Step 1: Data Cleaning

```
    try:
        systolic, diastolic = map(int, bp.split('/'))
        if systolic < 90 or systolic > 200 or diastolic < 60 or diastolic > 120:
            return np.nan
        else:
            return bp
    except:
        return np.nan

df['blood_pressure'] = df['blood_pressure'].apply(clean_bp)

# Handle missing or invalid data (age, weight, and blood pressure)
df.fillna({'age': df['age'].median(), 'weight_kg': df['weight_kg'].median(), 'blood_pressure': '120/80'}, inplace=True)

print(df)
```

```
   patient_id           name   age  weight_kg  height_cm blood_pressure  \
0           1       John Doe  45.0       85.0        175         130/85
1           2     Jane Smith  67.0       70.0        165         140/90
2           3    Bob Johnson  25.0       90.0        180         125/80
3           4  Alice Williams  40.0       68.0        160         120/80
4           5     Mary Brown  40.0       75.0        170         135/85
5           6   Invalid Entry  40.0       75.0        175         120/80
6           7   Michael Davis  34.0       85.0        180         132/86
7           8     David Clark  80.0       65.0        178         130/80
8           9     Susan Moore  28.0       54.0        155         120/75
9          10   Robert Wilson  39.0       80.0        170         135/85

  diagnosis_code  visit_date
0            I10  2024-09-15
1            E11  2024-09-16
2            I10  2024-09-17
3            E11  2024-09-18
4            I10  2024-09-19
5            E11  2024-09-20
6            I10  2024-09-21
7            E11  2024-09-22
```

Step 2: Data Transformation

```
# Calculate BMI (Body Mass Index) and add as a new column
df['bmi'] = df['weight_kg'] / (df['height_cm'] / 100) ** 2

# Categorize age into groups
df['age_group'] = pd.cut(df['age'], bins=[0, 18, 35, 50, 65, 100], labels=['Child', 'Young Adult', 'Adult', 'Senior Adult', 'Elderly'])

# Convert visit_date to datetime format for better analysis
df['visit_date'] = pd.to_datetime(df['visit_date'])

print(df[['patient_id', 'age', 'bmi', 'age_group', 'visit_date']])
```

```
   patient_id   age        bmi    age_group visit_date
0           1  45.0  27.755102        Adult 2024-09-15
1           2  67.0  25.711662      Elderly 2024-09-16
2           3  25.0  27.777778  Young Adult 2024-09-17
3           4  40.0  26.562500        Adult 2024-09-18
4           5  40.0  25.951557        Adult 2024-09-19
5           6  40.0  24.489796        Adult 2024-09-20
6           7  34.0  26.234568  Young Adult 2024-09-21
7           8  80.0  20.515086      Elderly 2024-09-22
8           9  28.0  22.476587  Young Adult 2024-09-23
9          10  39.0  27.681661        Adult 2024-09-24
```

Step 3: Data Integration

```
# Example of a new dataset (with patient_id)
additional_data = {
    'patient_id': [8, 9, 10],
    'allergies': ['Peanuts', 'None', 'Shellfish'],
    'medication': ['Aspirin', 'Metformin', 'Lisinopril']
}

# Convert to DataFrame
df_additional = pd.DataFrame(additional_data)

# Merge based on patient_id (left join to keep all existing records)
df = pd.merge(df, df_additional, on='patient_id', how='left')

# Replace missing allergy and medication values with 'None'
df['allergies'].fillna('None', inplace=True)
df['medication'].fillna('None', inplace=True)

print(df)
```

