Machine Learning - Clustering and Dimensionality Reduction

Assignment 2 - Anomaly Detection using PCA

Rhichard Koh

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Preperation

In [ ]:
```
# loading the data and replacing "?" with np.nans
df = pd.read_csv("data_arrhythmia.csv", sep=";", na_values="?")
df
```

Out[ ]:

	age	sex	height	weight	qrs_duration	p-r_interval	q-t_interval	t_interval	p_interval	qrs	...	KY
0	75	0	190	80	91	193	371	174	121	-16	...	0.0
1	56	1	165	64	81	174	401	149	39	25	...	0.0
2	54	0	172	95	138	163	386	185	102	96	...	0.0
3	55	0	175	94	100	202	380	179	143	28	...	0.0
4	75	0	190	80	88	181	360	177	103	-16	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
447	53	1	160	70	80	199	382	154	117	-37	...	0.0
448	37	0	190	85	100	137	361	201	73	86	...	0.0
449	36	0	166	68	108	176	365	194	116	-85	...	0.0
450	32	1	155	55	93	106	386	218	63	54	...	-0.4
451	78	1	160	70	79	127	364	138	78	28	...	0.0

452 rows × 280 columns

In [ ]:
```
# Checking the number of rows and columns
print(f'There are {df.shape[0]} rows and {df.shape[1]} columns')
```

```
There are 452 rows and 280 columns
```

In [ ]:
```
# Checking for missing values
print(f'There are {df.isna().sum().sum()} missing values')
```

```
There are 408 missing values
```

Dropping NAs

In [ ]:
```
# Dropping the missing values
df = df.dropna()
```

In [ ]:
```
# Checking the missing values again
print(f'There are {df.isna().sum().sum()} missing values')
```

```
There are 0 missing values
```

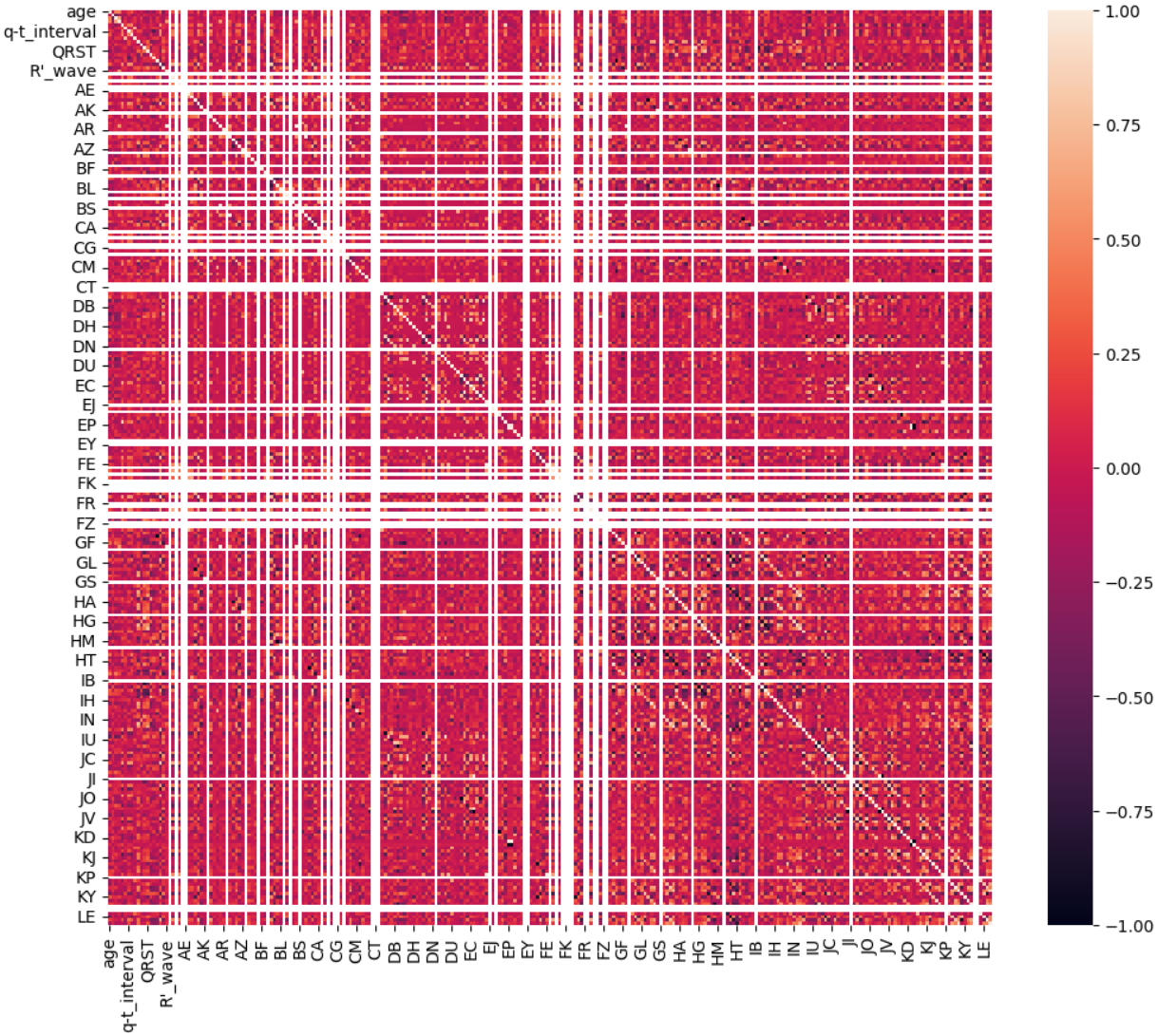
In [ ]:
```
# Checking the number of rows and columns after we dropped the missing values
print(f'There are {df.shape[0]} rows and {df.shape[1]} columns')
```

```
There are 68 rows and 280 columns
```

