[1] Import Libary

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import tensorflow as tf
```

In [2]:
```
dataset = pd.read_csv('biased_leukemia_dataset.csv')
```

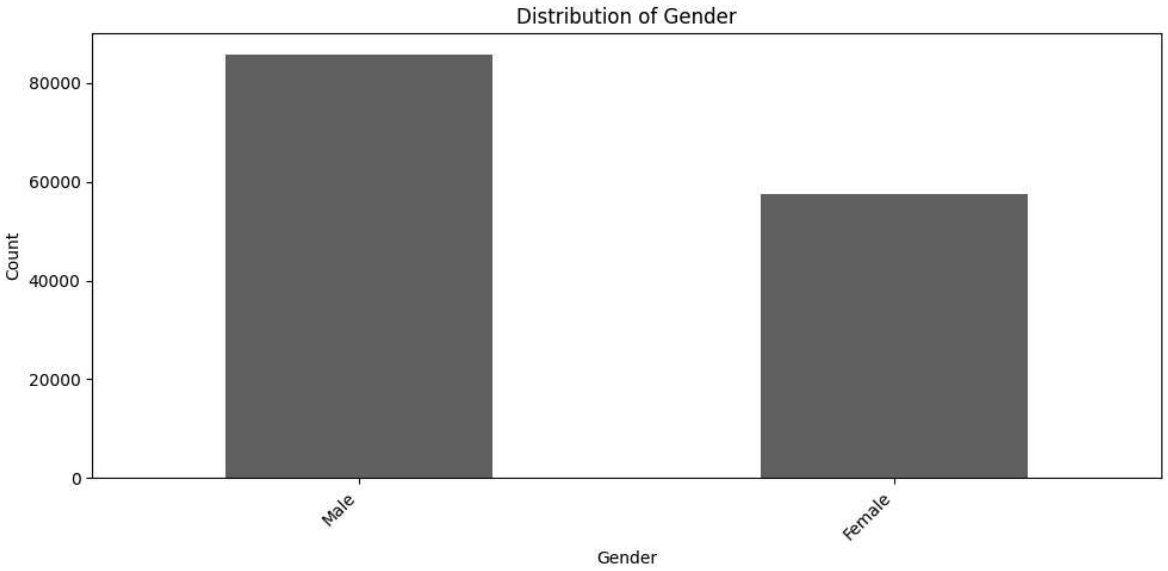
[2] EDA

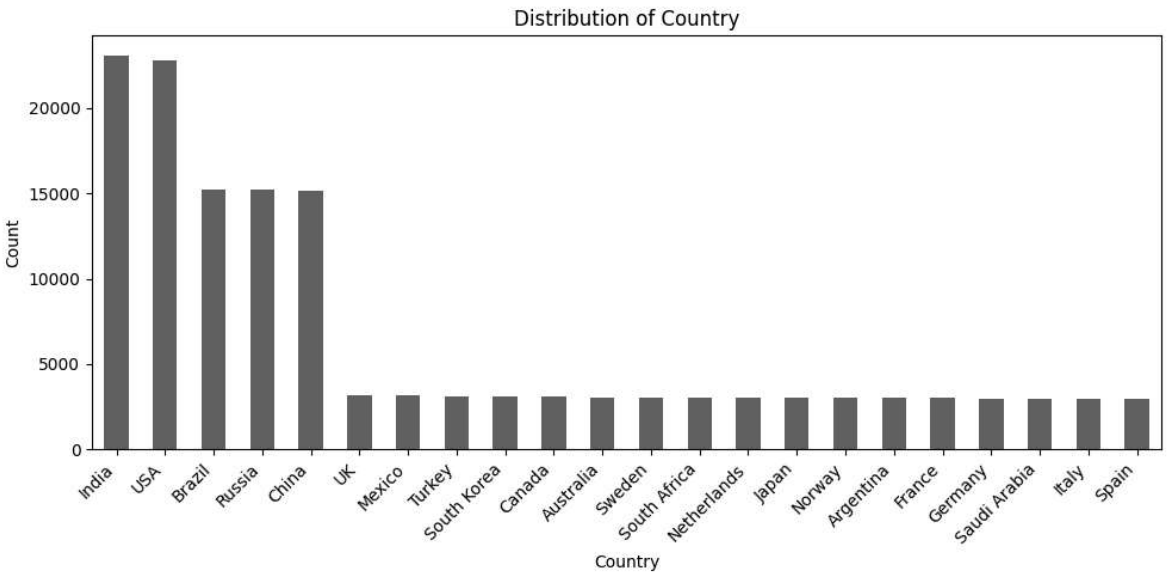
In [ ]:
```
#การวิเคราะห์ข้อมูลเบื้องต้น
dataset.info()
```

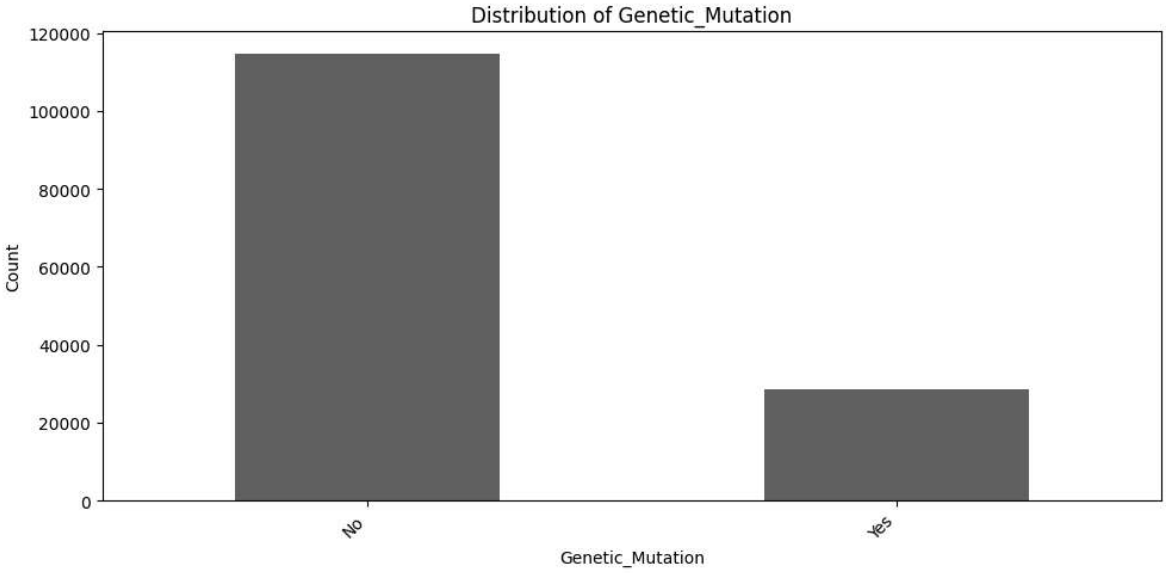
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 143194 entries, 0 to 143193
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Patient_ID            143194 non-null  int64  
