dashboard Tab 3 – Outlier Boxplot)

- The Outlier Detection tab uses statistical techniques to calculate outliers within the numeric attributes. The Z-score method was used to identify outliers in BMI, Physical Health, Mental Health, and Sleep Time, measuring how far each value is from the mean in standard units.

- **The results were:**
    - **BMI:** 4,081 outliers (Bounds: 9.26 – 47.39)
    - **Physical Health:** 20,159 outliers (Bounds: -20.48 – 27.22)
    - **Mental Health:** 18,205 outliers (Bounds: -19.97 – 27.76)
    - **Sleep Time:** 4,543 outliers (Bounds: 2.79 – 11.41)

- To augment this, a boxplot graphically highlights spread and concentration of outliers. Worth mentioning, BMI has extreme high values, while Physical and Mental Health have tails of high outliers. Sleep Time is more centralized with the presence of anomalies. This combination of statistical and graphical methods offers better understanding and pre-treatment of the data for future analysis by minimizing the impacts of extreme values.

Tab 4 - Data Transformation:



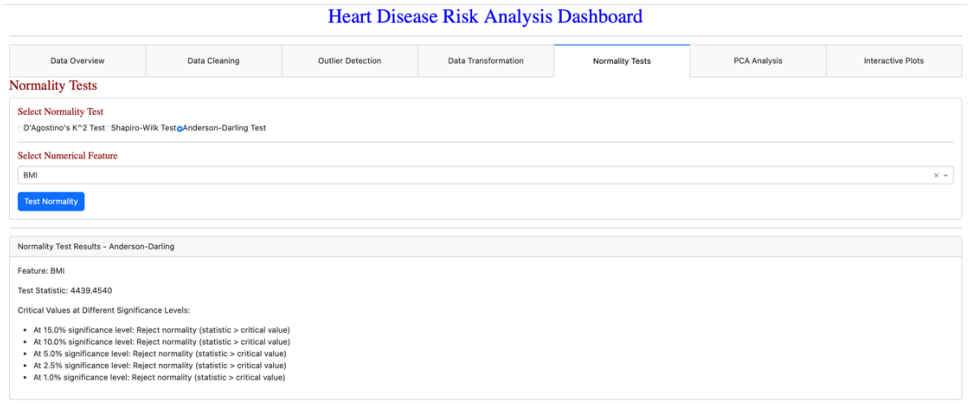
(Fig 37. Dashboard Tab 4 – Sleep Time Transformation)

- Data Transformation tab provides options for scaling and normalization of numerical features to optimize statistical performance as well as improve model accuracy. Square Root Transformation was applied here on the variable Sleep Time with the hope to reduce skewness and shift the distribution closer towards normal.

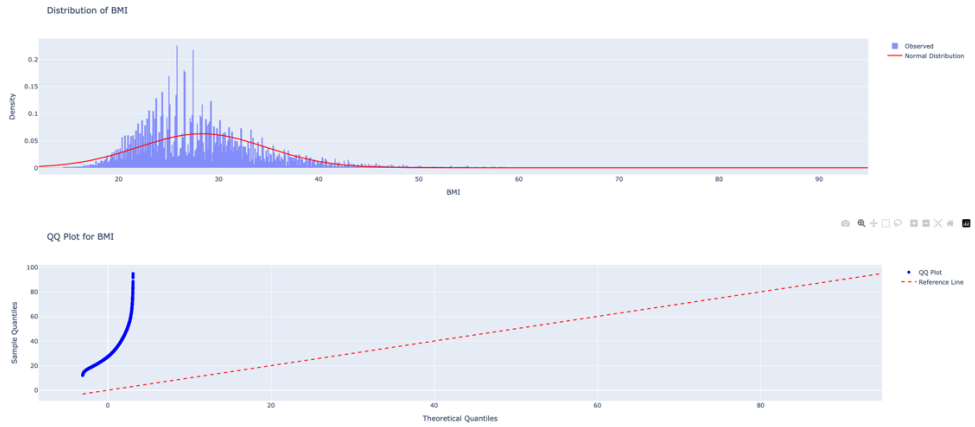
- **After transformation:**

    1. New column, Sleep Time - sqrt, was generated.
    2. The graphical plot comparison of the original and the transformed histograms shows that the square root transformation compressed the range and decreased the right skewness, making it a more symmetric data.
    3. This is particularly useful before applying procedures that assume normality or linearity, and it allows robust downstream analysis.

Tab 5 - Normality Tests:



(Fig 38. Dashboard Tab 5 – BMI Normality Test (Anderson-Darling))



(Fig 39. Dashboard Tab 5 – Histogram and QQ Plot for BMI)

- Normality Tests tab of the Heart Disease Risk Analysis Dashboard is used to check if a numerical attribute has a normal distribution—a requirement for most statistical models. During this analysis, the Anderson-Darling Test was applied for the BMI attribute to check its distributional property.

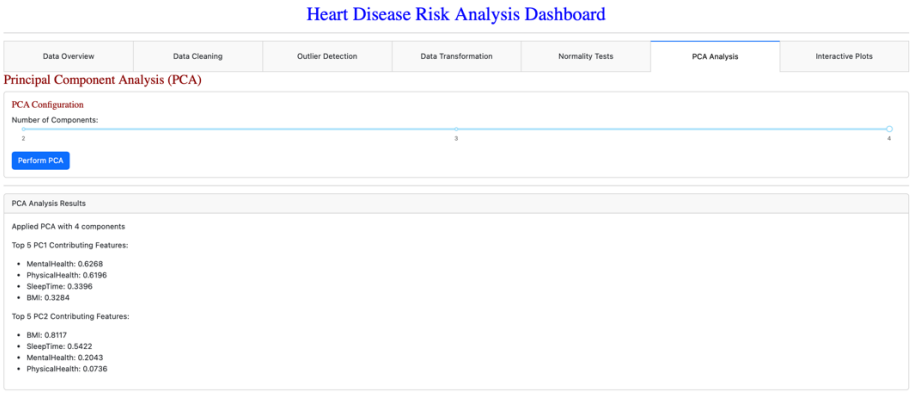
- Anderson-Darling test gave a hugely large test statistic of 4439.45, much higher than the critical values for any standard significance levels (15%, 10%, 5%, 2.5%, and 1%). This leads to us rejecting the null hypothesis and to the conclusion that the data for BMI is not normally distributed.

- **To conclude this graphically, two complementary plots were introduced:**

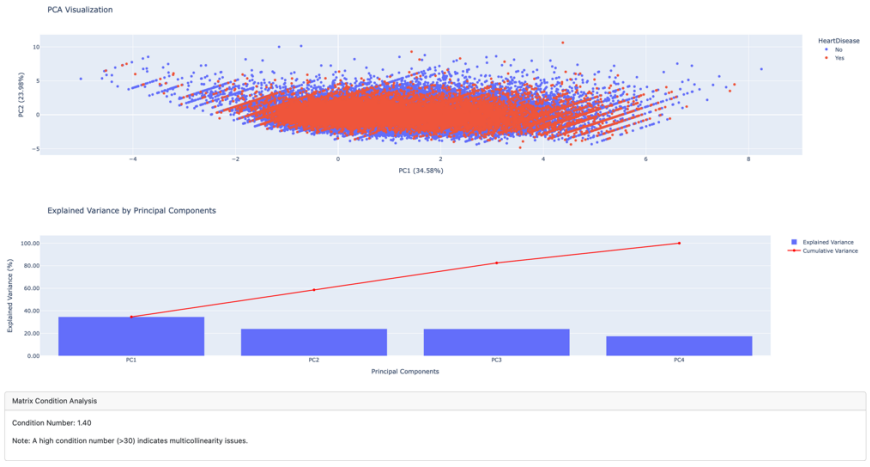
1. The **Density Plot of BMI** (with a red overlay of the normal distribution) clearly shows that the observed BMI distribution is right-skewed, deviating from the smooth bell-shaped curve of a normal distribution.
2. The **QQ Plot** (Quantile-Quantile Plot) compares the quantiles of the BMI data against a theoretical normal distribution. The sharp curvature and deviation of points from the reference line further confirm that BMI is not normally distributed.

- This combination of statistical testing and visual diagnostics ensures a comprehensive evaluation of distribution assumptions. Since BMI is not normally distributed, appropriate transformation or non-parametric methods should be considered for further modeling.

Tab 6 - PCA Analysis:



(Fig 40. Dashboard Tab 6 – PCA Results)



(Fig 41. Dashboard Tab 6 – PCA Scatterplot)

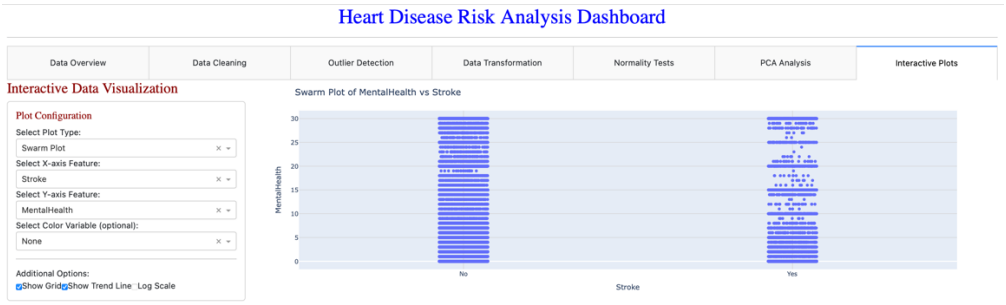
- The PCA tab provides dimensionality reduction output by identifying the principal components that account for the highest variance in the data. In the current case, PCA was applied with 4 components on the numeric features: **BMI, Sleep Time, Mental Health, and Physical Health.**

    1. PC1 accounted for the highest variance (34.58%) and was led by Mental Health (0.6268) and Physical Health (0.6196).
    2. PC2 explained an additional 23.98% of the variance, largely driven by BMI (0.8117) and Sleep Time (0.5422).
    3. The cumulative variance plot shows that the first components capture a high percentage of variability, and hence dimension reduction with minimal loss of data is justified.

- A scatterplot of PC1 vs PC2, conditioned upon heart disease status, helps visually identify clustering or separation between groups, although no strong separation occurs.

- Finally, a condition number of 1.40 shows no multicollinearity issue and ensures stable outcomes of PCA. PCA helps lower data complexity without sacrificing critical patterns that may prove helpful for predictive modeling.