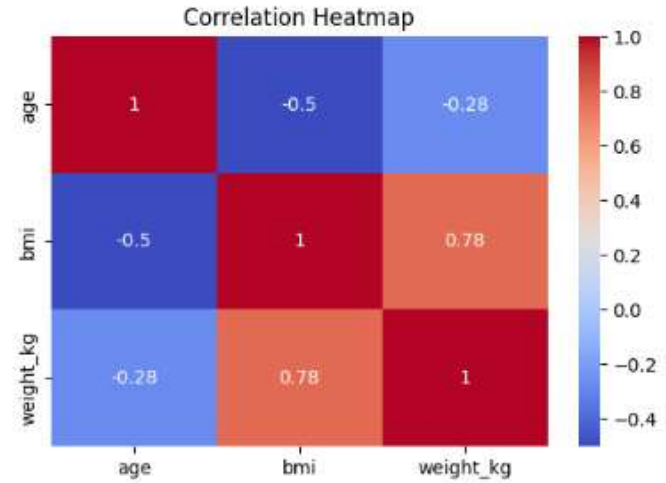
```
   patient_id            name   age  weight_kg  height_cm blood_pressure  \
0           1        John Doe  45.0       85.0        175         130/85
1           2      Jane Smith  67.0       70.0        165         140/90
2           3     Bob Johnson  25.0       90.0        180         125/80
3           4   Alice Williams  40.0       68.0        160         120/80
4           5      Mary Brown  40.0       75.0        170         135/85
5           6   Invalid Entry  40.0       75.0        175         120/80
6           7   Michael Davis  34.0       85.0        180         132/86
7           8     David Clark  80.0       65.0        178         130/80
8           9     Susan Moore  28.0       54.0        155         120/75
9          10   Robert Wilson  39.0       80.0        170         135/85

  diagnosis_code visit_date        bmi    age_group  allergies  medication
0            I10 2024-09-15  27.755102        Adult       None        None
1            E11 2024-09-16  25.711662      Elderly       None        None
2            I10 2024-09-17  27.777778  Young Adult       None        None
3            E11 2024-09-18  26.562500        Adult       None        None
4            I10 2024-09-19  25.951557        Adult       None        None
5            E11 2024-09-20  24.489796        Adult       None        None
6            I10 2024-09-21  26.234568  Young Adult       None        None
7            E11 2024-09-22  20.515086      Elderly    Peanuts     Aspirin
8            I10 2024-09-23  22.476587  Young Adult       None   Metformin
9            E11 2024-09-24  27.681661        Adult  Shellfish  Lisinopril
```