 1   Age                   143194 non-null  int64  
 2   Gender                143194 non-null  object 
 3   Country               143194 non-null  object 
 4   WBC_Count             143194 non-null  int64  
 5   RBC_Count             143194 non-null  float64
 6   Platelet_Count        143194 non-null  int64  
 7   Hemoglobin_Level      143194 non-null  float64
 8   Bone_Marrow_Blasts    143194 non-null  int64  
 9   Genetic_Mutation      143194 non-null  object 
 10  Family_History        143194 non-null  object 
 11  Smoking_Status        143194 non-null  object 
 12  Alcohol_Consumption   143194 non-null  object 
 13  Radiation_Exposure    143194 non-null  object 
 14  Infection_History     143194 non-null  object 
 15  BMI                   143194 non-null  float64
 16  Chronic_Illness       143194 non-null  object 
 17  Immune_Disorders      143194 non-null  object 
 18  Ethnicity             143194 non-null  object 
 19  Socioeconomic_Status  143194 non-null  object 
 20  Urban_Rural           143194 non-null  object 
 21  Leukemia_Status       143194 non-null  object 
dtypes: float64(3), int64(5), object(14)
memory usage: 24.0+ MB
```

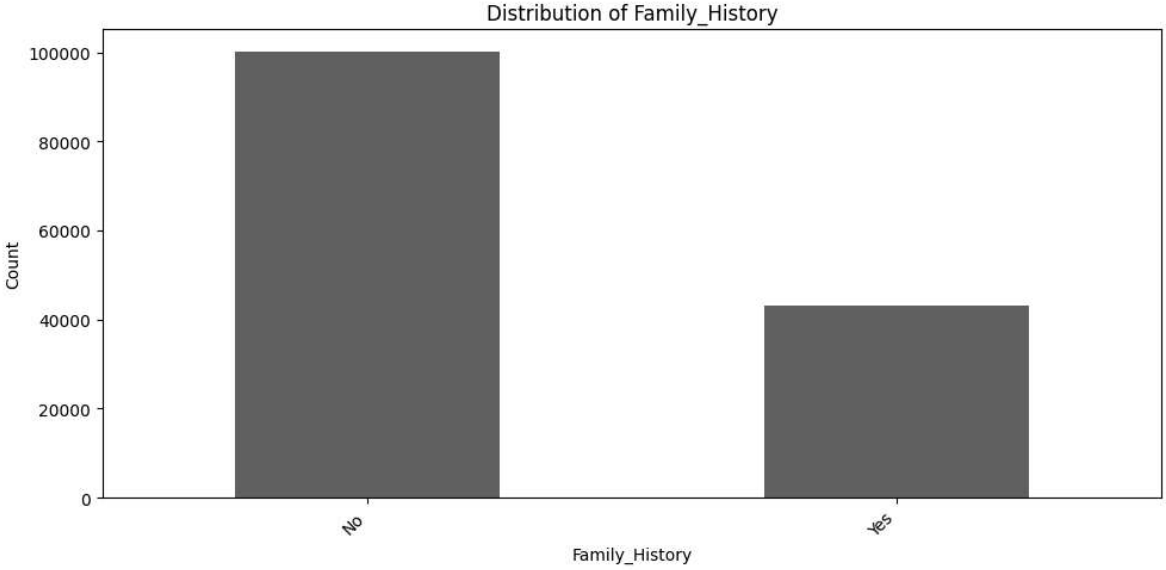
In [ ]:
```
#แยกชนิดของข้อมูลแล้ว plot ออกมาดูว่า ข้อมูลแต่ละชนิดมีปริมาณแตกต่างกันอย่างไร
categorical_cols = [
    'Gender', 'Country', 'Genetic_Mutation', 'Family_History',
    'Smoking_Status', 'Alcohol_Consumption', 'Radiation_Exposure',
    'Infection_History', 'Chronic_Illness', 'Immune_Disorders',
```

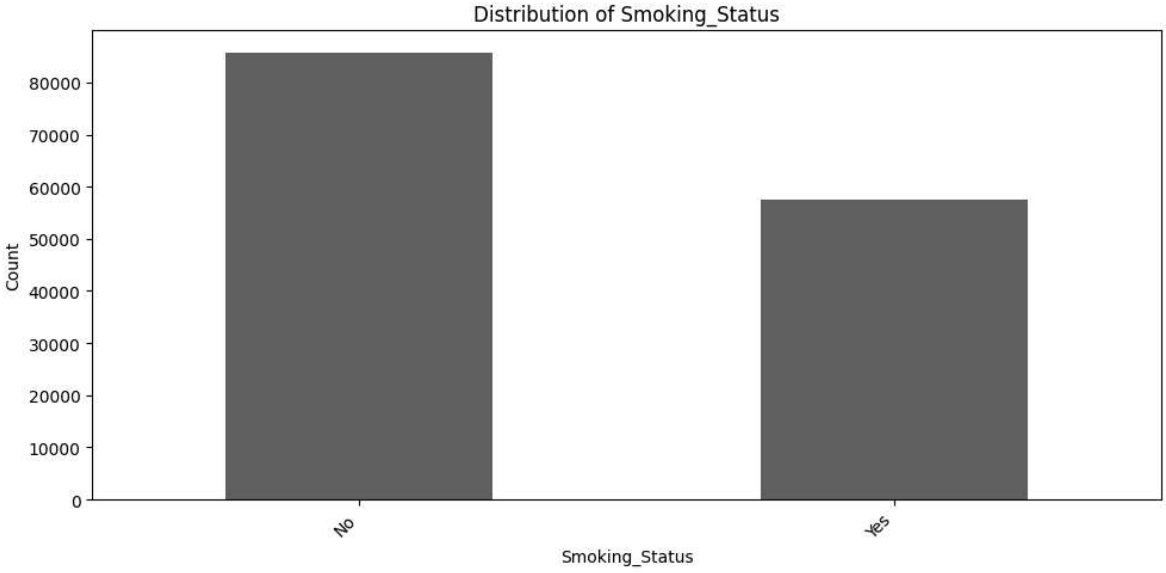
```