Tab 7 - Interactive Plots:



(Fig 42. Dashboard Tab 7 – Swarm Plot (Mental Health vs Stroke))

- The Interactive Plots tab allows users to dynamically investigate the correlations between different features with interactive visualization tools. Under the current setup, a Swarm Plot is utilized to analyze the distribution of Mental Health scores in people with and without a history of Stroke.

    1. The X-axis plots the categorical variable Stroke (Yes or No), and the Y-axis displays corresponding Mental Health values.
    2. Every point is a person's reported number of mentally unhealthy days.

Real comparison of mental health trends between the stroke and non-stroke groups.

    3. The plot suggests that respondents reporting stroke had a wider distribution of mentally unhealthy days than those without stroke, and the arrow indicates a possible mental burden associated with stroke history.

- This section adds further value in terms of interactivity because it enables users to experiment with different combinations of variables and types of plots to aid exploratory data analysis without coding.

Deployment (Phase -3)

As part of Phase 3, the final Heart Disease Risk Analysis Dashboard has been successfully deployed and made accessible via a web application. This interactive dashboard enables users to explore the dataset, perform statistical analyses, and visualize patterns related to heart disease in a user-friendly environment. It supports end-to-end functionality rom data cleaning to principal component analysis and story-telling plots offering both statistical depth and visual clarity.

You can access and interact with the deployed dashboard using the following link:
 View Dashboard Deployment

This deployment makes the entire analysis pipeline easily shareable and usable for stakeholders, researchers, or anyone interested in heart disease data exploration.

Conclusion

The Heart Disease Risk Analysis Dashboard delivers an interactive and integrated approach of exploring important health determinants associated with heart disease. With a properly groomed pipeline that ranges from data cleaning and deletion of outliers through transformation, normality checks, and principal component analysis, the dashboard ensures the integrity of data and the validity of statistical assumptions before in-depth exploration.

Variational and narrative plot visualizations and plots best capture significant patterns like BMI vs. physical health, mental health with heart disease variability, and how risk factors of smoking, physical inactivity, and stroke interact. Principal Component Analysis also exhibits the leading contributors to variance in terms of the most significant being mental and physical health. User-driven insights supplemented by interactive plotting provide additional facilitation of user-generated hypotheses along with further probing.

In general, the dashboard allows users to make data-driven, informed inferences that can lead to early risk identification and enhanced knowledge of heart disease correlates.

Reference

1. Indicators of heart Disease (2022 UPDA TE). (2023, October 12). Kaggle. https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease/data

2. Le, P ., Casper, M., & V aughan, A. S. (2022). A dynamic visualization tool of local trends in heart
disease and stroke mortality in the United States. Preventing Chronic Disease, 19. https://doi.org/10.5888/pcd19.220076

3. Comprehensive Analysis of Heart Disease Prediction: Machine Learning approach. (2022, October 7). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9972035