Check Correlation Matrix

In [ ]:
```
X = df.drop(columns=['diagnosis'])
y = df['diagnosis']
```

In [ ]:
```
# Plotting the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(X.corr())
plt.show()
```



A correlation matrix shows how variables in a dataset relate to each other and is crucial for Principal Component Analysis, a technique that simplifies data while preserving its essence. It helps detect overlapping variables (multicollinearity) and decide which ones to combine, thereby reducing complexity. The matrix also reveals the data's structure, guiding PCA in creating new, unique variables (principal components) that summarize the original data effectively. The success of PCA largely depends on these correlations; strong relationships allow PCA to efficiently reduce dimensions with fewer components, while weaker ones may require more. Essentially, the correlation matrix is key to how PCA simplifies and uncovers patterns in data. In this case it will find variables with high correlation wether positively or negatively correleted and combine them to reduce multicollinearity in our dataset. The very white vairables are correlated with one another positively whereas the purple variables are negatively correlated with one another.

Check Class Distribution

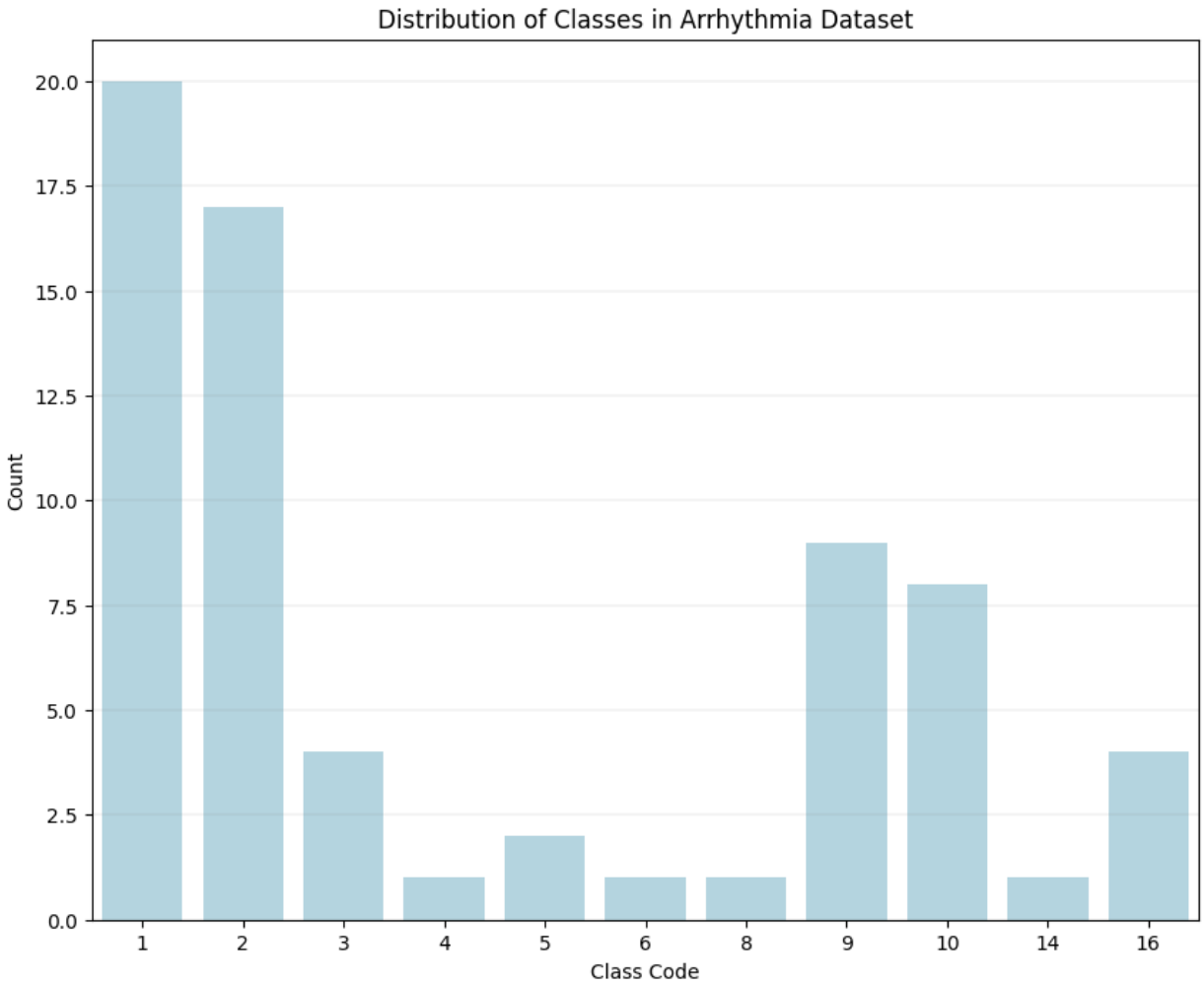
Dropped the NAs

In [ ]:
```
# Counting the classes
class_counts = df['diagnosis'].value_counts().reset_index()
class_counts.columns = ['Class Code', 'Count']
class_counts
```

Out[ ]:

	Class Code	Count
0	1	20
1	2	17
2	9	9
3	10	8
4	16	4
5	3	4
6	5	2
7	6	1
8	14	1
9	8	1
10	4	1

In [ ]:
```
# Plotting the class counts
plt.figure(figsize=(10, 8))
ax = sns.barplot(x='Class Code', y='Count', data=class_counts, color='lightblue')
plt.grid(axis='y', linestyle='-', color='gray', linewidth=0.25, alpha=0.5)
plt.title('Distribution of Classes in Arrhythmia Dataset')
plt.xlabel('Class Code')
plt.ylabel('Count')
plt.show()
```



Raw Data without drop NAs

In [ ]:
```
df_not_dropped = pd.read_csv("data_arrhythmia.csv", sep=";", na_values="?")
```

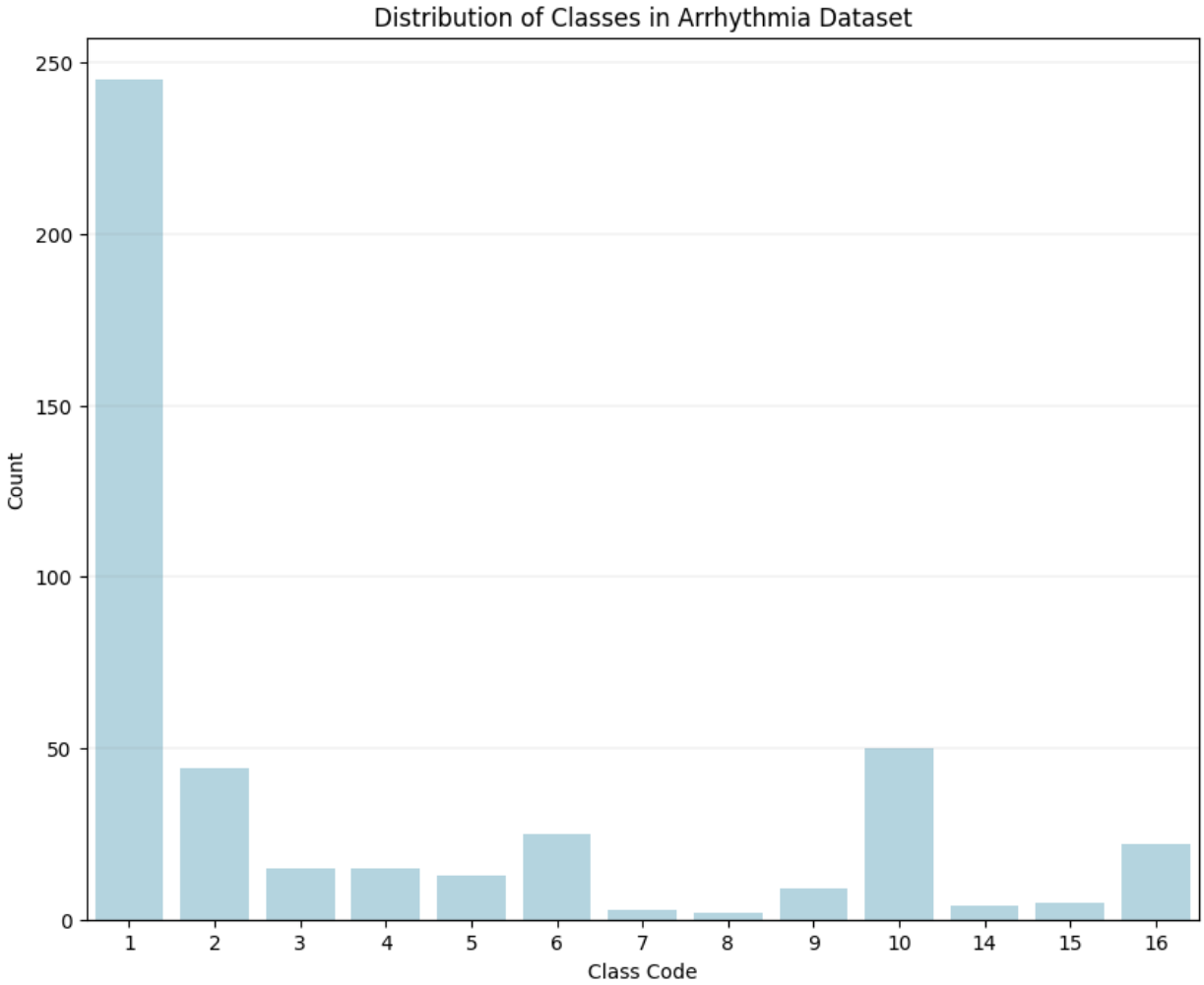
In [ ]:
```
# Counting the classes
class_counts_not_dropped = df_not_dropped['diagnosis'].value_counts().reset_index()
class_counts_not_dropped.columns = ['Class Code', 'Count']
class_counts_not_dropped
```