    'Ethnicity', 'Socioeconomic_Status', 'Urban_Rural', 'Leukemia_Status'
]

# Plot
for col in categorical_cols:
    plt.figure(figsize=(10, 5))
    dataset[col].value_counts().plot(kind='bar')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

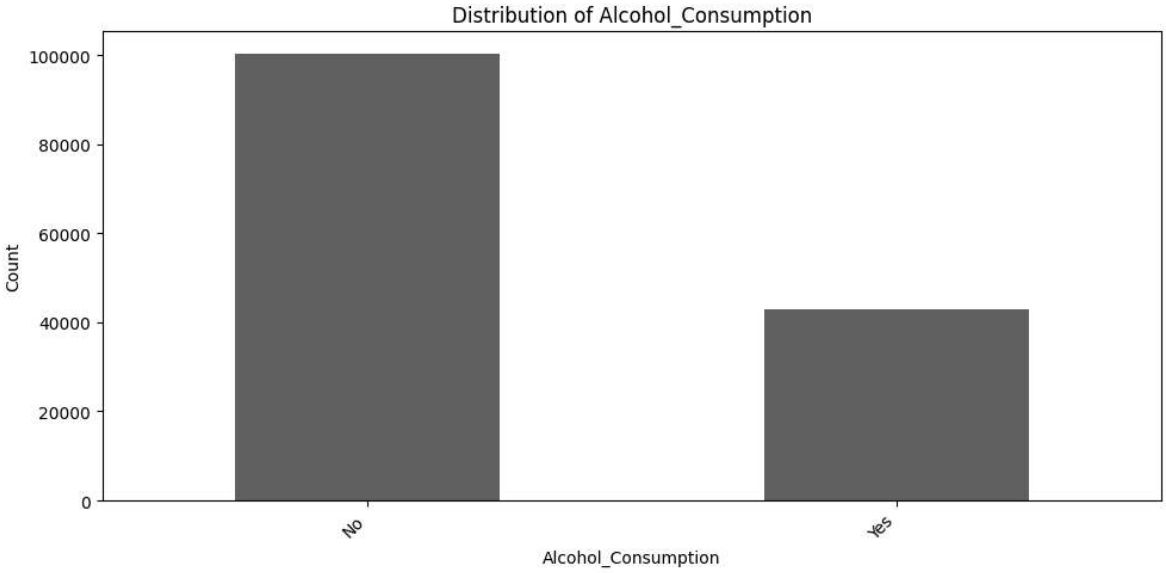


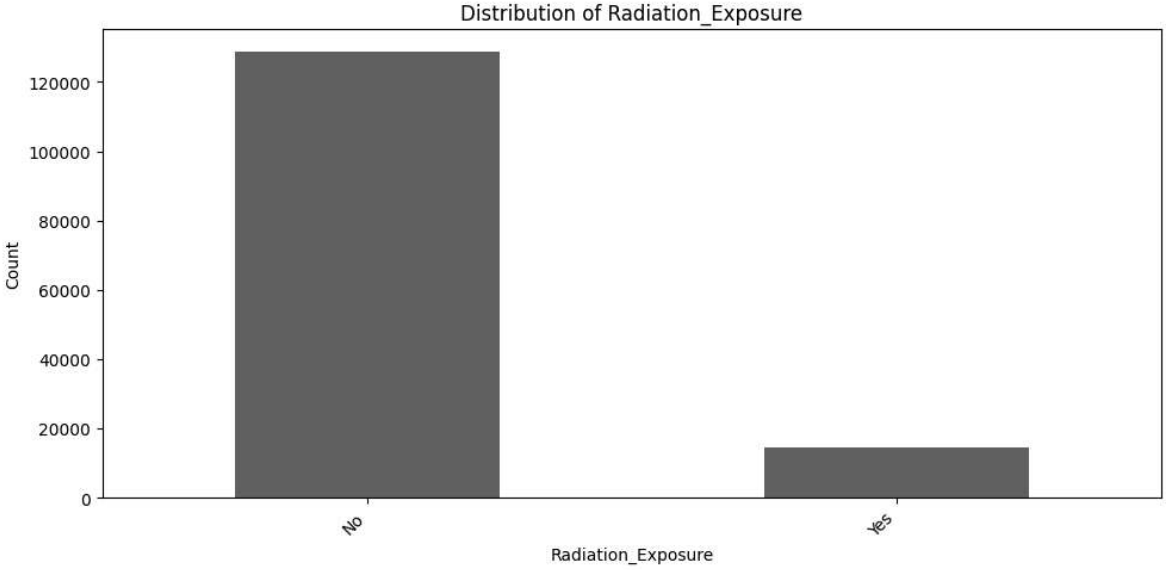


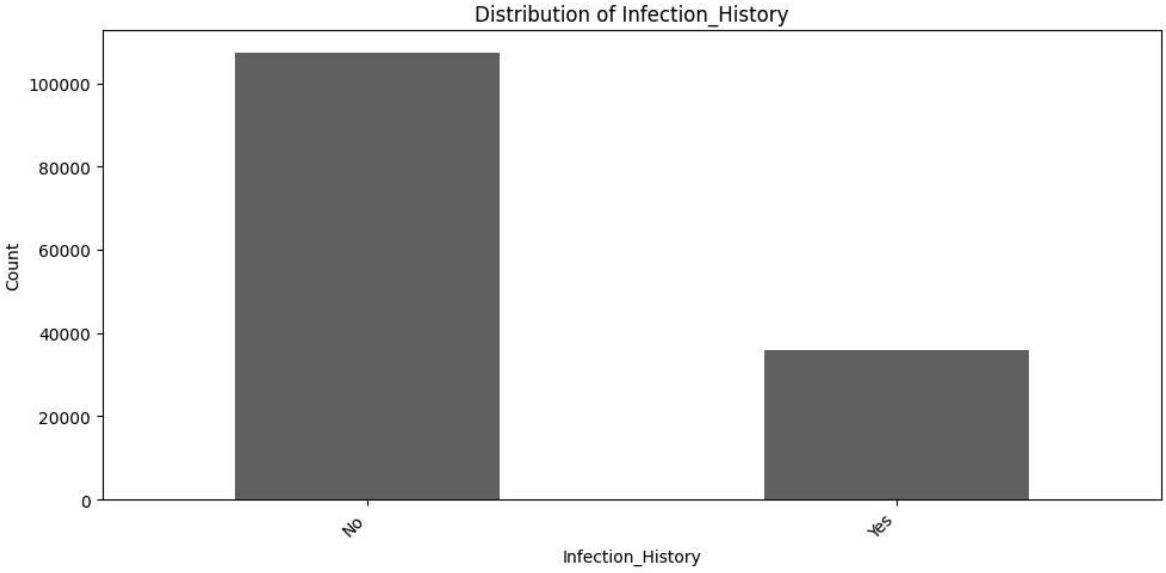

Distribution of Genetic_Mutation
Count
120000
100000
80000
60000
40000
20000
0
No
Yes
Genetic_Mutation

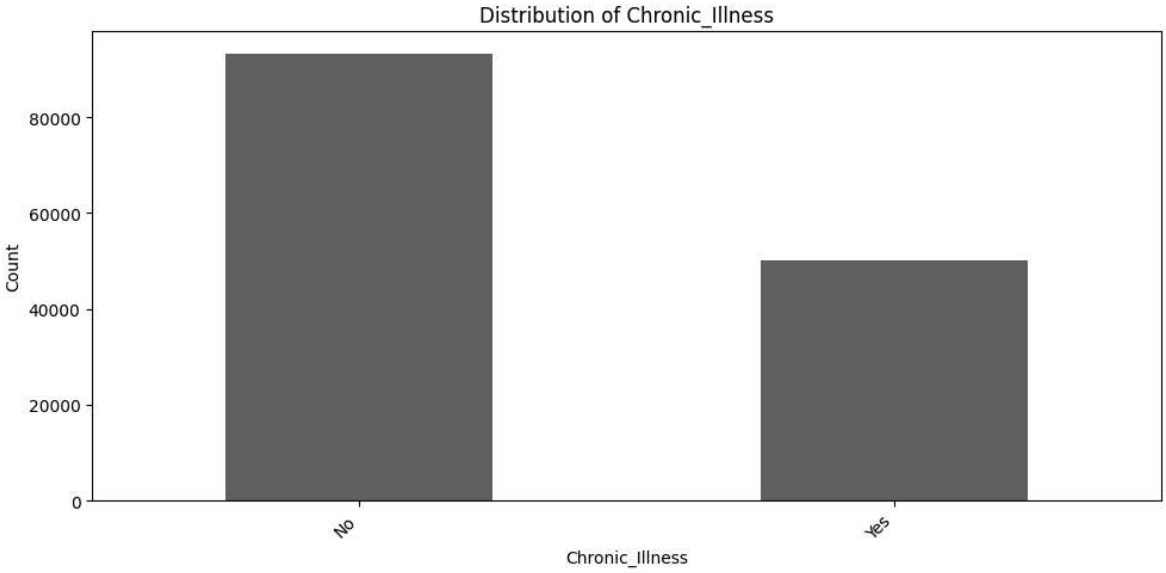

Distribution of Family_History