Appendix

```
#%%
# %%
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import normaltest, boxcox
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from prettytable import PrettyTable
from io import BytesIO
import base64
import re
import warnings
import sys
import os

warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)

RUN_PHASE_1 = True


def load_and_clean_data(file_path="heart_2020_cleaned.csv"):
    try:
        # Load data
        data = pd.read_csv(file_path)

        # Print initial info
        print("\nInitial Data Shape:", data.shape)
        print("\nAvailable Columns:", list(data.columns))
        print("\nFirst 5 rows:")
        print(data.head())

        # Print Before Cleaning snapshot
        print("\nSnapshot Before Cleaning:")
        print(data.head(10).to_string(index=False))
        nan_summary = data.isna().sum()

        # Create PrettyTable for missing values
        table = PrettyTable()
        table.field_names = ["Column", "Missing Values"]
        for column, value in nan_summary.items():
            if value > 0:
                table.add_row([column, value])
        if len(table.rows) > 0:
            table.title = 'Null Values in the Dataset'
            print("\nMissing Values Summary:")
            print(table)
        else:
            print("\nNo missing values found in the dataset")
```

```
        # Handle missing values
        initial_rows = data.shape[0]
        data = data.dropna()
        print(f"\nDropped {initial_rows - data.shape[0]} rows with missing values")
        print("After dropping NA values:", data.shape)

        # Remove duplicates
        initial_rows = data.shape[0]
        data = data.drop_duplicates()
        print(f"Dropped {initial_rows - data.shape[0]} duplicate rows")
        print("After dropping duplicates:", data.shape)

        # Rename column for consistency if it exists
        if 'HeartDisease' in data.columns:
            data['HeartDisease'] = data['HeartDisease'].map({'Yes': 1, 'No': 0})
        elif 'HadHeartAttack' in data.columns:
            data['HadHeartAttack'] = data['HadHeartAttack'].map({'Yes': 1, 'No': 0})
        for col in data.columns:
            if data[col].dtype == 'object':
                if set(data[col].unique()) == {'Yes', 'No'}:
                    data[col] = data[col].map({'Yes': 1, 'No': 0})
        data.reset_index(drop=True, inplace=True)

        # Print After Cleaning snapshot
        print("\nSnapshot After Cleaning:")
        print(data.head(10).to_string(index=False))

        return data

    except Exception as e:
        print(f"\nError loading data: {str(e)}")
        return None


def detect_and_remove_outliers(df, method='IQR'):
    try:
        print("\nSnapshot BEFORE Outlier Removal:")
        print(df.head(10).to_string(index=False))

        # Show 4 boxplots BEFORE outlier removal
        plt.figure(figsize=(15, 10))
        numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()[:4]
        for i, col in enumerate(numeric_cols):
            plt.subplot(2, 2, i + 1)
            df.boxplot(column=[col], flierprops=dict(marker='o', markerfacecolor='blue',
                                                     markersize=8, linestyle='none', alpha=0.2))
            plt.title(f'Before - {col}', fontsize=14, color='blue')
            plt.ylabel('Value', fontsize=12, color='darkred')
            plt.grid(True)
        plt.tight_layout()
        plt.show()
        plt.savefig('boxplots_before_outlier_removal.png')
        plt.close()

        numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
        numerical_columns = [col for col in numerical_columns if col not in ['HeartDisease', 'HadHeartAttack']]

        # Only proceed if we have numerical columns
```

```
    if len(numerical_columns) == 0:
        print("\nNo numerical columns found for outlier detection")
        return df

    print("\nNumerical columns for outlier detection:", list(numerical_columns))

    # Create box plots for outlier visualization
    num_plots = len(numerical_columns)
    rows = (num_plots + 1) // 2
    plt.figure(figsize=(18, 5 * rows))

    for i, col in enumerate(numerical_columns):
        plt.subplot(rows, 2, i + 1)
        df.boxplot(column=[col], flierprops=dict(marker='o', markerfacecolor='blue',
                                                 markersize=8, linestyle='none', alpha=0.2))
        plt.title(col, fontdict={'fontsize': 'large', 'fontweight': 'bold',
                                 'color': 'blue', 'fontname': 'serif'})
        plt.ylabel('Value', fontdict={'fontsize': 'large', 'fontweight': 'bold',
                                      'color': 'darkred', 'fontname': 'serif'})
        plt.grid(True)

    plt.tight_layout()
    plt.show()
    plt.savefig('outliers_boxplot.png')
    plt.close()

    # Create PrettyTable for outlier summary
    outlier_table = PrettyTable()
    outlier_table.field_names = ["Column Name", "Outliers Count", "Lower Bound", "Upper Bound"]
    df_clean = df.copy()

    # Remove outliers using IQR method
    if method == 'IQR':
        for column in numerical_columns:
            Q1 = df[column].quantile(0.25)
            Q3 = df[column].quantile(0.75)
            IQR = Q3 - Q1

            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
            outliers_count = outliers[column].count()

            outlier_table.add_row([column, outliers_count,
                                   round(lower_bound, 2), round(upper_bound, 2)])

            # Filter out rows with outliers
            df_clean = df_clean[(df_clean[column] >= lower_bound) & (df_clean[column] <= upper_bound)]

    print("\nOutlier Summary:")
    print(outlier_table)

    df_clean.reset_index(drop=True, inplace=True)
    print("\nFinal shape after outlier removal:", df_clean.shape)

    # Show 4 boxplots AFTER outlier removal
    plt.figure(figsize=(15, 10))
    numeric_cols = df_clean.select_dtypes(include=['int64', 'float64']).columns.tolist()[:4]
```

```
        for i, col in enumerate(numeric_cols):
            plt.subplot(2, 2, i + 1)
            df_clean.boxplot(column=[col], flierprops=dict(marker='o', markerfacecolor='blue',
                                                           markersize=8, linestyle='none', alpha=0.2))
            plt.title(f'After - {col}', fontsize=14, color='blue')
            plt.ylabel('Value', fontsize=12, color='darkred')
            plt.grid(True)
        plt.tight_layout()
        plt.show()
        plt.savefig('boxplots_after_outlier_removal.png')
        plt.close()

        # Print snapshot AFTER outlier removal
        print("\nSnapshot AFTER Outlier Removal:")
        print(df_clean.head(10).to_string(index=False))

        return df_clean

    except Exception as e:
        print(f"\nError during outlier removal: {str(e)}")
        return df


def check_normality(df):
    try:
        if len(df) > 10000:
            df = df.sample(10000, random_state=42)
        numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns

        if len(numerical_cols) == 0:
            print("\nNo numerical columns found for normality check")
            return {}, {}

        print("\nNormality Check Results:")

        # Store results for dashboard
        normality_results = {}
        qq_plot_images = {}

        for col in numerical_cols:
            try:
                col_data = df[col].dropna()
                if len(col_data) < 8:
                    print(f"\nColumn: {col} - Not enough samples for normality test (min 8 required)")
                    continue

                # D'Agostino's K^2 Test
                stat, p = normaltest(col_data)
                print(f"\nColumn: {col}")
                print(f"D'Agostino's K^2 Test: Statistics={stat:.2f}, p-value={p:.4f}")

                # Shapiro-Wilk Test
                shapiro_stat, shapiro_p = stats.shapiro(col_data[:5000])
                print(f"Shapiro-Wilk Test: Statistics={shapiro_stat:.2f}, p-value={shapiro_p:.4f}")

                # Anderson-Darling Test
                anderson_result = stats.anderson(col_data, dist='norm')
                print(f"Anderson-Darling Test: Statistics={anderson_result.statistic:.2f}")
```

```
            alpha = 0.05
            if p > alpha and shapiro_p > alpha:
                result = "The data is normally distributed (fail to reject H0)"
            else:
                result = "The data is not normally distributed (reject H0)"

            print(result)

            # Store results
            normality_results[col] = {
                'dagostino': {'stat': float(stat), 'p': float(p)},
                'shapiro': {'stat': float(shapiro_stat), 'p': float(shapiro_p)},
                'anderson': {'stat': float(anderson_result.statistic)},
                'result': result
            }

            # Create QQ plot for this column
            plt.figure(figsize=(8, 6))
            stats.probplot(col_data, dist="norm", plot=plt)
            plt.title(f'QQ Plot of {col}',
                      fontdict={'fontname': 'serif', 'color': 'blue', 'fontsize': 16})
            plt.xlabel('Theoretical Quantiles',
                       fontdict={'fontname': 'serif', 'color': 'darkred', 'fontsize': 14})
            plt.ylabel('Ordered Values',
                       fontdict={'fontname': 'serif', 'color': 'darkred', 'fontsize': 14})
            plt.grid(True)
            plt.tight_layout()
            plt.show()
            plt.savefig(f'qq_plot_{col}.png')
            plt.close()

            # Store image in dictionary
            buf = BytesIO()
            plt.figure(figsize=(8, 6))
            stats.probplot(col_data, dist="norm", plot=plt)
            plt.title(f'QQ Plot of {col}',
                      fontdict={'fontname': 'serif', 'color': 'blue', 'fontsize': 16})
            plt.xlabel('Theoretical Quantiles',
                       fontdict={'fontname': 'serif', 'color': 'darkred', 'fontsize': 14})
            plt.ylabel('Ordered Values',
                       fontdict={'fontname': 'serif', 'color': 'darkred', 'fontsize': 14})
            plt.grid(True)
            plt.tight_layout()
            plt.show()
            plt.savefig(buf, format='png')
            plt.close()

            buf.seek(0)
            img_str = base64.b64encode(buf.read()).decode('utf-8')
            qq_plot_images[col] = f'{img_str}'

        except Exception as e:
            print(f"\nError processing column {col}: {str(e)}")

    return normality_results, qq_plot_images

except Exception as e:
    print(f"\nError during normality check: {str(e)}")
    return {}, {}
```

```
def perform_data_transformation(df, method='log', columns=None):
    df_transformed = df.copy()

    if columns is None:
        columns = df.select_dtypes(include=['float64', 'int64']).columns

    transformation_results = {}

    for col in columns:
        # Skip binary columns and ensure positive values for log transform
        if set(df[col].unique()).issubset({0, 1}) or df[col].min() <= 0:
            continue

        try:
            if method == 'log':
                df_transformed[f"{col}_log"] = np.log(df[col])
                transformation_results[col] = {'method': 'log', 'new_column': f"{col}_log"}

            elif method == 'sqrt':
                df_transformed[f"{col}_sqrt"] = np.sqrt(df[col])
                transformation_results[col] = {'method': 'sqrt', 'new_column': f"{col}_sqrt"}

            elif method == 'boxcox':
                transformed_data, lambda_val = boxcox(df[col])
                df_transformed[f"{col}_boxcox"] = transformed_data
                transformation_results[col] = {
                    'method': 'boxcox',
                    'lambda': lambda_val,
                    'new_column': f"{col}_boxcox"
                }

            elif method == 'zscore':
                df_transformed[f"{col}_zscore"] = (df[col] - df[col].mean()) / df[col].std()
                transformation_results[col] = {'method': 'zscore', 'new_column': f"{col}_zscore"}

        except Exception as e:
            print(f"Error transforming {col} with {method}: {e}")

    return df_transformed, transformation_results

def perform_pca_analysis(df, n_components=2):
    try:
        if len(df) > 5000:
            df = df.sample(5000, random_state=42)