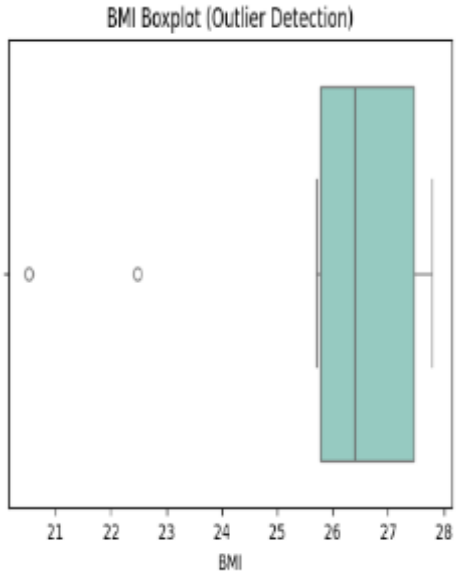
Step 4: Save to CSV

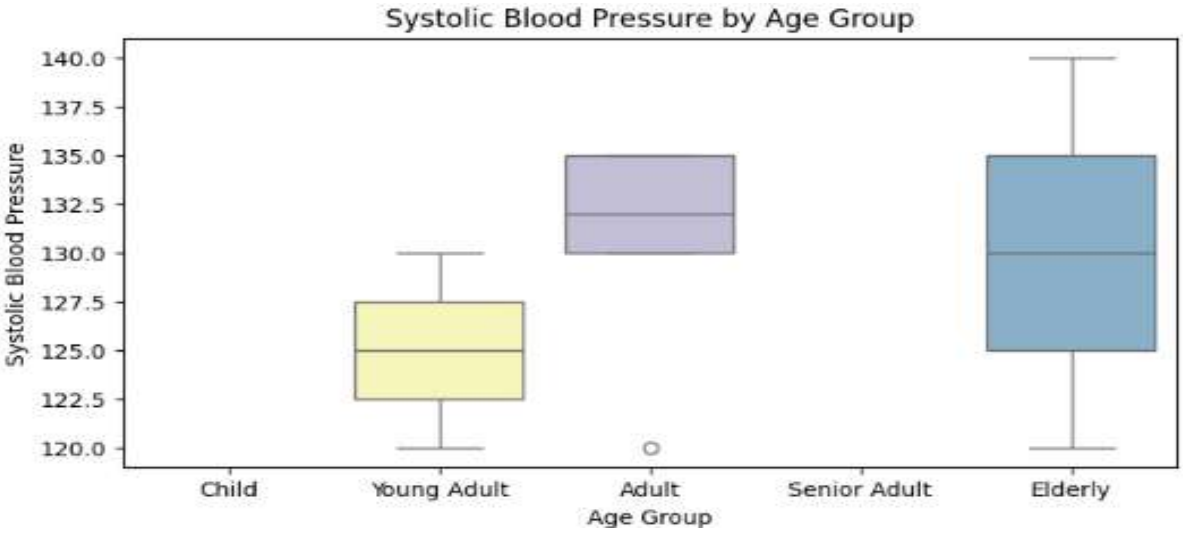
```
# Optionally, write the final DataFrame to a new CSV file
df.to_csv('cleaned_transformed_ehr.csv', index=False)

print("Data cleaning, integration, and transformation complete. CSV file generated.")
```

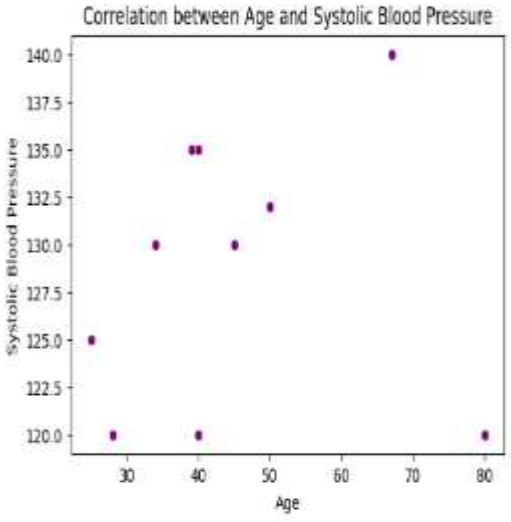
```
Data cleaning, integration, and transformation complete. CSV file generated.
```

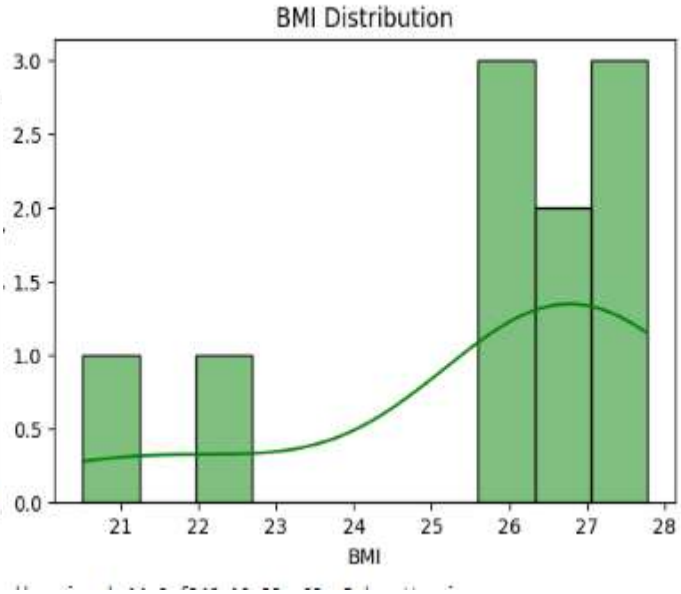
```
--- Correlation Matrix ---
                age       bmi  weight_kg
age        1.000000 -0.502723  -0.276090
bmi       -0.502723  1.000000   0.780925
weight_kg -0.276090  0.780925   1.000000
```