Out[ ]:

	Class Code	Count
0	1	245
1	10	50
2	2	44
3	6	25
4	16	22
5	3	15
6	4	15
7	5	13
8	9	9
9	15	5
10	14	4
11	7	3
12	8	2

In [ ]:

```
# Plotting the class counts
plt.figure(figsize=(10, 8))
ax = sns.barplot(x='Class Code', y='Count', data=class_counts_not_dropped, color='ligh
plt.grid(axis='y', linestyle='-', color='gray', linewidth=0.25, alpha=0.5)
plt.title('Distribution of Classes in Arrhythmia Dataset')
plt.xlabel('Class Code')
plt.ylabel('Count')
plt.show()
```



Anomaly Detection Task

Data Preprocessing: handle missing values and standardize the features

- handle missing values by imputing the mean value for each feature.
- standardize the features to have a mean of 0 and a standard deviation of 1.

In [ ]:
```
# Checking missing values
df_not_dropped.isna().sum().sum()
```

Out[ ]:
```
408
```

In [ ]:
```
# Getting the mean for each feature
feature_means = df_not_dropped.mean()
```

In [ ]:
```
# Replacing the missing values with the mean for each feature
df_imputed = df_not_dropped.fillna(feature_means)
```

In [ ]:
```
# Checking for missing values again
df_imputed.isna().sum().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
X_imputed = df_imputed.drop(columns=['diagnosis'])
y_imputed = df_imputed['diagnosis']
```

In [ ]:
```
scaler = StandardScaler()
```

In [ ]:
```
# Standardizing the features
X_imputed_scaled = scaler.fit_transform(X_imputed)
```

PCA: Apply PCA to reduce the dimensionality of the dataset.

- Apply PCA to the standardized data, retaining enough components to explain 95% of the variance.

In [ ]:
```
# Retaining 95% of the variance for the PCA class
pca = PCA(n_components=0.95)
```

In [ ]:
```
# Applying PCA to the X_imputed_scaled_PCA
X_imputed_scaled_PCA = pca.fit_transform(X_imputed_scaled)
```

In [ ]:
```
# Checking the shape after PCA
X_imputed_scaled_PCA.shape
```

Out[ ]:
```
(452, 103)
```

Anomaly Detection: Use the PCA-transformed data to detect anomalies.

- calculate the reconstruction error, which represents the difference between the original data and the reconstructed data using the reduced dimensionality.
- Identify Anomalies based on the reconstruction error exceeding a certain threshold (e.g. the 95th percentile of the reconstruction errors).
- Boxplot the anomalies (outliers). Reproduce following plot. Discuss your choice of threshold from this plot.

In [ ]:
```
# calculate the reconstruction error, which represents the difference between the orig
X_reconstructed = pca.inverse_transform(X_imputed_scaled_PCA)
print("Reconstruction Error (MSE):", mean_squared_error(X_imputed_scaled, X_reconstruc
```

```
Reconstruction Error (MSE): 0.0461241780979491
```