        # Select numerical columns
        numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

        # Remove binary columns
        numerical_cols = [col for col in numerical_cols if len(df[col].unique()) > 2]

        # Standardize the data
        scaler = StandardScaler()
        # Limit to 5000 rows for performance
        df_sample = df[numerical_cols].dropna().copy()
        if len(df_sample) > 5000:
            df_sample = df_sample.sample(5000, random_state=42)
```

```
scaled_data = scaler.fit_transform(df_sample)

# Perform PCA
pca = PCA(n_components=n_components)
principal_components = pca.fit_transform(scaled_data)
pca_df = pd.DataFrame(
    data=principal_components,
    columns=[f'PC{i + 1}' for i in range(n_components)]
)

# Find target column if it exists
target_col = None
for col_name in ['HeartDisease', 'HadHeartAttack']:
    if col_name in df.columns:
        target_col = col_name
        break

if target_col:
    pca_df[target_col] = df.loc[df_sample.index, target_col].values

# Calculate condition number
cond_num = np.linalg.cond(scaled_data)
_, singular_values, _ = np.linalg.svd(scaled_data)

# Calculate explained variance
explained_variance = pca.explained_variance_ratio_
# Explained Variance and Cumulative Variance
explained_variance_percent = explained_variance * 100
cumulative_variance = np.cumsum(explained_variance_percent)

print("\nExplained Variance Ratio:")
for i, var in enumerate(explained_variance_percent):
    print(f"PC{i+1}: {var:.2f}%")

print("\nCumulative Explained Variance:")
for i, cum_var in enumerate(cumulative_variance):
    print(f"PC{i+1}: {cum_var:.2f}%")

# Correlation matrix of PCs
pc_df = pd.DataFrame(principal_components, columns=[f"PC{i+1}" for i in range(n_components)])
corr_matrix = pc_df.corr()

print("\n" + "+" + "-"*12 + "+" + "-"*24 + "+" + "-"*48 + "+")
print("| {:<10} | {:<22} | {:<46} |".format("Comparison", "Correlation Coefficient", "Observations"))
print("+" + "-"*12 + "+" + "-"*24 + "+" + "-"*48 + "+")
for i in range(n_components):
    for j in range(i, n_components):
        comparison = f"PC{i+1} vs PC{j+1}"
        coeff = corr_matrix.iloc[i, j]
        if i == j:
            note = "Perfect positive correlation, as expected."
        else:
            note = "No correlation, indicating orthogonality."
        print("| {:<10} | {:<22.1f} | {:<46} |".format(comparison, coeff, note))
print("+" + "-"*12 + "+" + "-"*24 + "+" + "-"*48 + "+")

# Create PCA results dictionary
pca_results = {
    'pca_df': pca_df,
```

```
            'explained_variance': explained_variance,
            'condition_number': cond_num,
            'singular_values': singular_values,
            'feature_names': numerical_cols,
            'loadings': pca.components_
        }

        # Plot PCA visualization
        plt.figure(figsize=(10, 8))
        if target_col and target_col in pca_df.columns:
            plt.scatter(pca_df['PC1'], pca_df['PC2'],
                        c=pca_df[target_col],
                        cmap='coolwarm',
                        alpha=0.7)
            plt.colorbar(label=target_col)
            plt.title('PCA: PC1 vs PC2 colored by Heart Disease', fontsize=16)
            plt.xlabel(f'PC1 ({explained_variance[0]:.2%} variance)', fontsize=14)
            plt.ylabel(f'PC2 ({explained_variance[1]:.2%} variance)', fontsize=14)
            plt.grid(True)
            plt.tight_layout()
            plt.show()
            plt.savefig('pca_visualization.png')
            plt.close()

        return pca_results

    except Exception as e:
        print(f"Error in PCA analysis: {e}")
        return None


def create_subplots(df):
    # 1. Storytelling Subplot - Bar plots: PhysicalActivity vs HeartDisease by Sex
    if 'Sex' in df.columns and 'PhysicalActivity' in df.columns and 'BMI' in df.columns and 'HeartDisease' in df.columns:
        plt.figure(figsize=(15, 6))
        for i, gender in enumerate(df['Sex'].unique()):
            plt.subplot(1, 2, i + 1)
            subset = df[df['Sex'] == gender]
            sns.barplot(x='PhysicalActivity', y='BMI', hue='HeartDisease', data=subset)
            plt.title(f'Physical Activity vs BMI - {gender}', fontsize=14, color='blue')
            plt.xlabel('Physical Activity', fontsize=12, color='darkred')
            plt.ylabel('BMI', fontsize=12, color='darkred')
            plt.legend(title='Heart Disease')
        plt.suptitle('Story: Physical Activity vs Heart Disease by Gender', fontsize=16, color='blue')
        plt.tight_layout()
        plt.savefig("subplot1_bar_gender.png")
        plt.show()
        plt.close()

    # 2. Storytelling Subplot - Count plots for Smoking, Alcohol, and Stroke
    risk_factors = [col for col in ['Smoking', 'AlcoholDrinking', 'Stroke'] if col in df.columns]
    if len(risk_factors) > 0 and 'HeartDisease' in df.columns:
        plt.figure(figsize=(18, 5))
        for i, col in enumerate(risk_factors):
            plt.subplot(1, len(risk_factors), i + 1)
            sns.countplot(x=col, hue='HeartDisease', data=df)
            plt.ylabel('Count', fontsize=12, color='darkred')
            plt.xlabel(col, fontsize=12, color='darkred')
            plt.legend(title='Heart Disease')
```

```
        plt.suptitle('Story: Risk Factors vs Heart Disease', fontsize=16, color='blue')
        plt.tight_layout()
        plt.show()
        plt.savefig("subplot2_count_risks.png")
        plt.close()

    # 3. Box plot comparing 'Sex' and BMI
    if 'Sex' in df.columns and 'BMI' in df.columns:
        plt.figure(figsize=(10, 6))
        sns.boxplot(x='Sex', y='BMI', hue='Sex', data=df, palette='pastel')
        plt.title('Box Plot: BMI by Gender', fontsize=16, color='blue')
        plt.xlabel('Gender', fontsize=14, color='darkred')
        plt.ylabel('BMI', fontsize=14, color='darkred')
        plt.tight_layout()
        plt.show()
        plt.savefig("boxplot_bmi_gender.png")
        plt.close()

    # 4. Joint Plot with KDE
    if 'BMI' in df.columns and 'SleepTime' in df.columns:
        plt.figure(figsize=(10, 8))
        g = sns.jointplot(x='BMI', y='SleepTime', data=df, kind='kde', fill=True, cmap='Blues')
        g.fig.suptitle('Joint KDE Plot: BMI vs Sleep Hours', fontsize=16, color='blue', y=1.02)
        g.fig.tight_layout()
        plt.show()
        plt.savefig("joint_kde_bmi_sleep.png")
        plt.close()

def create_static_visualizations(df):
    try:
        if len(df) > 5000:
            df = df.sample(5000, random_state=42)

        # Set base style for all plots
        plt.style.use('seaborn-v0_8')
        plt.rcParams.update({
            'font.family': 'serif',
            'axes.titlecolor': 'blue',
            'axes.labelcolor': 'darkred',
            'figure.figsize': (10, 6)
        })

        print("\nCreating all required static visualizations...")

        # Identify target column (heart disease)
        target_column = None
        for col in ['HeartDisease', 'HadHeartAttack']:
            if col in df.columns:
                target_column = col
                break

        # Find age column
        age_col = None
        for col in ['Age', 'AgeCategory']:
            if col in df.columns:
                age_col = col
                break

        # 1. Line Plot
```

```
if age_col and 'BMI' in df.columns:
    plt.figure(figsize=(10, 6))
    if age_col == 'AgeCategory':
        age_order = sorted(df[age_col].unique(),
                           key=lambda x: int(re.search(r'\d+', x.split('-')[0]).group()) if '-' in x else 0)
        avg_bmi = df.groupby(age_col)['BMI'].mean().reindex(age_order)
        plt.plot(avg_bmi.index, avg_bmi.values, marker='o', linewidth=2, color='navy')
    else:
        # For numerical age
        avg_bmi = df.groupby(age_col)['BMI'].mean()
        plt.plot(avg_bmi.index, avg_bmi.values, linewidth=2, color='navy')

    plt.title('Average BMI by Age', fontsize=16, color='blue')
    plt.xlabel('Age Category', fontsize=14, color='darkred')
    plt.ylabel('Average BMI', fontsize=14, color='darkred')
    plt.xticks(rotation=45)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    plt.savefig('line_plot_age_bmi.png')
    plt.close()
    print("Created Line Plot")

# 2. Bar Plot - Group
if 'Sex' in df.columns and target_column and 'BMI' in df.columns:
    plt.figure(figsize=(10, 6))
    grouped_data = df.groupby(['Sex', target_column])['BMI'].mean().unstack()
    ax = grouped_data.plot(kind='bar', width=0.7)
    plt.title('Average BMI by Gender and Heart Disease Status', fontsize=16, color='blue')
    plt.xlabel('Gender', fontsize=14, color='darkred')
    plt.ylabel('Average BMI', fontsize=14, color='darkred')
    plt.legend(title='Heart Disease')
    plt.grid(True, axis='y')

    # Add value labels on top of bars
    for container in ax.containers:
        ax.bar_label(container, fmt='%.2f')

    plt.tight_layout()
    plt.show()
    plt.savefig('grouped_bar_plot_bmi.png')
    plt.close()
    print("Created Grouped Bar Plot")

# 3. Bar Plot - Stacked
if age_col and target_column:
    plt.figure(figsize=(10, 6))
    # Group by age and count target variable
    if age_col == 'AgeCategory':
        age_order = sorted(df[age_col].unique(),
                           key=lambda x: int(re.search(r'\d+', x.split('-')[0]).group()) if '-' in x else 0)
        crosstab = pd.crosstab(df[age_col], df[target_column])
        crosstab = crosstab.reindex(age_order)
    else:
        age_bins = pd.cut(df[age_col], bins=6)
        crosstab = pd.crosstab(age_bins, df[target_column])

    # Create stacked bar plot
    ax = crosstab.plot(kind='bar', stacked=True)
```

```
    plt.title(f'Heart Disease by Age Category', fontsize=16, color='blue')
    plt.xlabel('Age Category', fontsize=14, color='darkred')
    plt.ylabel('Count', fontsize=14, color='darkred')
    plt.legend(title='Heart Disease')
    plt.grid(True, axis='y')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
    plt.savefig('stacked_bar_plot_age.png')
    plt.close()
    print("Created Stacked Bar Plot")

# 4. Count Plot
if 'Sex' in df.columns:
    plt.figure(figsize=(10, 6))
    ax = sns.countplot(x='Sex', data=df)
    plt.title('Distribution of Gender', fontsize=16, color='blue')
    plt.xlabel('Gender', fontsize=14, color='darkred')
    plt.ylabel('Count', fontsize=14, color='darkred')
    plt.grid(True, axis='y')
    for container in ax.containers:
        ax.bar_label(container)

    plt.tight_layout()
    plt.show()
    plt.savefig('count_plot_gender.png')
    plt.close()
    print("Created Count Plot")

# 5. Pie Chart
if target_column:
    plt.figure(figsize=(10, 8))
    target_counts = df[target_column].value_counts()
    explode = [0.1 if i == target_counts.idxmax() else 0 for i in target_counts.index]
    plt.pie(target_counts,
            labels=[str(label) for label in target_counts.index],
            autopct='%1.1f%%',
            colors=plt.cm.Paired.colors[:len(target_counts)],
            explode=explode,
            shadow=True,
            startangle=90,
            textprops={'fontsize': 12})
    plt.title('Heart Disease Distribution', fontsize=16, color='blue')
    plt.axis('equal')
    plt.tight_layout()
    plt.show()
    plt.savefig('pie_chart_heart_disease.png')
    plt.close()
    print("Created Pie Chart")

# 6. Dist Plot (Histogram with KDE)
if 'BMI' in df.columns:
    plt.figure(figsize=(10, 6))
    sns.histplot(x='BMI', data=df, kde=True, color='skyblue')
    plt.title('BMI Distribution', fontsize=16, color='blue')
    plt.xlabel('BMI', fontsize=14, color='darkred')
    plt.ylabel('Frequency', fontsize=14, color='darkred')
    plt.grid(True)
    plt.tight_layout()
```

```
    plt.show()
    plt.savefig('dist_plot_bmi.png')
    plt.close()
    print("Created Dist Plot")

# 7. KDE Plot (filled)
if 'BMI' in df.columns and target_column:
    plt.figure(figsize=(10, 6))
    for target_val, color, label in zip([0, 1], ['skyblue', 'tomato'], ['No Heart Disease', 'Heart Disease']):
        subset = df[df[target_column] == target_val]
        if not subset.empty:
            sns.kdeplot(
                x='BMI',
                data=subset,
                fill=True,
                alpha=0.6,
                linewidth=2,
                color=color,
                label=label
            )