Count
100000
80000
60000
40000
20000
0
No
Yes
Family_History

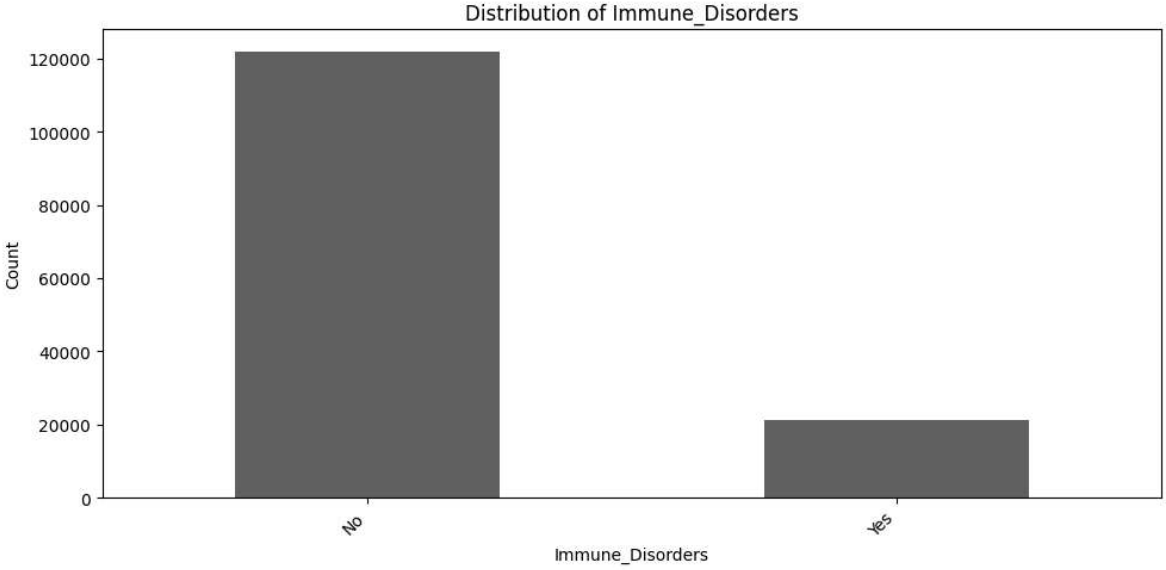

Distribution of Smoking_Status
Count
80000
70000
60000
50000
40000
30000
20000
10000
0
No
Yes
Smoking_Status

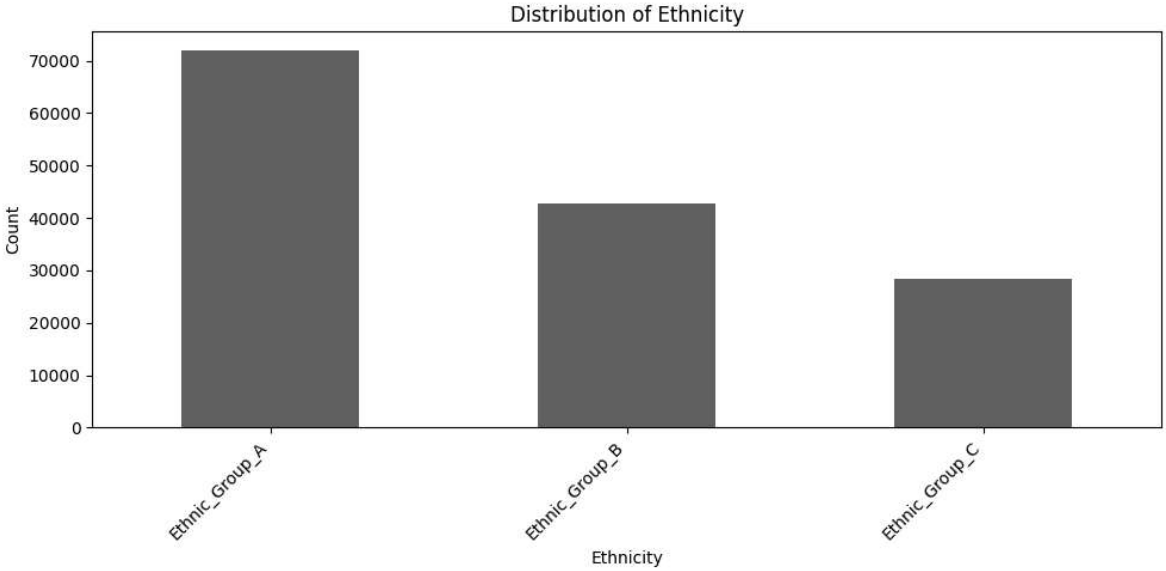

Distribution of Alcohol_Consumption
Count
100000
80000
60000
40000
20000
0
No
Yes
Alcohol_Consumption

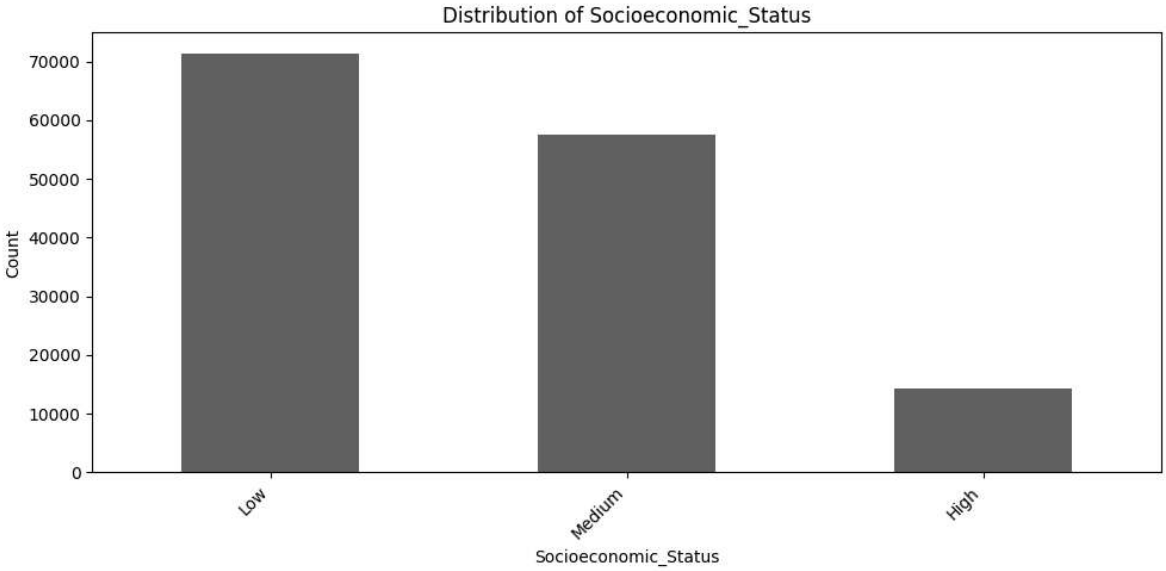

Distribution of Radiation_Exposure
Count
120000
100000
80000
60000
40000
20000
0
No
Yes
Radiation_Exposure