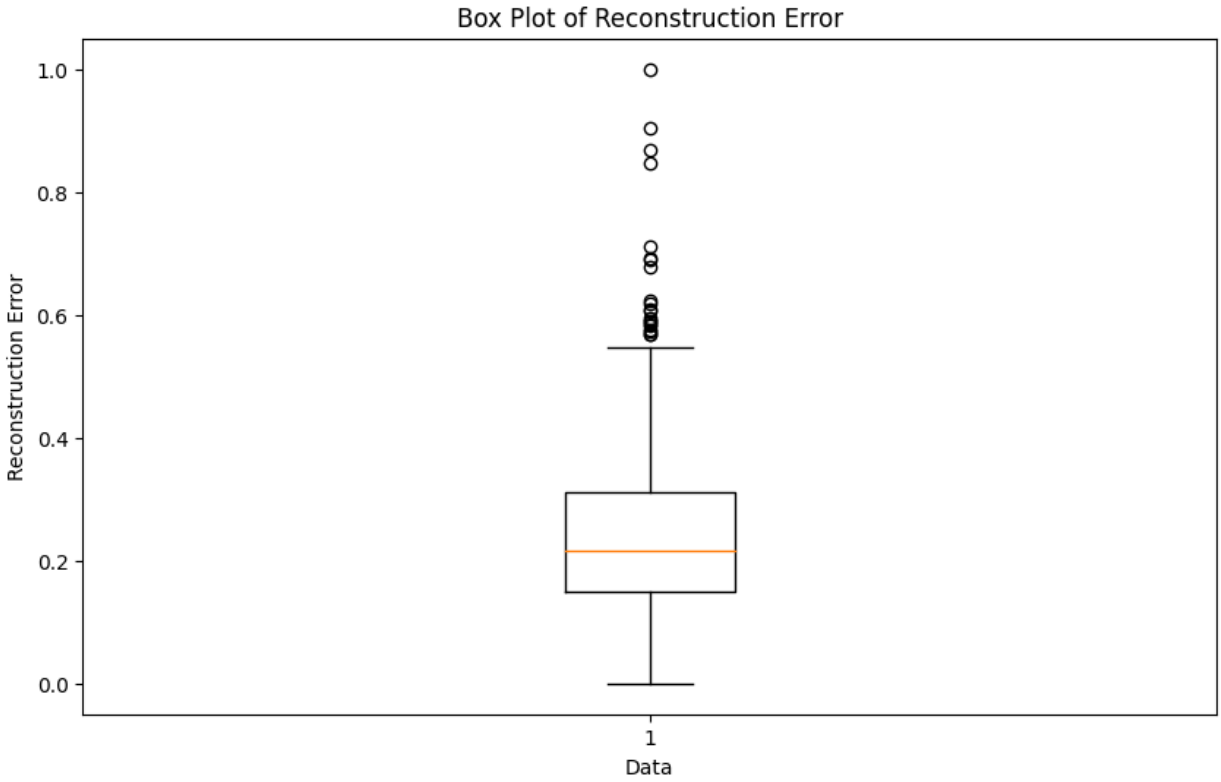
In [ ]:
```
# Identify Anomalies based on the reconstruction error exceeding a certain threshold (
loss = np.sum((X_imputed_scaled - X_reconstructed) ** 2, axis=1)
normalized_loss = (loss - np.min(loss)) / (np.max(loss) - np.min(loss))
errors = pd.Series(data=normalized_loss)
```

```
threshold = np.percentile(errors, 95)
print(f"threshold: {threshold}")

anomalies = np.where(errors > threshold)

print(f"Anomaly Indices: {anomalies[0]}")
for i in anomalies[0]:
  print(f'Index: {i} | Anomaly: {errors[i]}')
```

```
threshold: 0.5432183867485985
Anomaly Indices: [ 36  86 106 139 150 153 189 190 234 241 257 271 299 300 303 306 321
322
 336 361 381 397 406]
Index: 36 | Anomaly: 0.5470067066101647
Index: 86 | Anomaly: 0.6785205751370855
Index: 106 | Anomaly: 0.5686508524699695
Index: 139 | Anomaly: 0.5693133912357772
Index: 150 | Anomaly: 0.7108548744242277
Index: 153 | Anomaly: 0.5900547182662634
Index: 189 | Anomaly: 0.6192124534066579
Index: 190 | Anomaly: 0.5755514755531949
Index: 234 | Anomaly: 0.8473663109329854
Index: 241 | Anomaly: 0.6890604647031408
Index: 257 | Anomaly: 0.5950839581797683
Index: 271 | Anomaly: 0.6053680529308366
Index: 299 | Anomaly: 0.6929916088054251
Index: 300 | Anomaly: 0.5720159709916024
Index: 303 | Anomaly: 0.5844793614406575
Index: 306 | Anomaly: 0.5891955422711942
Index: 321 | Anomaly: 0.6089256702530722
Index: 322 | Anomaly: 0.58379860103729
Index: 336 | Anomaly: 0.6238009074720482
Index: 361 | Anomaly: 0.5437571579502922
Index: 381 | Anomaly: 0.9030289378765842
Index: 397 | Anomaly: 0.8680341361238549
Index: 406 | Anomaly: 1.0
```

In [ ]:

```
# Plotting the Box Plot
plt.figure(figsize=(10, 6))
plt.boxplot(errors)
plt.title("Box Plot of Reconstruction Error")
plt.xlabel("Data")
plt.ylabel("Reconstruction Error")
plt.show()

Q1 = np.percentile(errors, 25)
Q3 = np.percentile(errors, 75)
IQR = Q3 - Q1

threshold = Q3 + 1.5 * IQR
print(f"Threshold for anomalies: {threshold}")
```




```
Threshold for anomalies: 0.550832070025888
```

Reproduce the confusion matrix comparing the detected outliers and actual outliers in the dataset.