    plt.title('BMI Distribution by Heart Disease Status', fontsize=16, color='blue')
    plt.xlabel('BMI', fontsize=14, color='darkred')
    plt.ylabel('Density', fontsize=14, color='darkred')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    plt.savefig('kde_plot_filled_bmi.png')
    plt.close()
    print("Created KDE Plot (filled)")

# 8. Pair Plot
subset_cols = [col for col in ['BMI', 'Age', 'SleepTime', 'PhysicalHealth'] if col in df.columns]
if len(subset_cols) >= 2 and target_column:
    subset_data = df[subset_cols + [target_column]].sample(min(1000, len(df)), random_state=42)
    subset_data[target_column] = subset_data[target_column].astype(str)

    pair_plot = sns.pairplot(
        data=subset_data,
        vars=subset_cols,
        hue=target_column,
        palette='coolwarm',
        diag_kind='kde',
        plot_kws={'alpha': 0.6, 's': 30, 'edgecolor': 'k', 'linewidth': 0.5},
        diag_kws={'fill': True, 'alpha': 0.5}
    )
    pair_plot.fig
    pair_plot = sns.pairplot(
        data=subset_data,
        vars=subset_cols,
        hue=target_column,
        palette='coolwarm',
        diag_kind='kde',
        plot_kws={'alpha': 0.6, 's': 30, 'edgecolor': 'k', 'linewidth': 0.5},
        diag_kws={'fill': True, 'alpha': 0.5}
    )
    pair_plot.fig.suptitle('Pair Plot of Key Features', fontsize=18, color='blue', y=1.02)
    plt.tight_layout()
    plt.show()
```

```
    plt.savefig('pair_plot.png')
    plt.close()
    print("Created Pair Plot")

# 9. Correlation Heatmap
num_cols = df.select_dtypes(include=['float64', 'int64']).columns
if len(num_cols) >= 2:
    plt.figure(figsize=(12, 10))
    corr_matrix = df[num_cols].corr()
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    sns.heatmap(
        corr_matrix,
        mask=mask,
        annot=True,
        fmt='.2f',
        cmap='coolwarm',
        linewidths=0.5,
        cbar_kws={'label': 'Correlation Coefficient'}
    )
    plt.title('Correlation Heatmap', fontsize=16, color='blue')
    plt.tight_layout()
    plt.show()
    plt.savefig('correlation_heatmap.png')
    plt.close()
    print("Created Correlation Heatmap")

# 10. Box Plot
if 'PhysicalHealth' in df.columns and 'Sex' in df.columns:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='Sex', y='PhysicalHealth', data=df)
    plt.title('Physical Health by Gender', fontsize=16, color='blue')
    plt.xlabel('Gender', fontsize=14, color='darkred')
    plt.ylabel('Physical Health', fontsize=14, color='darkred')
    plt.grid(True, axis='y')
    plt.tight_layout()
    plt.show()
    plt.savefig('box_plot_physicalhealth.png')
    plt.close()
    print("Created Box Plot")

# 11. Violin Plot
if 'SleepTime' in df.columns and target_column:
    plt.figure(figsize=(10, 6))
    sns.violinplot(x=target_column, y='SleepTime', data=df, inner='quartile')
    plt.title('Sleep Time Distribution by Heart Disease Status', fontsize=16, color='blue')
    plt.xlabel('Heart Disease', fontsize=14, color='darkred')
    plt.ylabel('Sleep Time (hours)', fontsize=14, color='darkred')
    plt.grid(True, axis='y')
    plt.tight_layout()
    plt.show()
    plt.savefig('violin_plot_sleeptime.png')
    plt.close()
    print("Created Violin Plot")

# 12. Regression Plot
if 'BMI' in df.columns and 'PhysicalHealth' in df.columns:
    plt.figure(figsize=(10, 6))
    sns.regplot(x='BMI', y='PhysicalHealth', data=df, scatter_kws={'alpha': 0.5}, line_kws={'color': 'red'})
    plt.title('Relationship between BMI and Physical Health', fontsize=16, color='blue')
```

```
    plt.xlabel('BMI', fontsize=14, color='darkred')
    plt.ylabel('Physical Health', fontsize=14, color='darkred')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    plt.savefig('regression_plot_bmi_health.png')
    plt.close()
    print("Created Regression Plot")

# 13. Categorical Point Plot with Confidence Intervals
if age_col and 'BMI' in df.columns and target_column:
    plt.figure(figsize=(10, 6))
    if age_col == 'AgeCategory':
        age_order = sorted(df[age_col].unique(),
                           key=lambda x: int(re.search(r'\d+', x.split('-')[0]).group()) if '-' in x else 0)
        sns.pointplot(x=age_col, y='BMI', hue=target_column, data=df,
                      palette='Set2', order=age_order,
                      dodge=True, errorbar=('ci', 95), err_kws={'linewidth': 2})
    else:
        # For numerical age, bin it
        df['AgeBin'] = pd.cut(df[age_col], bins=6)
        sns.pointplot(x='AgeBin', y='BMI', hue=target_column, data=df,
                      palette='Set2', dodge=True, ci=95, errwidth=2)
    plt.title('Average BMI by Age Groups with 95% CI', fontsize=16, color='blue')
    plt.xlabel('Age Group', fontsize=14, color='darkred')
    plt.ylabel('Average BMI', fontsize=14, color='darkred')
    plt.legend(title='Heart Disease')
    plt.grid(True, axis='y')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
    plt.savefig('point_plot_age_bmi.png')
    plt.close()
    print("Created Point Plot")

# 14. Swarm Plot
if 'MentalHealth' in df.columns and target_column:
    plt.figure(figsize=(10, 6))
    sample_df = df.sample(min(1000, len(df)), random_state=42)
    sns.swarmplot(x=target_column, y='MentalHealth', data=sample_df)
    plt.title('Mental Health by Heart Disease Status', fontsize=16, color='blue')
    plt.xlabel('Heart Disease', fontsize=14, color='darkred')
    plt.ylabel('Mental Health', fontsize=14, color='darkred')
    plt.grid(True, axis='y')
    plt.tight_layout()
    plt.show()
    plt.savefig('swarm_plot_mentalhealth.png')
    plt.close()
    print("Created Swarm Plot")

# 15. Joint Plot
if 'BMI' in df.columns and 'PhysicalHealth' in df.columns:
    sample_df = df.sample(min(2000, len(df)), random_state=42)
    joint_plot = sns.jointplot(
        x='BMI',
        y='PhysicalHealth',
        data=sample_df,
        kind='scatter',
        color='purple',
```

```
            height=8,
            ratio=5,
            marginal_kws=dict(bins=20, fill=True)
        )
        joint_plot.fig.suptitle('Joint Distribution of BMI and Physical Health', fontsize=16, color='blue', y=1.02)
        joint_plot.fig.tight_layout()
        plt.show()
        plt.savefig('joint_plot_bmi_health.png')
        plt.close()
        print("Created Joint Plot")

    # 16. Strip Plot Facet Grid
    if 'SleepTime' in df.columns and 'Sex' in df.columns and target_column:
        plt.figure(figsize=(12, 6))
        # Split by gender and heart disease
        g = sns.FacetGrid(df, col='Sex', hue=target_column, height=5, aspect=1)
        g.map(sns.stripplot, 'SleepTime', alpha=0.5, jitter=True)
        g.add_legend(title='Heart Disease')
        g.fig.suptitle('Sleep Time Distribution by Gender and Heart Disease', fontsize=16, color='blue', y=1.05)
        plt.tight_layout()
        plt.show()
        plt.savefig('strip_plot_facet_sleep.png')
        plt.close()
        print("Created Strip Plot Facet Grid")

    # 17. Facet Grid Histogram
    if 'BMI' in df.columns and 'Sex' in df.columns and target_column:
        g = sns.FacetGrid(df, col=target_column, row='Sex', height=4, aspect=1.5)
        g.map(sns.histplot, 'BMI', kde=True, fill=True, alpha=0.6)
        g.set_axis_labels('BMI', 'Count')
        g.set_titles(col_template='{col_name} Heart Disease', row_template='{row_name}')
        g.fig.suptitle('BMI Distribution by Gender and Heart Disease', fontsize=16, color='blue', y=1.05)
        plt.tight_layout()
        plt.show()
        plt.savefig('facet_grid_hist_bmi.png')
        plt.close()
        print("Created Facet Grid Histogram")

    # Create additional specialized plots
    create_subplots(df)

    print("\nAll visualizations created successfully!")
    return True

  except Exception as e:
    print(f"\nError during visualization creation: {str(e)}")
    import traceback
    traceback.print_exc()
    return False


def main():
  print("Heart Disease Data Analysis Pipeline - Phase 1")
  print("=" * 50)

  try:
    # 1. Load and clean data
    data = load_and_clean_data()
    if data is None:
```

```
            print("Could not load data. Exiting...")
            return

        # 2. Detect and remove outliers
        data_no_outliers = detect_and_remove_outliers(data, method='IQR')

        # 3. Check normality of data
        normality_results, qq_plots = check_normality(data_no_outliers)

        # 4. Perform data transformation
        data_transformed, transform_results = perform_data_transformation(data_no_outliers)

        # 5. Perform PCA analysis
        pca_results = perform_pca_analysis(data_no_outliers)

        # 6. Create static visualizations
        create_static_visualizations(data_no_outliers)

        # 7. Create subplots for storytelling
        create_subplots(data_no_outliers)

        print("\nData analysis complete! All visualizations have been saved.")

    except Exception as e:
        print(f"\nError in main function: {str(e)}")

#%%
#%%
## Phase - 2 ##
#%%
#%%
import dash
from dash import Dash, html, dcc, Input, Output, dash_table
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output, State
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np
from scipy import stats
from scipy.special import boxcox
from io import BytesIO
import base64
from prettytable import PrettyTable
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer

# Initialize the Dash app
app = Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])
server = app.server

# Load the cleaned data (assuming it's already processed from Phase 1)
df = pd.read_csv("heart_2020_cleaned.csv")

# Generate a dictionary of available columns by type
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
```

```
binary_cols = [col for col in numerical_cols if df[col].nunique() <= 2]
numerical_cols = [col for col in numerical_cols if col not in binary_cols]
categorical_cols = categorical_cols + binary_cols

# Generate the options for dropdowns
numerical_options = [{'label': col, 'value': col} for col in numerical_cols]
categorical_options = [{'label': col, 'value': col} for col in categorical_cols]

# Create list of all available graph types
graph_types = [
    {'label': 'Bar Plot', 'value': 'bar'},
    {'label': 'Line Plot', 'value': 'line'},
    {'label': 'Scatter Plot', 'value': 'scatter'},
    {'label': 'Box Plot', 'value': 'box'},
    {'label': 'Violin Plot', 'value': 'violin'},
    {'label': 'Histogram', 'value': 'histogram'},
    {'label': 'Pie Chart', 'value': 'pie'},
    {'label': 'Heatmap', 'value': 'heatmap'},
    {'label': 'Density Plot', 'value': 'density'},
    {'label': 'Strip Plot', 'value': 'strip'},
    {'label': 'Swarm Plot', 'value': 'swarm'},
    {'label': 'Count Plot', 'value': 'count'},
    {'label': '3D Scatter', 'value': '3d-scatter'}
]

# Find target column (heart disease)
target_column = None
for col in ['HeartDisease', 'HadHeartAttack']:
    if col in df.columns:
        target_column = col
        break

# Create the layout
app.layout = dbc.Container([
    dbc.Row([
        dbc.Col([
            html.H1("Heart Disease Risk Analysis Dashboard",
                    style={'textAlign': 'center', 'color': 'blue', 'fontFamily': 'serif'}),
            html.Hr()
        ]
        )
    ]),

    # Tabs for different sections
    dcc.Tabs([
        # Tab 1: Data Overview
        dcc.Tab(label='Data Overview', children=[
            dbc.Row([
                dbc.Col([
                    html.H3("Dataset Information",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    html.Div(id='data-info'),
                    html.Hr(),
                    html.H4("Data Preview",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    dash_table.DataTable(
                        id='data-table',
                        columns=[{"name": col, "id": col} for col in df.columns],
                        data=df.head(10).to_dict('records'),
```

```
                style_table={'overflowX': 'auto'},
                style_cell={'textAlign': 'left', 'padding': '5px'},
                style_header={
                    'backgroundColor': 'rgb(230, 230, 230)',
                    'fontWeight': 'bold'
                }
            ),
            html.Hr(),
            html.Div([
                html.H4("Download Data",
                        style={'color': 'darkred', 'fontFamily': 'serif'}),