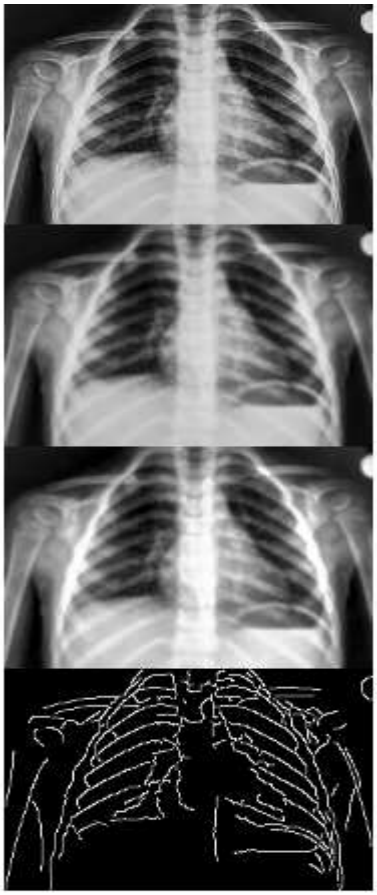




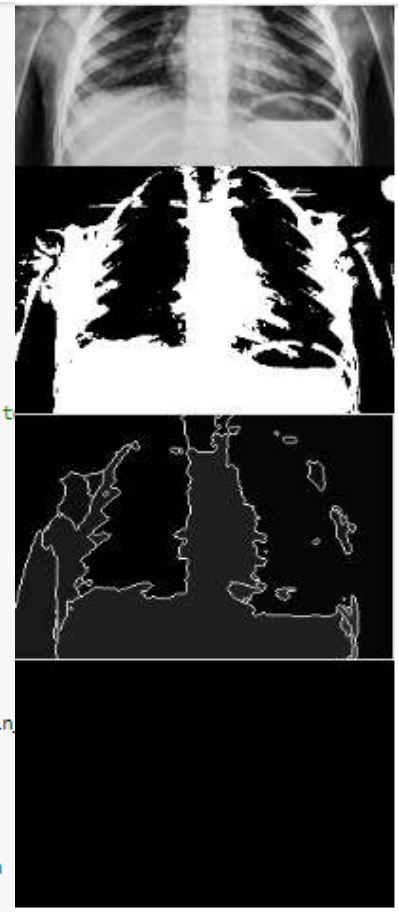




```python
import cv2
import numpy as np
from skimage import exposure
from google.colab.patches import cv2_imshow  # Use cv2_imshow for Google Colab

# Load the medical image (e.g., MRI or X-ray)
image = cv2.imread('medical_image.jfif', cv2.IMREAD_GRAYSCALE)

# 1. Noise Reduction using Gaussian Filter
smoothed_image = cv2.GaussianBlur(image, (5, 5), 0)

# 2. Histogram Equalization for Contrast Enhancement
equalized_image = cv2.equalizeHist(smoothed_image)

# 3. Edge Detection using Canny Edge Detector
edges = cv2.Canny(equalized_image, 100, 200)

# 4. Normalization (Optional, to scale pixel values between 0 and 1)
normalized_image = cv2.normalize(equalized_image, None, 0, 255, cv2.NORM_MINMAX)

# Display the results using cv2_imshow for Colab
cv2_imshow(image)            # Original Image
cv2_imshow(smoothed_image)   # Smoothed Image
cv2_imshow(equalized_image)  # Equalized Image
cv2_imshow(edges)            # Edge Detected Image
```