In [ ]:
```
# Checking the value counts for the True y
y_imputed.value_counts()
```

Out[ ]:
```
1     245
10     50
2      44
6      25
16     22
3      15
4      15
5      13
9       9
15      5
14      4
7       3
8       2
Name: diagnosis, dtype: int64
```

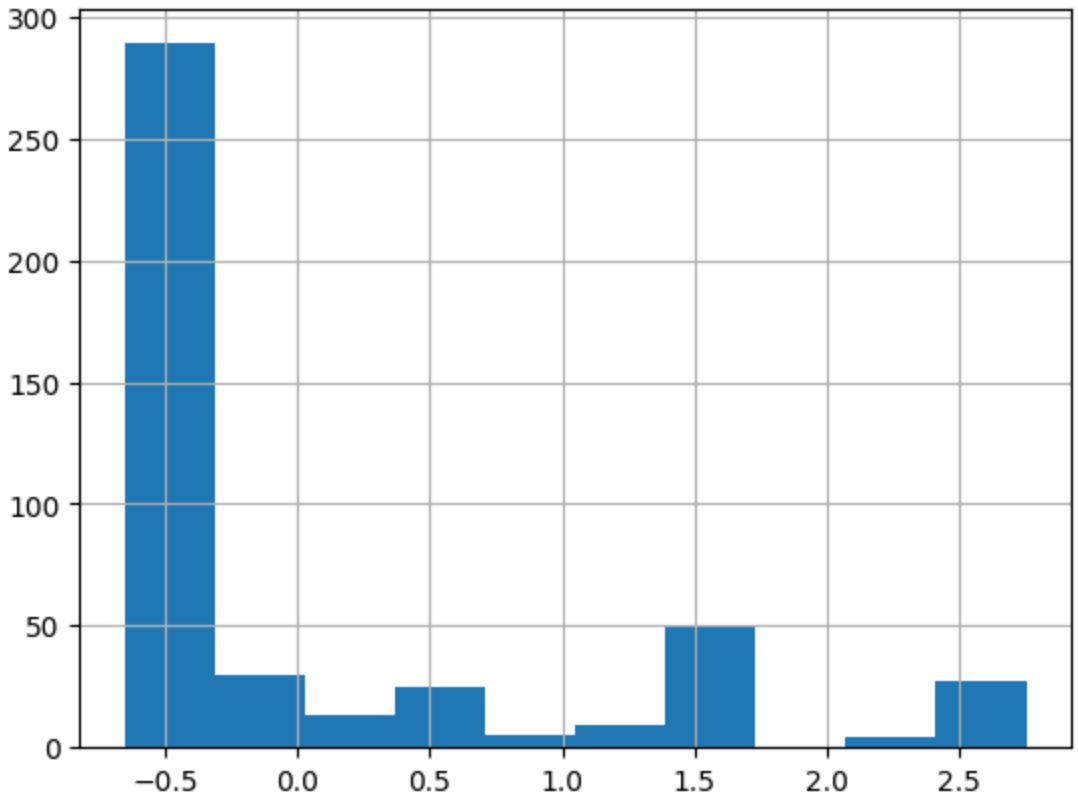
In [ ]:
```
# Standardizing the y_imputed
y_imputed_scaled = scaler.fit_transform(y_imputed.to_numpy().reshape(-1,1))
```

In [ ]:
```
# Unravelling the numpy array
y_imputed_scaled = y_imputed_scaled.ravel()
```

In [ ]:
```
# Converting the numpy array to a pd.Series to plot
y_imputed_scaled = pd.Series(y_imputed_scaled)
```

In [ ]:
```
# Plotting the distribution of y_imputed_scaled
y_imputed_scaled.hist()
```

Out[ ]:
```
<Axes: >
```



In [ ]:
```
# Converting the target to useful labels, assuming anything above class 1 is an anomal
class_label = y_imputed.apply(lambda x: 0 if x == 1 else 1)
```

In [ ]:
```
# Checking the binary classification
class_label
```

Out[ ]:
```
0      1
1      1
2      1
3      0
4      1
      ..
447    0
448    1
449    1
450    0
451    0
Name: diagnosis, Length: 452, dtype: int64
```