                dcc.Download(id="download-dataframe-csv"),
                html.Button("Download CSV", id="btn-download-csv"),
            ]),
        ])
    ])
]),

# Tab 2: Data Cleaning
dcc.Tab(label='Data Cleaning', children=[
    dbc.Row([
        dbc.Col([
            html.H3("Data Cleaning Options",
                    style={'color': 'darkred', 'fontFamily': 'serif'}),
            dbc.Card([
                dbc.CardBody([
                    html.H5("Handle Missing Values",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    dcc.RadioItems(
                        id='missing-values-method',
                        options=[
                            {'label': 'Drop rows with missing values', 'value': 'drop'},
                            {'label': 'Fill with mean', 'value': 'mean'},
                            {'label': 'Fill with median', 'value': 'median'},
                            {'label': 'Fill with mode', 'value': 'mode'}
                        ],
                        value='drop',
                        inline=True
                    ),
                    html.Hr(),
                    html.H5("Handle Duplicate Rows",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    dcc.RadioItems(
                        id='duplicate-method',
                        options=[
                            {'label': 'Drop duplicates', 'value': 'drop'},
                            {'label': 'Keep duplicates', 'value': 'keep'}
                        ],
                        value='drop',
                        inline=True
                    ),
                    html.Button('Apply Cleaning', id='apply-cleaning-btn',
                                className='btn btn-primary mt-3')
                ])
            ]),
            html.Hr(),
            html.Div(id='cleaning-results')
        ]),
    ])
```

```
    ]),

    # Tab 3: Outlier Detection
    dcc.Tab(label='Outlier Detection', children=[
        dbc.Row([
            dbc.Col([
                html.H3("Outlier Detection and Removal",
                        style={'color': 'darkred', 'fontFamily': 'serif'}),
                dbc.Card([
                    dbc.CardBody([
                        html.H5("Select Outlier Detection Method",
                                style={'color': 'darkred', 'fontFamily': 'serif'}),
                        dcc.RadioItems(
                            id='outlier-method',
                            options=[
                                {'label': 'IQR Method', 'value': 'IQR'},
                                {'label': 'Z-Score Method', 'value': 'Z-score'},
                                {'label': 'Modified Z-Score Method', 'value': 'Modified Z-score'}
                            ],
                            value='IQR',
                            inline=True
                        ),
                        html.Button('Detect Outliers', id='detect-outliers-btn',
                                    className='btn btn-primary mt-3')
                    ])
                ]),
                html.Hr(),
                dcc.Loading(
                    id="loading-outliers",
                    type="circle",
                    children=[html.Div(id='outlier-results')]
                )
            ]),
        ])
    ]),

    # Tab 4: Data Transformation
    dcc.Tab(label='Data Transformation', children=[
        dbc.Row([
            dbc.Col([
                html.H3("Data Transformation",
                        style={'color': 'darkred', 'fontFamily': 'serif'}),
                dbc.Card([
                    dbc.CardBody([
                        html.H5("Select Transformation Method",
                                style={'color': 'darkred', 'fontFamily': 'serif'}),
                        dcc.RadioItems(
                            id='transform-method',
                            options=[
                                {'label': 'Log Transformation', 'value': 'log'},
                                {'label': 'Square Root Transformation', 'value': 'sqrt'},
                                {'label': 'Box-Cox Transformation', 'value': 'boxcox'},
                                {'label': 'Z-Score Standardization', 'value': 'zscore'}
                            ],
                            value='log',
                            inline=True
                        ),
                        html.Hr(),
                        html.H5("Select Numerical Features",
```

```
                        style={'color': 'darkred', 'fontFamily': 'serif'}),
                    dcc.Checklist(
                        id='transform-features',
                        options=[{'label': col, 'value': col} for col in numerical_cols],
                        value=[numerical_cols[0]] if numerical_cols else [],
                        inline=True
                    ),
                    html.Button('Apply Transformation', id='apply-transform-btn',
                                classNname='btn btn-primary mt-3')
                ])
            ]),
            html.Hr(),
            dcc.Loading(
                id="loading-transform",
                type="circle",
                children=[html.Div(id='transform-results')]
            )
        ]),
    ])
]),

# Tab 5: Normality Tests
dcc.Tab(label='Normality Tests', children=[
    dbc.Row([
        dbc.Col([
            html.H3("Normality Tests",
                    style={'color': 'darkred', 'fontFamily': 'serif'}),
            dbc.Card([
                dbc.CardBody([
                    html.H5("Select Normality Test",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    dcc.RadioItems(
                        id='normality-test',
                        options=[
                            {'label': "D'Agostino's K^2 Test", 'value': 'dagostino'},
                            {'label': 'Shapiro-Wilk Test', 'value': 'shapiro'},
                            {'label': 'Anderson-Darling Test', 'value': 'anderson'}
                        ],
                        value='shapiro',
                        inline=True
                    ),
                    html.Hr(),
                    html.H5("Select Numerical Feature",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    dcc.Dropdown(
                        id='normality-feature',
                        options=[{'label': col, 'value': col} for col in numerical_cols],
                        value=numerical_cols[0] if numerical_cols else None
                    ),
                    html.Button('Test Normality', id='test-normality-btn',
                                classNname='btn btn-primary mt-3')
                ])
            ]),
            html.Hr(),
            dcc.Loading(
                id="loading-normality",
                type="circle",
                children=[
                    html.Div(id='normality-results'),
```

```
                    html.Div(id='qq-plot')
                ]
            )
        ]),
    ])
]),

# Tab 6: PCA Analysis
dcc.Tab(label='PCA Analysis', children=[
    dbc.Row([
        dbc.Col([
            html.H3("Principal Component Analysis (PCA)",
                    style={'color': 'darkred', 'fontFamily': 'serif'}),
            dbc.Card([
                dbc.CardBody([
                    html.H5("PCA Configuration",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    html.Label("Number of Components:"),
                    dcc.Slider(
                        id='pca-components',
                        min=2,
                        max=min(10, len(numerical_cols)),
                        value=2,
                        marks={i: str(i) for i in range(2, min(10, len(numerical_cols)) + 1)},
                        step=1
                    ),
                    html.Button('Perform PCA', id='run-pca-btn',
                                className='btn btn-primary mt-3')
                ])
            ]),
            html.Hr(),
            dcc.Loading(
                id="loading-pca",
                type="circle",
                children=[
                    html.Div(id='pca-results'),
                    dcc.Graph(id='pca-plot'),
                    html.Div(id='explained-variance'),
                    html.Div(id='condition-number')
                ]
            )
        ]),
    ])
]),

# Tab 7: Interactive Plots
dcc.Tab(label='Interactive Plots', children=[
    dbc.Row([
        dbc.Col([
            html.H3("Interactive Data Visualization",
                    style={'color': 'darkred', 'fontFamily': 'serif'}),
            dbc.Card([
                dbc.CardBody([
                    html.H5("Plot Configuration",
                            style={'color': 'darkred', 'fontFamily': 'serif'}),
                    html.Label("Select Plot Type:"),
                    dcc.Dropdown(
                        id='plot-type',
                        options=graph_types,
```

```
                        value='bar'
                    ),
                    html.Div(id='feature-selection', children=[
                        html.Label("Select X-axis Feature:"),
                        dcc.Dropdown(
                            id='x-feature',
                            options=categorical_options + numerical_options,
                            value=categorical_cols[0] if categorical_cols else numerical_cols[0]
                        ),
                        html.Label("Select Y-axis Feature:"),
                        dcc.Dropdown(
                            id='y-feature',
                            options=numerical_options,
                            value=numerical_cols[0] if numerical_cols else None
                        ),
                        html.Label("Select Color Variable (optional):"),
                        dcc.Dropdown(
                            id='color-feature',
                            options=[{'label': 'None', 'value': 'none'}] + categorical_options,
                            value='none'
                        ),
                        html.Div(id='z-feature-container', children=[
                            html.Label("Select Z-axis Feature (for 3D):"),
                            dcc.Dropdown(
                                id='z-feature',
                                options=numerical_options,
                                value=numerical_cols[1] if len(numerical_cols) > 1 else numerical_cols[0]
                            )
                        ], style={'display': 'none'})
                    ]),
                    html.Hr(),
                    html.Label("Additional Options:"),
                    dcc.Checklist(
                        id='plot-options',
                        options=[
                            {'label': 'Show Grid', 'value': 'grid'},
                            {'label': 'Show Trend Line', 'value': 'trend'},
                            {'label': 'Log Scale', 'value': 'log'}
                        ],
                        value=['grid'],
                        inline=True
                    )
                ])
            ]),
        ], width=3),
        dbc.Col(children=[
            dcc.Loading(
                id="loading-plot",
                type="circle",
                children=[dcc.Graph(id='interactive-plot')]
            )
        ], width=9)
    ])
  ])
], fluid=True)


# ==========================================
```

```
# CALLBACK FUNCTIONS
# ============================================

@app.callback(
    Output('data-info', 'children'),
    Input('data-table', 'data')
)
def update_data_info(data):
    """Update data info section with dataset metadata"""
    # Create a card with dataset information
    info_card = dbc.Card([
        dbc.CardBody([
            html.H5("Dataset Summary"),
            html.P(f"Total Records: {len(df)}"),
            html.P(f"Features: {len(df.columns)}"),
            html.P(f"Target Variable: {target_column if target_column else 'N/A'}"),
            html.P(f"Numerical Features: {len(numerical_cols)}"),
            html.P(f"Categorical Features: {len(categorical_cols)}")
        ])
    ])
    return info_card


@app.callback(
    Output("download-dataframe-csv", "data"),
    Input("btn-download-csv", "n_clicks"),
    prevent_initial_call=True
)
def download_csv(n_clicks):