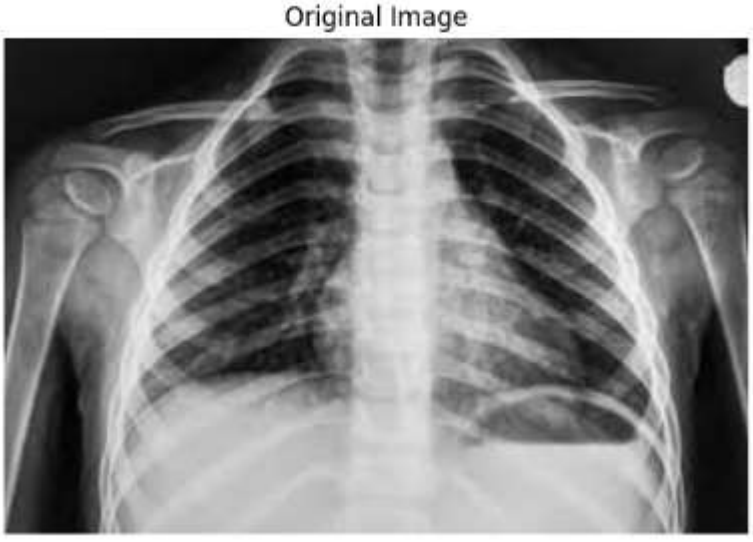


```python
image = cv2.imread('medical_image.jfif', cv2.IMREAD_GRAYSCALE)

# Convert the grayscale image to a 3-channel image
image_color = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)

# 1. Thresholding Segmentation
_, thresholded_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# 2. Watershed Segmentation
# Apply Sobel filter for edge detection
gradient = sobel(image)

# Perform binary thresholding and distance transform
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
dist_transform = cv2.distanceTransform(binary_image, cv2.DIST_L2, 5)

# Define markers for the watershed algorithm
# Mark sure foreground is set to a different value (positive), background t
ret, markers = cv2.connectedComponents(np.uint8(binary_image))

# Add 1 to all markers so background is 1 instead of 0
markers = markers + 1

# Mark the unknown region (edges) with 0
markers[binary_image == 0] = 0

# Watershed segmentation using the 3-channel image
cv2.watershed(image_color, markers)

# 3. Morphological Segmentation (Remove small objects)
segmented_image = morphology.remove_small_objects(markers.astype(bool), min

# Display the results using cv2_imshow for Colab
cv2_imshow(image)              # Original Image
cv2_imshow(thresholded_image) # Thresholded Image
cv2_imshow(markers.astype(np.uint8))  # Watershed Segmentation
cv2_imshow(segmented_image.astype(np.uint8))  # Morphological Segmentation
```



```python
# Normalize image
image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)

# Function to compute GLCM
def compute_glcm(image, distances, angles):
    max_val = int(np.max(image))
    glcm = np.zeros((max_val + 1, max_val + 1), dtype=np.float64)

    for d in distances:
        for a in angles:
            for i in range(image.shape[0]):
                for j in range(image.shape[1]):
                    if a == 0:  # horizontal
                        if j + d < image.shape[1]:
                            glcm[image[i, j], image[i, j + d]] += 1
                    elif a == 90:  # vertical
                        if i + d < image.shape[0]:
                            glcm[image[i, j], image[i + d, j]] += 1
    return glcm

# Calculate GLCM
distances = [1]
angles = [0]  # 0 degrees
glcm = compute_glcm(image, distances, angles)

# Normalize GLCM
glcm_normalized = glcm / np.sum(glcm)

# Calculate texture features from the GLCM (example)
contrast = np.sum(glcm_normalized * (np.arange(glcm.shape[0])[:, None] - np.arange(glcm.shape[1])) *

print("Contrast:", contrast)

# Display original image
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

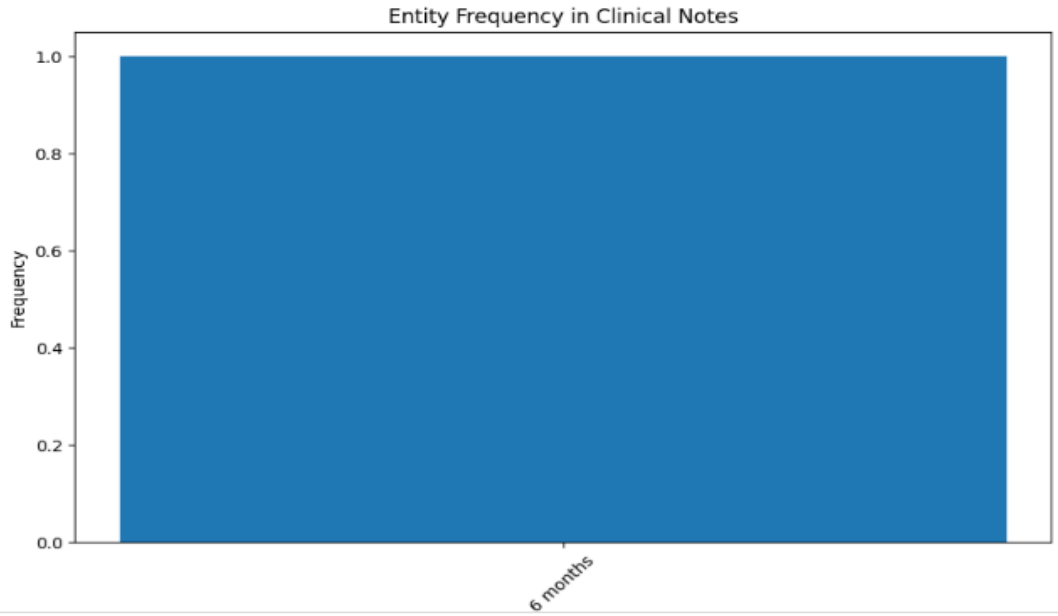
```
Contrast: 206.0527103027402
```