In [ ]:
```
# Checking the values counts
class_label.value_counts()
```

Out[ ]:
```
0    245
1    207
Name: diagnosis, dtype: int64
```

In [ ]:
```
# Setting a threshold for predicted anomalies
threshold = np.percentile(errors, 95)
```

```
threshold
```

Out[ ]:
```
0.5432183867485985
```

In [ ]:
```
# Connverting the predicted class labels into 0 (not anomaly) if its less than our thr
predicted_class_label = errors.apply(lambda x: 0 if x < threshold else 1)
```

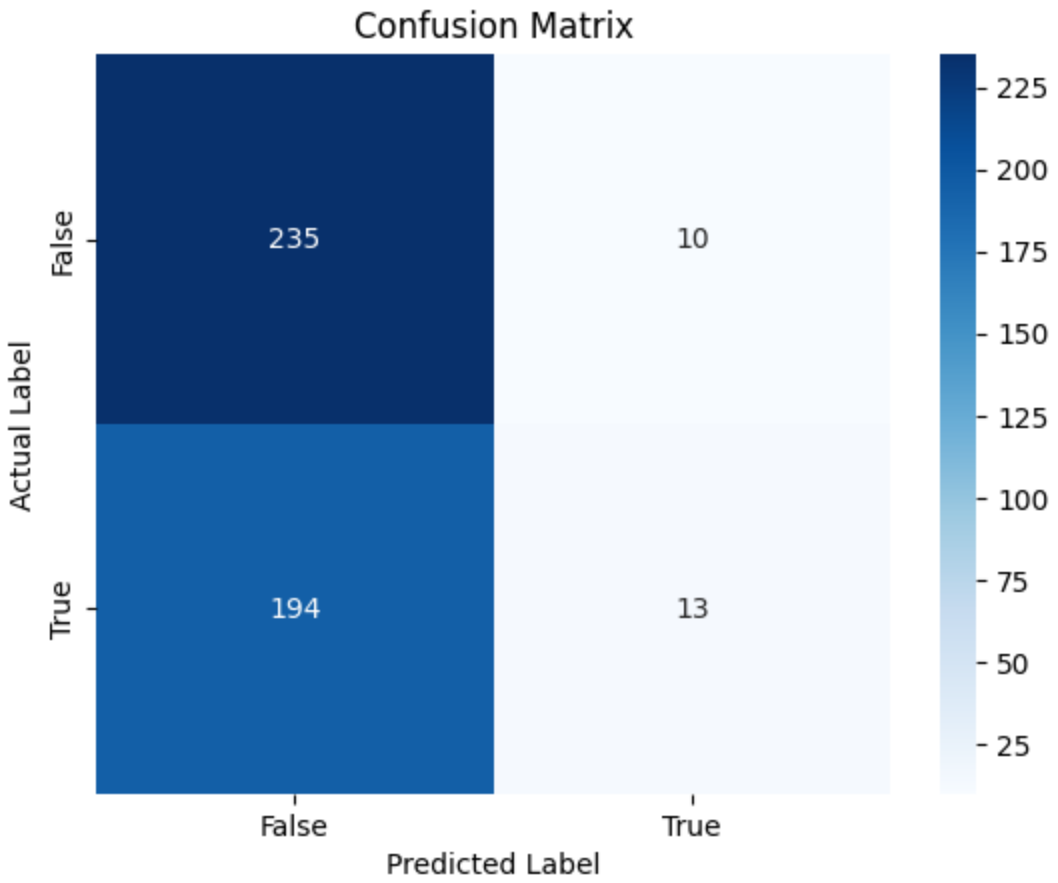
In [ ]:
```
# Checking the value counts for our predicted class
predicted_class_label.value_counts()
```

Out[ ]:
```
0    429
1     23
dtype: int64
```

In [ ]:
```
# Creating a confusion matrix
cm = confusion_matrix(class_label, predicted_class_label)
cm
```

Out[ ]:
```
array([[235,  10],
       [194,  13]])
```

In [ ]:
```
# Making the confusion matrix look nicer
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['False', 'True'], ytic
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
plt.show()
```



Reproduce the following plot showing detected outliers and actual outliers using first two PCs.

In [105…
```
# Converting so we only use the first two PCs
first_two = X_imputed_scaled_PCA[:, :2]
n_components = pca.n_components_
data_reduced = np.zeros(shape=(X_imputed_scaled_PCA.shape[0], n_components))
data_reduced[:, :2] = first_two
data_inverse_two_pca = pca.inverse_transform(data_reduced)
```

In [106…
```
# Converting into a dataframe
df_first_two = pd.DataFrame(first_two, columns=['PC1', 'PC2'])

# Finding the IQR
Q1 = df_first_two.quantile(0.25)
Q3 = df_first_two.quantile(0.75)
IQR = Q3 - Q1

# Defining our upper and lower bound
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Finding the outliers
outlier_data = df_first_two[((df_first_two < lower_bound) | (df_first_two > upper_bour

print(f"Number of outliers: {outlier_data.shape[0]}")
```