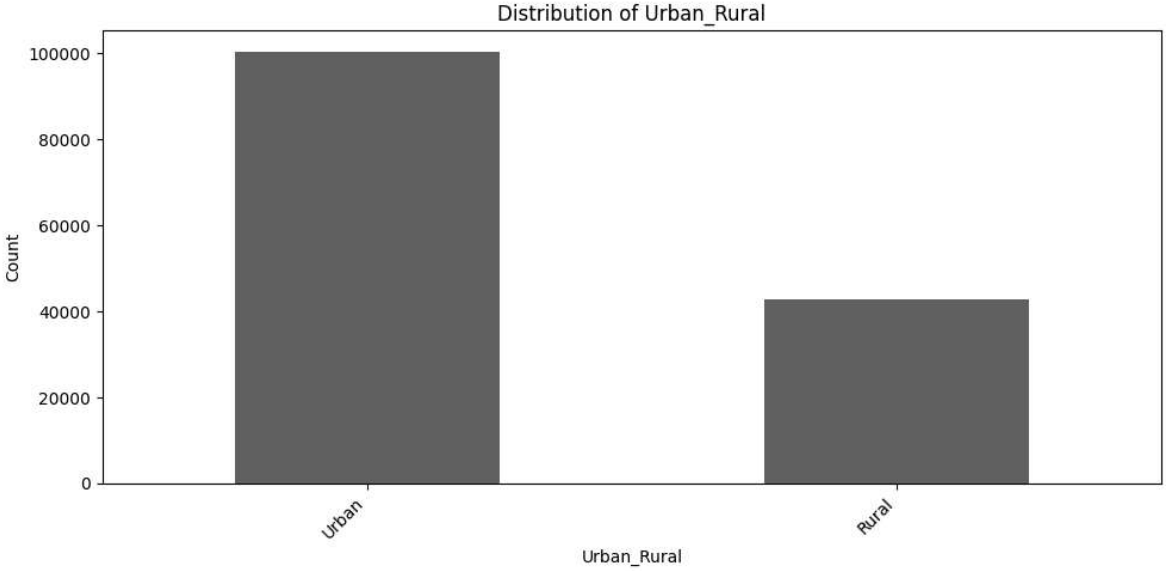

Distribution of Infection_History
Count
100000
80000
60000
40000
20000
0
No
Yes
Infection_History

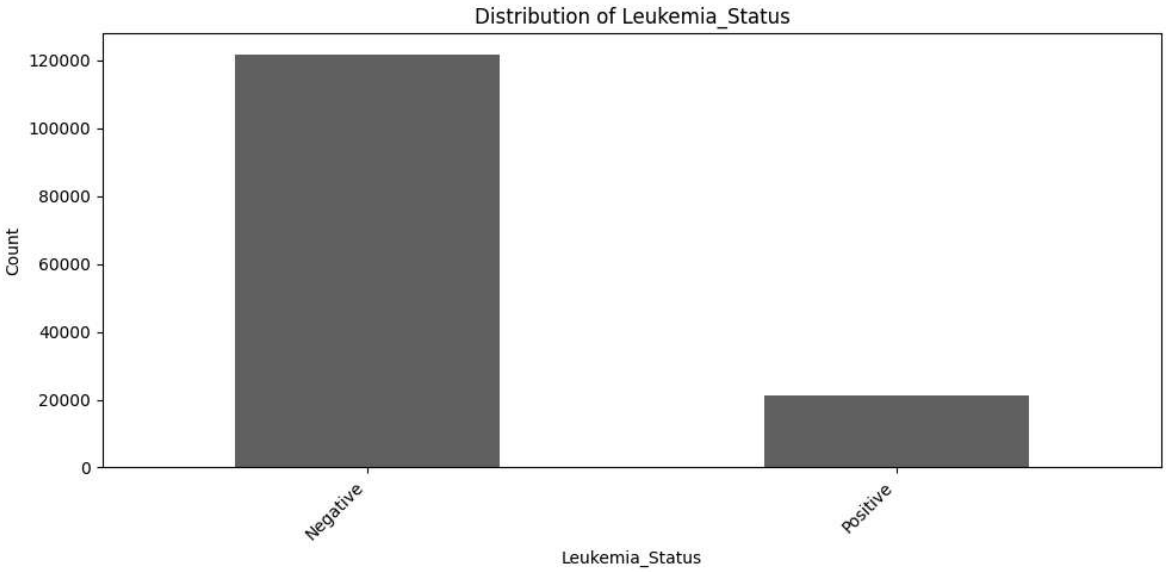

Distribution of Chronic_Illness
Count
0
20000
40000
60000
80000
No
Yes
Chronic_Illness


Distribution of Immune_Disorders
Count
0
20000
40000
60000
80000
100000
120000
No
Yes
Immune_Disorders


Distribution of Ethnicity
Count
0
10000
20000
30000
40000
50000
60000
70000
Ethnic_Group_A
Ethnic_Group_B
Ethnic_Group_C
Ethnicity

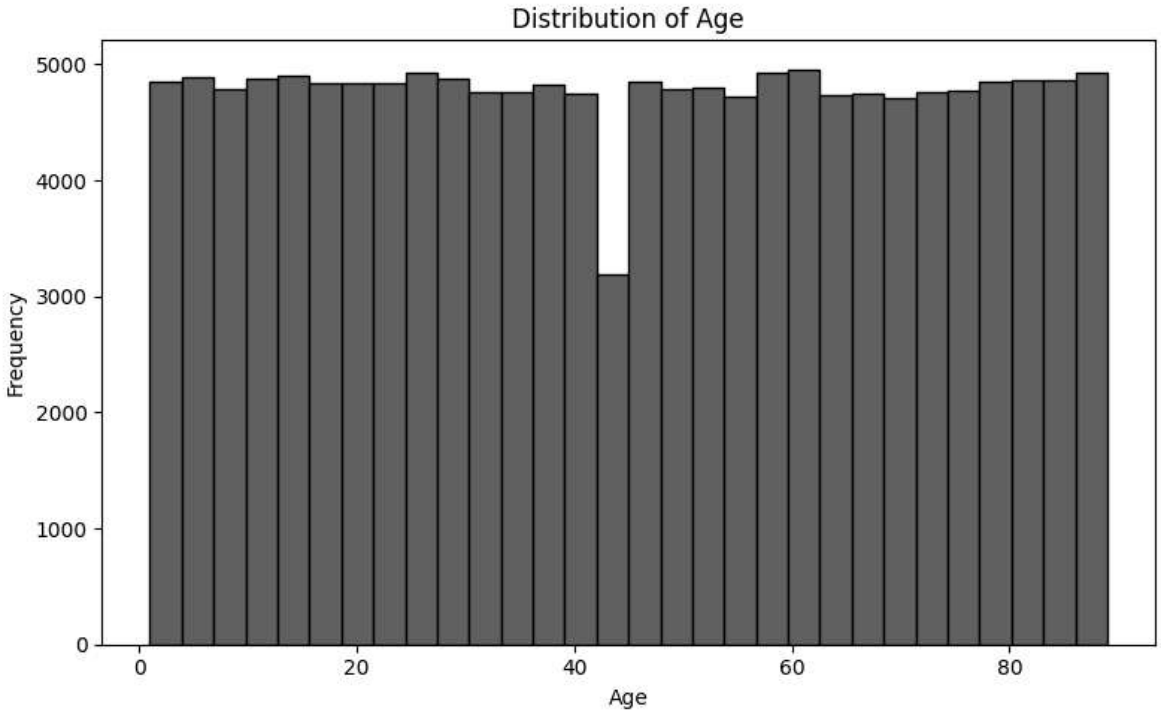