```
stop_words = set(stopwords.words('english'))

df['cleaned_notes'] = df['cleaned_notes'].apply(
    lambda x: ' '.join([word for word in x.split() if word not in stop_words])
)

# 2. Named Entity Recognition (NER)
def extract_entities(text):
    doc = nlp(text)
    return [(ent.text, ent.label_) for ent in doc.ents]

df['entities'] = df['cleaned_notes'].apply(extract_entities)

# 3. Visualization of Entity Frequency
entity_freq = {}
for entities in df['entities']:
    for entity, label in entities:
        if entity in entity_freq:
            entity_freq[entity] += 1
        else:
            entity_freq[entity] = 1

# Plotting entity frequency
plt.figure(figsize=(10, 6))
plt.bar(entity_freq.keys(), entity_freq.values())
plt.title('Entity Frequency in Clinical Notes')
plt.xlabel('Entities')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()

# Display the DataFrame with cleaned notes and extracted entities
print(df[['clinical_notes', 'cleaned_notes', 'entities']])
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

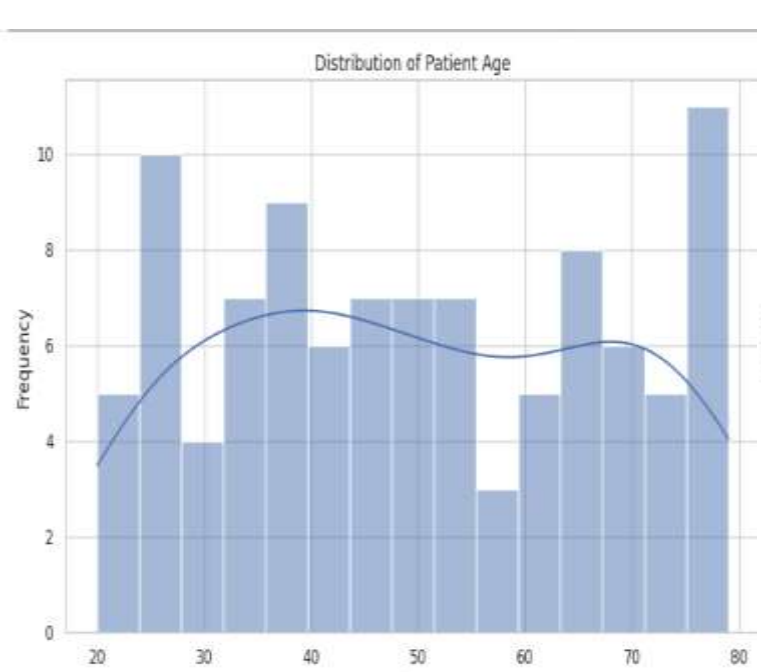
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