    return dcc.send_data_frame(df.to_csv, "heart_disease_data.csv", index=False)


@app.callback(
    Output('cleaning-results', 'children'),
    Input('apply-cleaning-btn', 'n_clicks'),
    [State('missing-values-method', 'value'),
     State('duplicate-method', 'value')]
)
def apply_cleaning(n_clicks, missing_method, duplicate_method):
    if n_clicks is None:
        return html.Div()
    cleaned_df = df.copy()
    original_rows = len(cleaned_df)

    # Handle missing values
    if missing_method == 'drop':
        cleaned_df = cleaned_df.dropna()
    elif missing_method == 'mean':
        imputer = SimpleImputer(strategy='mean')
        numerical_data = cleaned_df[numerical_cols].values
        imputed_data = imputer.fit_transform(numerical_data)
        cleaned_df[numerical_cols] = imputed_data
    elif missing_method == 'median':
        imputer = SimpleImputer(strategy='median')
        numerical_data = cleaned_df[numerical_cols].values
        imputed_data = imputer.fit_transform(numerical_data)
        cleaned_df[numerical_cols] = imputed_data
    elif missing_method == 'mode':
        for col in cleaned_df.columns:
```

```
            if col in numerical_cols:
                cleaned_df[col].fillna(cleaned_df[col].mode()[0], inplace=True)
            else:
                cleaned_df[col].fillna(cleaned_df[col].mode()[0], inplace=True)

    # Handle duplicates
    if duplicate_method == 'drop':
        cleaned_df = cleaned_df.drop_duplicates()

    cleaned_rows = len(cleaned_df)
    removed_rows = original_rows - cleaned_rows

    # Create results card
    results_card = dbc.Card([
        dbc.CardHeader("Cleaning Results"),
        dbc.CardBody([
            html.P(f"Original Dataset Size: {original_rows} rows"),
            html.P(f"Cleaned Dataset Size: {cleaned_rows} rows"),
            html.P(f"Removed {removed_rows} rows ({(removed_rows / original_rows * 100):.2f}%)"),
            html.Hr(),
            html.H6("Missing Values Handling:"),
            html.P(f"Method: {missing_method}"),
            html.H6("Duplicate Rows Handling:"),
            html.P(f"Method: {duplicate_method}")
        ])
    ])

    return results_card

@app.callback(
    Output('outlier-results', 'children'),
    Input('detect-outliers-btn', 'n_clicks'),
    State('outlier-method', 'value')
)
def detect_outliers(n_clicks, method):
    if n_clicks is None:
        return html.Div()

    results = []
    try:
        # Create table to display outlier information
        outlier_table = PrettyTable()
        outlier_table.field_names = ["Column", "Outliers Count", "Lower Bound", "Upper Bound"]

        # Check outliers in numerical columns
        for col in numerical_cols:
            if method == 'IQR':
                Q1 = df[col].quantile(0.25)
                Q3 = df[col].quantile(0.75)
                IQR = Q3 - Q1
                lower = Q1 - 1.5 * IQR
                upper = Q3 + 1.5 * IQR
                outliers = df[(df[col] < lower) | (df[col] > upper)]
                count = len(outliers)
            elif method == 'Z-score':
                z_scores = abs(stats.zscore(df[col].dropna()))
                threshold = 3
                outliers = df[abs(stats.zscore(df[col].fillna(df[col].mean()))) > threshold]
                count = len(outliers)
```

```
            mean = df[col].mean()
            std = df[col].std()
            lower = mean - threshold * std
            upper = mean + threshold * std
        elif method == 'Modified Z-score':
            median = df[col].median()
            MAD = np.median(np.abs(df[col] - median))
            if MAD == 0:
                MAD = 0.1
            modified_z_scores = 0.6745 * (df[col] - median) / MAD
            threshold = 3.5
            outliers = df[abs(modified_z_scores) > threshold]
            count = len(outliers)
            lower = median - threshold * MAD / 0.6745
            upper = median + threshold * MAD / 0.6745

        outlier_table.add_row([col, count, f"{lower:.2f}", f"{upper:.2f}"])
    table_html = outlier_table.get_html_string()

    # Create a results card
    results_card = dbc.Card([
        dbc.CardHeader(f"Outlier Detection Results ({method})"),
        dbc.CardBody([
            html.Div([
                html.P(
                    f"Outlier detection using {method} method identified potential outliers in the dataset."),
                html.Hr(),
                html.Iframe(srcDoc=table_html,
                            style={'width': '100%', 'height': '400px', 'border': '1px solid #ccc'})
            ])
        ])
    ])

    results.append(results_card)

    # Create boxplots for visualization
    fig = go.Figure()
    for col in numerical_cols[:5]:
        fig.add_trace(go.Box(
            y=df[col],
            name=col,
            boxpoints='outliers',
            marker=dict(color='rgb(8,81,156)'),
            boxmean=True
        ))

    fig.update_layout(
        title_text='Boxplot for Outlier Visualization',
        yaxis_title='Values',
        showlegend=False
    )

    results.append(dcc.Graph(figure=fig))

except Exception as e:
    results.append(html.Div(f"Error detecting outliers: {str(e)}"))

return html.Div(results)
```

```
@app.callback(
    Output('transform-results', 'children'),
    Input('apply-transform-btn', 'n_clicks'),
    [State('transform-method', 'value'),
     State('transform-features', 'value')]
)
def apply_transformation(n_clicks, method, features):
    if n_clicks is None or not features:
        return html.Div()

    results = []
    try:
        # Apply transformation
        df_transformed = df.copy()
        transformation_results = {}

        for col in features:
            # Skip binary columns
            if col in binary_cols:
                continue

            try:
                data = df[col].dropna()

                # Handle non-positive values for log and boxcox
                if method in ['log', 'boxcox']:
                    if data.min() <= 0:
                        shift_value = abs(data.min()) + 1
                        data = data + shift_value

                if method == 'log':
                    df_transformed[f"{col}_log"] = np.log(data)
                    transformation_results[col] = {'method': 'log', 'new_column': f"{col}_log"}

                elif method == 'sqrt':
                    if data.min() >= 0:
                        df_transformed[f"{col}_sqrt"] = np.sqrt(data)
                        transformation_results[col] = {'method': 'sqrt', 'new_column': f"{col}_sqrt"}

                elif method == 'boxcox':
                    try:
                        transformed_data, lambda_val = stats.boxcox(data)
                        df_transformed[f"{col}_boxcox"] = transformed_data
                        transformation_results[col] = {
                            'method': 'boxcox',
                            'lambda': lambda_val,
                            'new_column': f"{col}_boxcox"
                        }
                    except Exception as e:
                        # Skip if boxcox fails
                        print(f"BoxCox failed for {col}: {str(e)}")

                elif method == 'zscore':
                    df_transformed[f"{col}_zscore"] = (data - data.mean()) / data.std()
                    transformation_results[col] = {'method': 'zscore', 'new_column': f"{col}_zscore"}

            except Exception as e:
                print(f"Error transforming {col} with {method}: {e}")
```

```
        # Create visualization of original vs transformed
        figures = []
        for col in features:
            if col in transformation_results:
                new_col = transformation_results[col]['new_column']
                if new_col in df_transformed.columns:
                    # Create subplot with original and transformed data
                    fig = make_subplots(rows=1, cols=2,
                                        subplot_titles=(f'Original {col}', f'Transformed {col} ({method})'))

                    # Add original data histogram
                    fig.add_trace(
                        go.Histogram(x=df[col].dropna(), name='Original', marker_color='blue', opacity=0.7),
                        row=1, col=1
                    )

                    # Add transformed data histogram
                    fig.add_trace(
                        go.Histogram(x=df_transformed[new_col].dropna(), name='Transformed',
                                     marker_color='red', opacity=0.7),
                        row=1, col=2
                    )

                    fig.update_layout(height=400, title_text=f"Transformation of {col}")
                    figures.append(dcc.Graph(figure=fig))

        # Create summary card
        if transformation_results:
            info_card = dbc.Card([
                dbc.CardHeader(f"Data Transformation Results ({method})"),
                dbc.CardBody([
                    html.P(f"Applied {method} transformation to {len(transformation_results)} feature(s)."),
                    html.P("New columns have been created with the transformed values."),
                    html.Hr(),
                    html.Div([
                        html.H6("Transformed Features:"),
                        html.Ul([html.Li(f"{col} → {transformation_results[col]['new_column']}")
                                 for col in transformation_results])
                    ])
                ])
            ])
            results.append(info_card)
            results.extend(figures)
        else:
            results.append(html.Div(
                "No transformations could be applied. This may be due to non-positive values in the data for log or boxcox
transformations."))

    except Exception as e:
        results.append(html.Div(f"Error applying transformation: {str(e)}"))

    return html.Div(results)

@app.callback(
    Output('normality-results', 'children'),
    Output('qq-plot', 'children'),
    Input('test-normality-btn', 'n_clicks'),
    [State('normality-test', 'value'),
     State('normality-feature', 'value')]
```

```
)
def test_normality(n_clicks, test_type, feature):
    if n_clicks is None or feature is None:
        return html.Div(), html.Div()

    try:
        # Get the data without NaN values
        data = df[feature].dropna()

        # Perform selected normality test
        if test_type == 'shapiro':
            stat, p_value = stats.shapiro(data)
            test_name = "Shapiro-Wilk"
        elif test_type == 'dagostino':
            stat, p_value = stats.normaltest(data)
            test_name = "D'Agostino's K^2"
        elif test_type == 'anderson':
            result = stats.anderson(data, dist='norm')
            stat = result.statistic
            critical_values = result.critical_values
            significance_levels = [15., 10., 5., 2.5, 1.]
            test_name = "Anderson-Darling"
            # For Anderson test we handle results differently
            anderson_results = []
            for sl, cv in zip(significance_levels, critical_values):
                if result.statistic > cv:
                    anderson_results.append(
                        f"At {sl}% significance level: Reject normality (statistic > critical value)")
                else:
                    anderson_results.append(
                        f"At {sl}% significance level: Cannot reject normality (statistic <= critical value)")

        # Create histogram with normal curve overlay
        figure = go.Figure()
        figure.add_trace(go.Histogram(
            x=data,
            histnorm='probability density',
            name='Observed',
            opacity=0.75
        ))

        # Calculate normal distribution curve
        x = np.linspace(min(data), max(data), 100)
        y = stats.norm.pdf(x, data.mean(), data.std())
        figure.add_trace(go.Scatter(
            x=x,
            y=y,
            mode='lines',
            name='Normal Distribution',
            line=dict(color='red', width=2)
        ))

        figure.update_layout(
            title=f"Distribution of {feature}",
            xaxis_title=feature,
            yaxis_title="Density",
            barmode='overlay'
        )
```

```
    # Create QQ plot
    qq_fig = go.Figure()