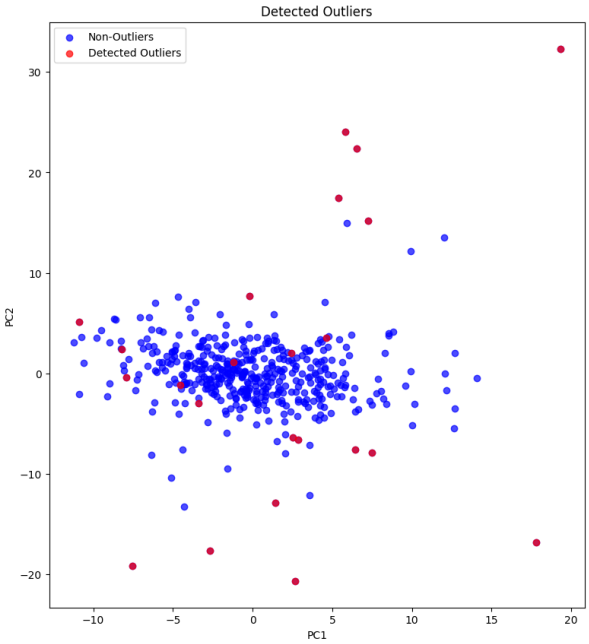
```
Number of outliers: 30
```

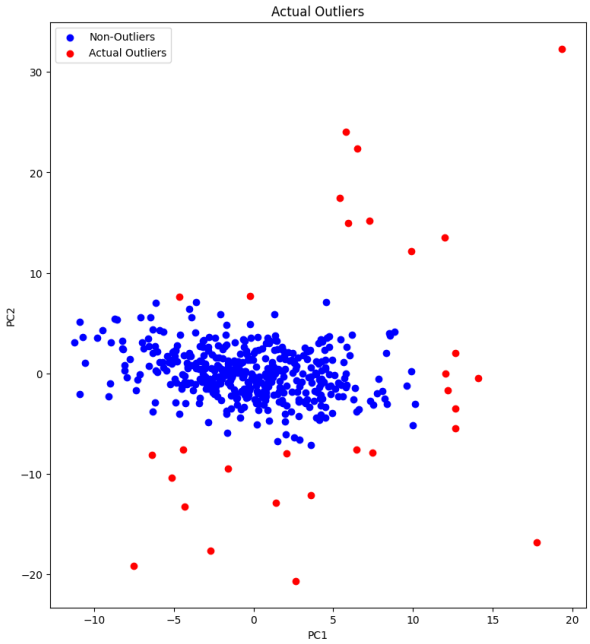
In [115…
```
# Reconstruction error
loss_2 = np.sum((X_imputed_scaled - data_inverse_two_pca) ** 2, axis=1)
normalized_loss_2 = (loss_2 - np.min(loss_2)) / (np.max(loss_2) - np.min(loss_2))
anomaly_scores_2 = pd.Series(data=normalized_loss_2)

threshold_2 = np.percentile(anomaly_scores_2, 95)

print(f"The threshold is {threshold_2}")

anomalies_2 = np.where(anomaly_scores_2 > threshold_2)[0]

anomalies_series_2 = pd.Series(anomaly_scores_2[anomalies_2], index=anomalies_2)

print(anomalies_series_2)
```

```
The threshold is 0.31443420806623384
76     0.507962
85     0.724400
88     0.326256
108    0.498179
116    0.315177
133    0.366059
141    1.000000
204    0.497862
207    0.405084
213    0.405144
218    0.614961
297    0.565333
308    0.393552
312    0.397815
318    0.398043
354    0.475001
363    0.338066
370    0.330875
379    0.885820
388    0.399453
403    0.422311
424    0.315100
449    0.563177
dtype: float64
```

In [116…

```
# Plotting the Detected and Acual Outliers
plt.figure(figsize=(20, 10))
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot

# Plot not outliers as blue
plt.scatter(first_two[:, 0], first_two[:, 1], color='blue', label='Non-Outliers', alph

# Plot outliers as red
plt.scatter(first_two[anomalies_2, 0], first_two[anomalies_2, 1], color='red', label='
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Detected Outliers')
plt.legend()

plt.subplot(1, 2, 2)

# Plotting Non-Outliers
plt.scatter(df_first_two.loc[~((df_first_two < lower_bound) | (df_first_two > upper_bo
# Plotting Outliers
plt.scatter(df_first_two.loc[((df_first_two < lower_bound) | (df_first_two > upper_bou

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Actual Outliers')
plt.legend()
plt.show()
```


Detected Outliers
Non-Outliers
Detected Outliers
PC1
PC2


Actual Outliers
Non-Outliers
Actual Outliers
PC1
PC2