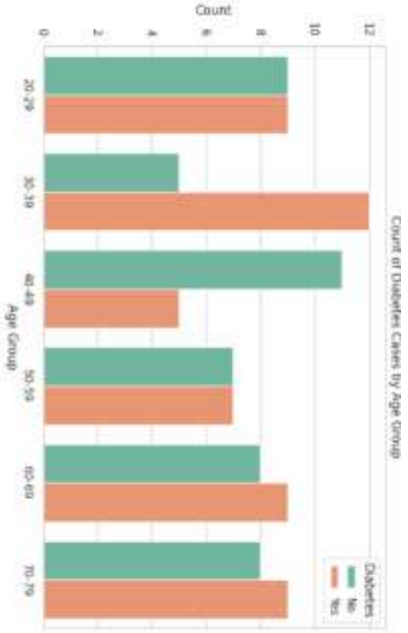


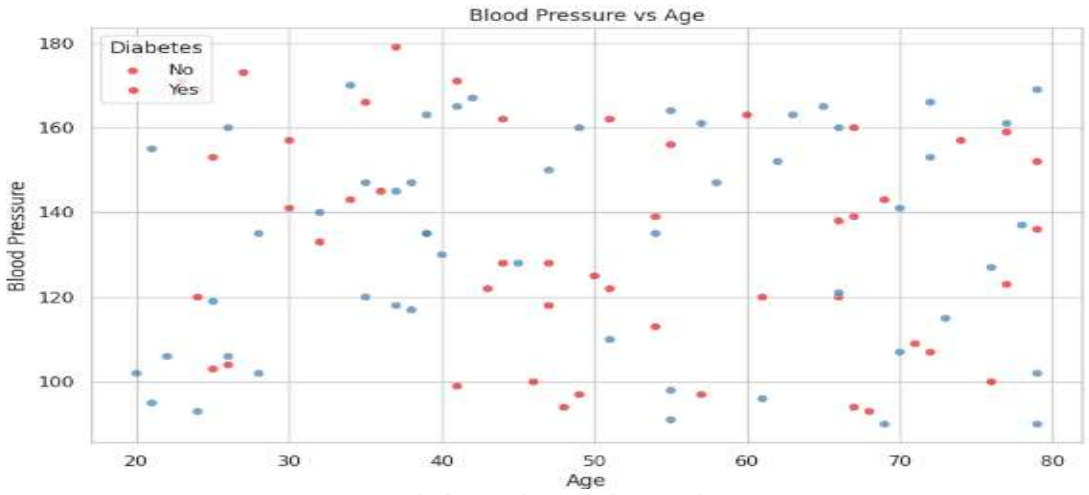

```
                                      clinical_notes  \
0  Patient diagnosed with diabetes. Prescribed me...
1  Patient shows symptoms of hypertension. Recomm...
2  The MRI shows no significant abnormalities. Fo...
3  Patient complains of headaches and nausea. Con...

                                      cleaned_notes            entities
0  patient diagnosed diabetes. prescribed metformin.                  []
1  patient shows symptoms hypertension. recommend...                  []
2  mri shows significant abnormalities. follow 6 ...  [(6 months, DATE)]
3  patient complains headaches nausea. consider t...                  []
```