    sorted_data = np.sort(data)
    theoretical_quantiles = stats.norm.ppf(np.linspace(0.001, 0.999, len(sorted_data)))

    # Add scatter plot
    qq_fig.add_trace(go.Scatter(
        x=theoretical_quantiles,
        y=sorted_data,
        mode='markers',
        name='QQ Plot',
        marker=dict(color='blue')
    ))

    min_val = min(min(theoretical_quantiles), min(sorted_data))
    max_val = max(max(theoretical_quantiles), max(sorted_data))
    line_vals = np.linspace(min_val, max_val, 100)

    qq_fig.add_trace(go.Scatter(
        x=line_vals,
        y=line_vals,
        mode='lines',
        name='Reference Line',
        line=dict(color='red', dash='dash')
    ))

    qq_fig.update_layout(
        title=f"QQ Plot for {feature}",
        xaxis_title="Theoretical Quantiles",
        yaxis_title="Sample Quantiles"
    )

    # Create results card
    if test_type == 'anderson':
        results_card = dbc.Card([
            dbc.CardHeader(f"Normality Test Results - {test_name}"),
            dbc.CardBody([
                html.P(f"Feature: {feature}"),
                html.P(f"Test Statistic: {stat:.4f}"),
                html.P("Critical Values at Different Significance Levels:"),
                html.Ul([html.Li(anderson_results[i]) for i in range(len(anderson_results))])
            ])
        ])
    else:
        # Interpret p-value
        alpha = 0.05
        if p_value > alpha:
            interpretation = f"Data appears to be normally distributed (fail to reject H0, p > {alpha})"
        else:
            interpretation = f"Data does not appear to be normally distributed (reject H0, p <= {alpha})"

        results_card = dbc.Card([
            dbc.CardHeader(f"Normality Test Results - {test_name}"),
            dbc.CardBody([
                html.P(f"Feature: {feature}"),
                html.P(f"Test Statistic: {stat:.4f}"),
                html.P(f"P-value: {p_value:.4f}"),
                html.P(f"Interpretation: {interpretation}"),
                html.P("Note: H0 = The data is normally distributed")
```

```
                ])
            ])

        return results_card, [dcc.Graph(figure=figure), dcc.Graph(figure=qq_fig)]

    except Exception as e:
        error_div = html.Div(f"Error performing normality test: {str(e)}")
        return error_div, html.Div()


@app.callback(
    [Output('pca-results', 'children'),
     Output('pca-plot', 'figure'),
     Output('explained-variance', 'children'),
     Output('condition-number', 'children')],
    Input('run-pca-btn', 'n_clicks'),
    State('pca-components', 'value')
)
def run_pca(n_clicks, n_components):
    if n_clicks is None:
        fig = go.Figure()
        fig.update_layout(title="No PCA analysis performed yet")
        return html.Div(), fig, html.Div(), html.Div()

    try:
        X = df[numerical_cols].copy()
        # Handle missing values with mean imputation
        imputer = SimpleImputer(strategy='mean')
        X_imputed = imputer.fit_transform(X)
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X_imputed)

        # Apply PCA
        pca = PCA(n_components=n_components)
        X_pca = pca.fit_transform(X_scaled)
        pca_cols = [f"PC{i + 1}" for i in range(n_components)]
        pca_df = pd.DataFrame(data=X_pca, columns=pca_cols)

        # Add target column if it exists
        if target_column in df.columns:
            pca_df[target_column] = df[target_column].values
        explained_variance = pca.explained_variance_ratio_ * 100
        cumulative_variance = np.cumsum(explained_variance)
        loadings = pca.components_.T
        loadings_df = pd.DataFrame(loadings, columns=pca_cols, index=numerical_cols)
        cond_num = np.linalg.cond(X_scaled)

        # Create scatterplot for first two PCs
        if target_column and target_column in df.columns:
            fig = px.scatter(
                pca_df, x="PC1", y="PC2",
                color=target_column,
                title="PCA Visualization",
                labels={"PC1": f"PC1 ({explained_variance[0]:.2f}%)",
                        "PC2": f"PC2 ({explained_variance[1]:.2f}%)"}
            )
        else:
            fig = px.scatter(
                pca_df, x="PC1", y="PC2",
```

```
            title="PCA Visualization",
            labels={"PC1": f"PC1 ({explained_variance[0]:.2f}%)",
                    "PC2": f"PC2 ({explained_variance[1]:.2f}%)"}
        )

        # Create results card
        results_card = dbc.Card([
            dbc.CardHeader("PCA Analysis Results"),
            dbc.CardBody([
                html.P(f"Applied PCA with {n_components} components"),
                html.P("Top 5 PC1 Contributing Features:"),
                html.Ul([
                    html.Li(f"{feature}: {abs(loadings_df['PC1'][feature]):.4f}")
                    for feature in loadings_df['PC1'].abs().sort_values(ascending=False).index[:5]
                ]),
                html.P("Top 5 PC2 Contributing Features:"),
                html.Ul([
                    html.Li(f"{feature}: {abs(loadings_df['PC2'][feature]):.4f}")
                    for feature in loadings_df['PC2'].abs().sort_values(ascending=False).index[:5]
                ])
            ])
        ])

        # Create explained variance plot
        variance_fig = go.Figure()
        variance_fig.add_trace(go.Bar(
            x=[f"PC{i + 1}" for i in range(len(explained_variance))],
            y=explained_variance,
            name="Explained Variance"
        ))
        variance_fig.add_trace(go.Scatter(
            x=[f"PC{i + 1}" for i in range(len(cumulative_variance))],
            y=cumulative_variance,
            name="Cumulative Variance",
            line=dict(color='red', width=2),
            mode='lines+markers'
        ))
        variance_fig.update_layout(
            title="Explained Variance by Principal Components",
            xaxis_title="Principal Components",
            yaxis_title="Explained Variance (%)",
            yaxis=dict(tickformat=".2f")
        )

        # Create condition number card
        condition_card = dbc.Card([
            dbc.CardHeader("Matrix Condition Analysis"),
            dbc.CardBody([
                html.P(f"Condition Number: {cond_num:.2f}"),
                html.P("Note: A high condition number (>30) indicates multicollinearity issues.")
            ])
        ])

        return results_card, fig, dcc.Graph(figure=variance_fig), condition_card

    except Exception as e:
        error_div = html.Div(f"Error performing PCA: {str(e)}")
        empty_fig = go.Figure()
        empty_fig.update_layout(title=f"Error: {str(e)}")
```

```
        return error_div, empty_fig, html.Div(), html.Div()

@app.callback(
    Output('z-feature-container', 'style'),
    Input('plot-type', 'value')
)
def toggle_z_feature(plot_type):
    if plot_type == '3d-scatter':
        return {'display': 'block'}
    else:
        return {'display': 'none'}

@app.callback(
    Output('interactive-plot', 'figure'),
    [Input('plot-type', 'value'),
     Input('x-feature', 'value'),
     Input('y-feature', 'value'),
     Input('color-feature', 'value'),
     Input('z-feature', 'value'),
     Input('plot-options', 'value')]
)
def update_plot(plot_type, x_feature, y_feature, color_feature, z_feature, options):
    if x_feature is None or (plot_type != 'pie' and y_feature is None):
        fig = go.Figure()
        fig.update_layout(title="Please select features to plot")
        return fig

    try:
        color_var = None if color_feature == 'none' else color_feature

        layout_kwargs = {
            'title': f"{plot_type.capitalize()} Plot of {y_feature if y_feature else ''} vs {x_feature}"
        }

        if 'grid' in options:
            layout_kwargs['xaxis'] = {'showgrid': True, 'gridwidth': 1, 'gridcolor': 'lightgray'}
            layout_kwargs['yaxis'] = {'showgrid': True, 'gridwidth': 1, 'gridcolor': 'lightgray'}

        if 'log' in options and plot_type not in ['pie', 'count', 'box', 'violin']:
            if y_feature and y_feature in numerical_cols:
                layout_kwargs['yaxis_type'] = 'log'

        # Create appropriate plot based on type
        if plot_type == 'bar':
            if x_feature in categorical_cols:
                grouped_data = df.groupby(x_feature)[y_feature].mean().reset_index()
                fig = px.bar(grouped_data, x=x_feature, y=y_feature, color=color_var, barmode='group')
            else:
                fig = px.histogram(df, x=x_feature, y=y_feature, color=color_var)

        elif plot_type == 'line':
            if x_feature in categorical_cols:
                grouped_data = df.groupby(x_feature)[y_feature].mean().reset_index()
                fig = px.line(grouped_data, x=x_feature, y=y_feature, markers=True)
            else:
                sorted_data = df.sort_values(by=x_feature)
                fig = px.line(sorted_data, x=x_feature, y=y_feature, color=color_var)

        elif plot_type == 'scatter':
```

```
        fig = px.scatter(df, x=x_feature, y=y_feature, color=color_var)

        # Add trend line if requested
        if 'trend' in options:
            fig.update_layout(
                shapes=[
                    dict(
                        type='line',
                        xref='x', yref='y',
                        x0=df[x_feature].min(), y0=df[y_feature].min(),
                        x1=df[x_feature].max(), y1=df[y_feature].max(),
                        line=dict(color='red', width=2, dash='dash')
                    )
                ]
            )

    elif plot_type == '3d-scatter':
        fig = px.scatter_3d(df, x=x_feature, y=y_feature, z=z_feature, color=color_var)

    elif plot_type == 'box':
        fig = px.box(df, x=x_feature, y=y_feature, color=color_var)

    elif plot_type == 'violin':
        fig = px.violin(df, x=x_feature, y=y_feature, color=color_var, box=True)

    elif plot_type == 'histogram':
        fig = px.histogram(df, x=x_feature, color=color_var)

    elif plot_type == 'pie':
        if x_feature in categorical_cols:
            value_counts = df[x_feature].value_counts().reset_index()
            value_counts.columns = [x_feature, 'count']
            fig = px.pie(value_counts, names=x_feature, values='count')
        else:
            fig = px.pie(df, names=pd.cut(df[x_feature], bins=10).astype(str))

    elif plot_type == 'heatmap':
        # Create correlation matrix for heatmap
        if x_feature in numerical_cols and y_feature in numerical_cols:
            corr_matrix = df[[x_feature, y_feature]].corr()
            fig = px.imshow(corr_matrix, text_auto=True)
        else:
            cross_tab = pd.crosstab(df[x_feature], df[y_feature])
            fig = px.imshow(cross_tab, text_auto=True)

    elif plot_type == 'density':
        fig = px.density_contour(df, x=x_feature, y=y_feature)

        if 'trend' in options:
            fig.update_traces(contours_coloring="fill", contours_showlabels=True)

    elif plot_type == 'strip':
        fig = px.strip(df, x=x_feature, y=y_feature, color=color_var)

    elif plot_type == 'swarm':
        jitter_data = df.copy()
        jitter_data[x_feature] = jitter_data[x_feature].astype(str)  # Convert to string for jittering
        fig = px.strip(jitter_data, x=x_feature, y=y_feature, color=color_var)
```

```
        elif plot_type == 'count':
            if x_feature in categorical_cols:
                count_data = df[x_feature].value_counts().reset_index()
                count_data.columns = [x_feature, 'count']
                fig = px.bar(count_data, x=x_feature, y='count', color=color_var)
            else:
                fig = px.histogram(df, x=x_feature, color=color_var)
        fig.update_layout(**layout_kwargs)

        return fig

    except Exception as e:
        fig = go.Figure()
        fig.update_layout(title=f"Error creating plot: {str(e)}")
        return fig


# Run the app
if __name__ == "__main__":
    if __name__ == "__main__":
        if os.environ.get("WERKZEUG_RUN_MAIN") != "true":
            print("\n Running Phase 1: Static Data Analysis...\n")
            main()
            print("\n Phase 1 completed. Now launching Phase 2 (Dash app)...\n")

        # Phase 2 — Dash app always runs
        app.run(debug=True, port=8055)
```