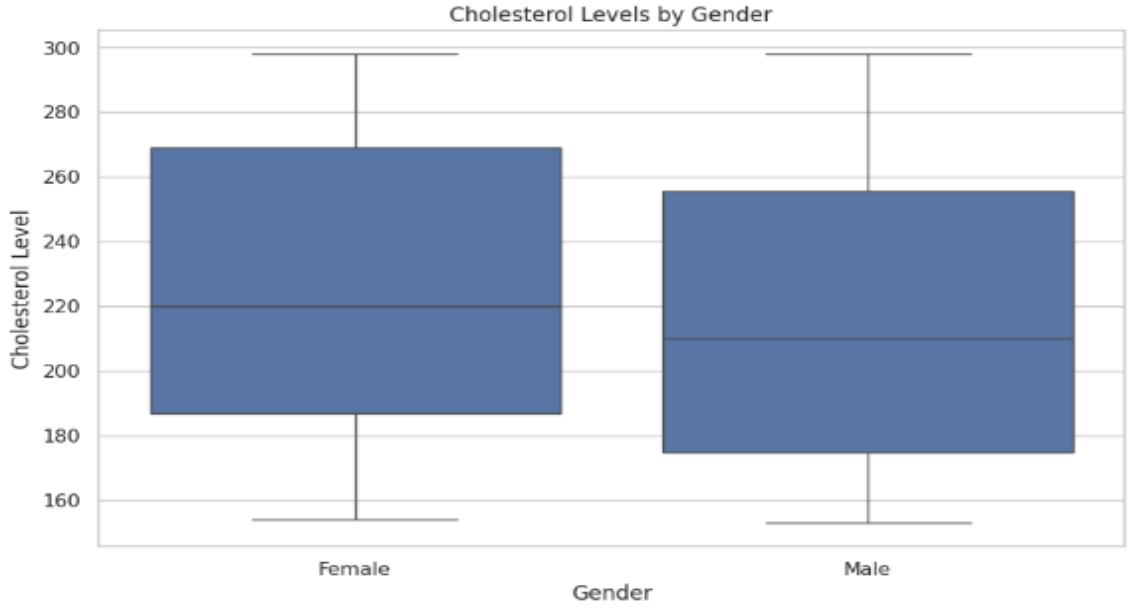
```python
df = pd.DataFrame(data)

# Features and target variable
X = df.drop('Diabetes_Risk', axis=1)
y = df['Diabetes_Risk']

# 1. Data Preprocessing
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 2. Model Selection
# Using Random Forest Classifier
model = RandomForestClassifier(random_state=42)

# 3. Model Training
model.fit(X_train, y_train)

# 4. Model Prediction
y_pred = model.predict(X_test)

# 5. Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]))

# Example of predicting risk for a new patient
new_patient = [[45, 28, 130, 220, 1]]  # Age, BMI, Blood Pressure, Cholesterol, Family History
new_patient_scaled = scaler.transform(new_patient)
risk_prediction = model.predict(new_patient_scaled)

print(f"Predicted Diabetes Risk for the new patient: {'High' if risk_prediction[0] == 1 else 'Low'}")
```

```
Confusion Matrix:
 [[1 0]
 [1 0]]

Classification Report:
               precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

ROC AUC Score: 1.0
Predicted Diabetes Risk for the new patient: High
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```


Distribution of Patient Age
Frequency
0
2
4
6
8
10
20
30
40
50
60
70
80


Count
0
2
4
6
8
10
12
20-29
30-39
40-49
50-59
60-69
70-79
Age Group
Diabetes
No
Yes
Count of Diabetes Cases by Age Group


Blood Pressure vs Age
Diabetes
No
Yes
Blood Pressure
100
120
140
160
180
20
30
40
50
60
70
80
Age


Cholesterol Levels by Gender
Cholesterol Level
160
180
200
220
240
260
280
300
Female
Male
Gender