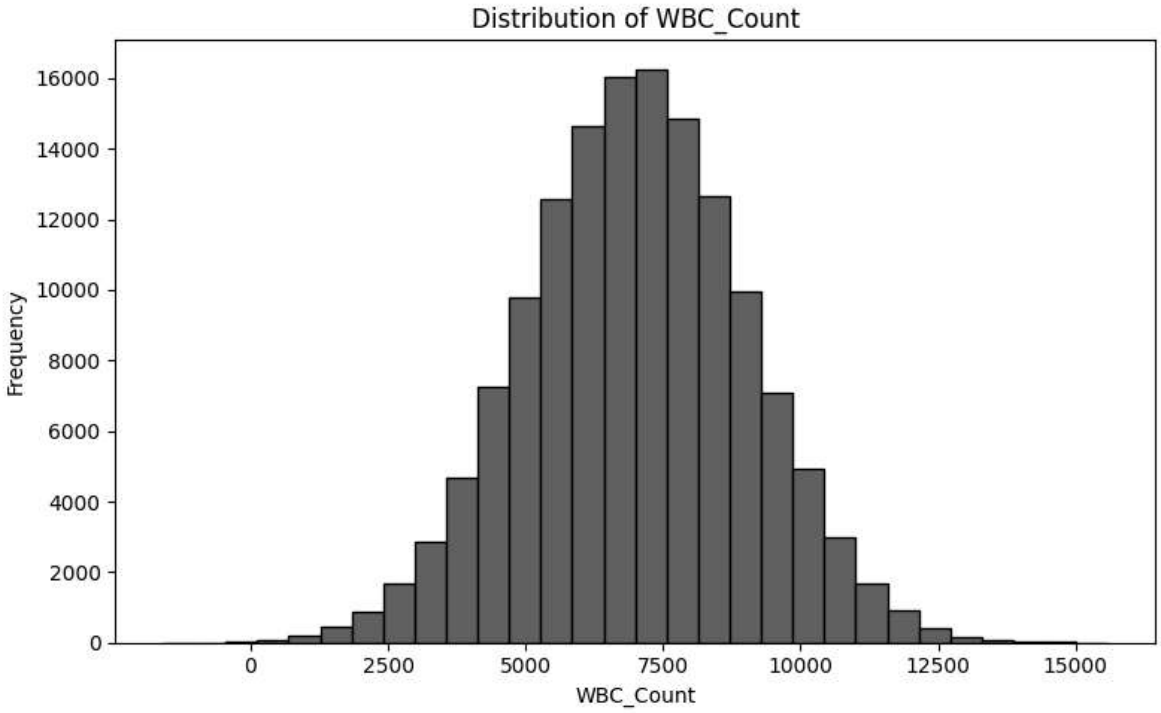


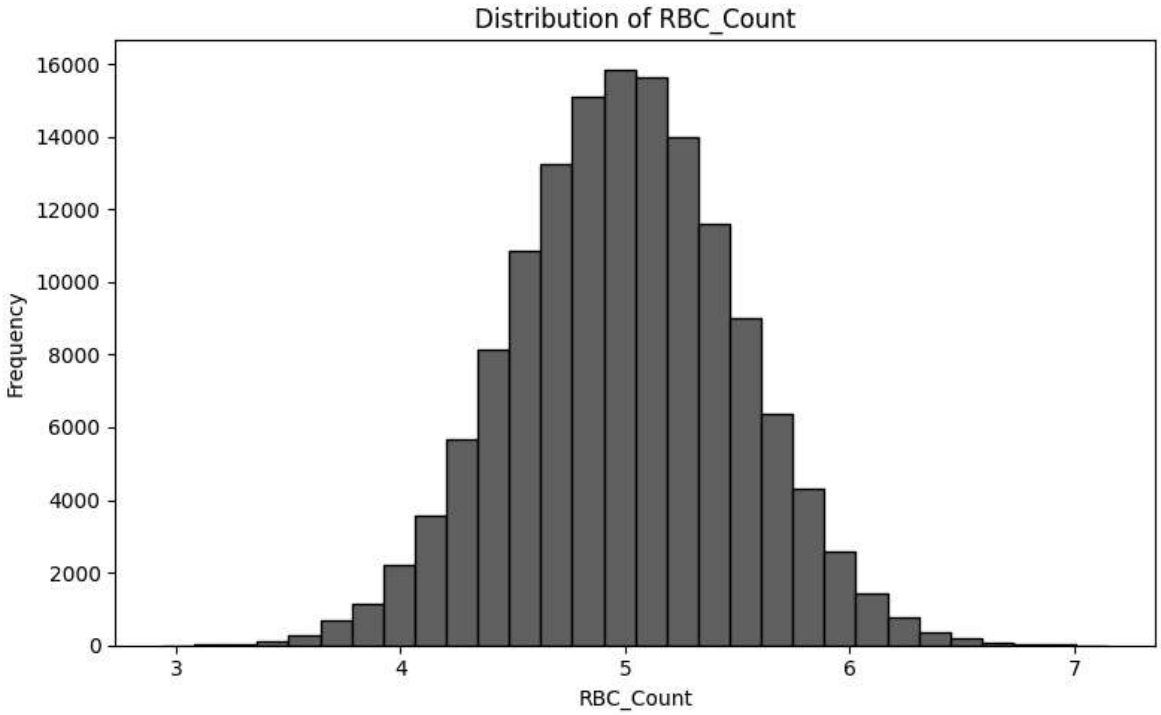


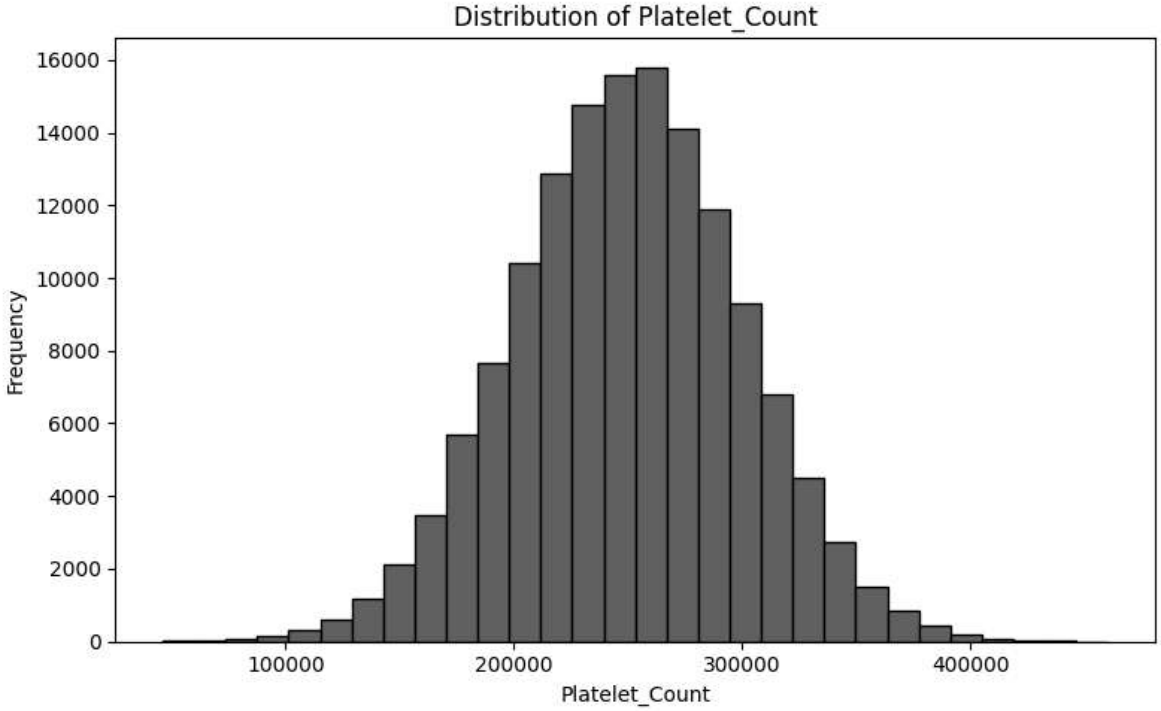
In [ ]:
```
#แยกข้อมูลที่เป็นตัวเลขออกมาดูความแตกต่างของข้อมูล
numeric_cols = ['Age', 'WBC_Count', 'RBC_Count', 'Platelet_Count', 'Hemoglobin_L

for col in numeric_cols:
    plt.figure(figsize=(8, 5))
    plt.hist(dataset[col], bins=30, edgecolor='black')
    plt.title(f'Distribution of {col}')
```

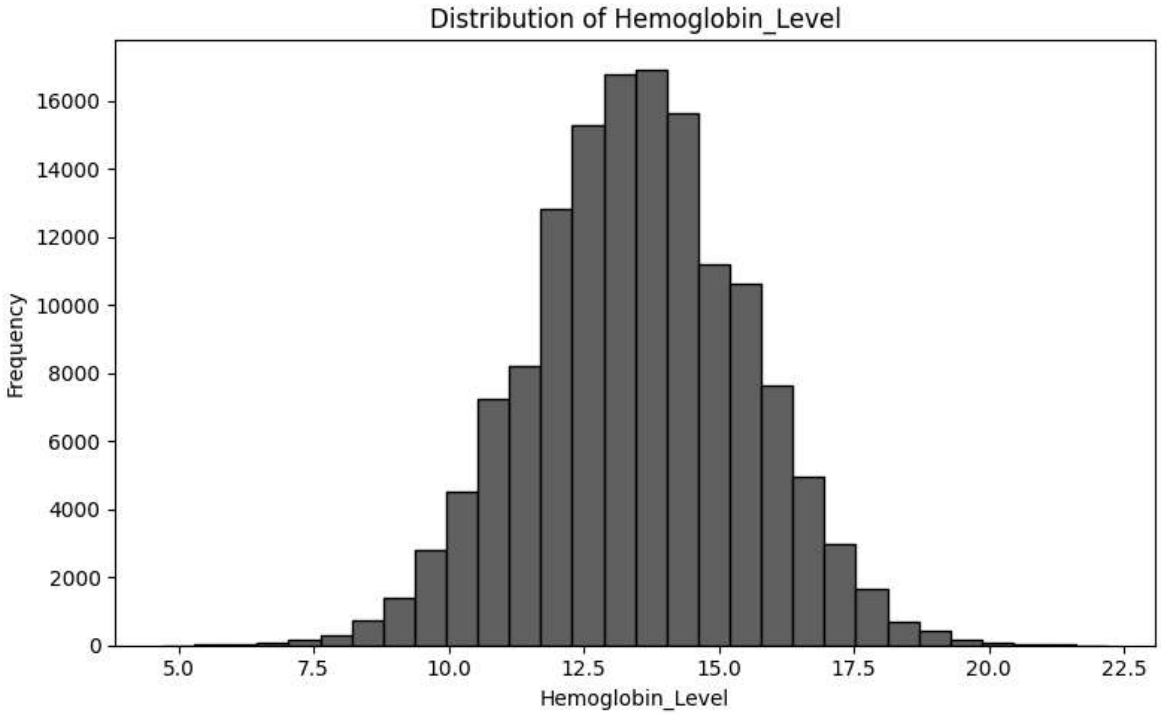
```
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```

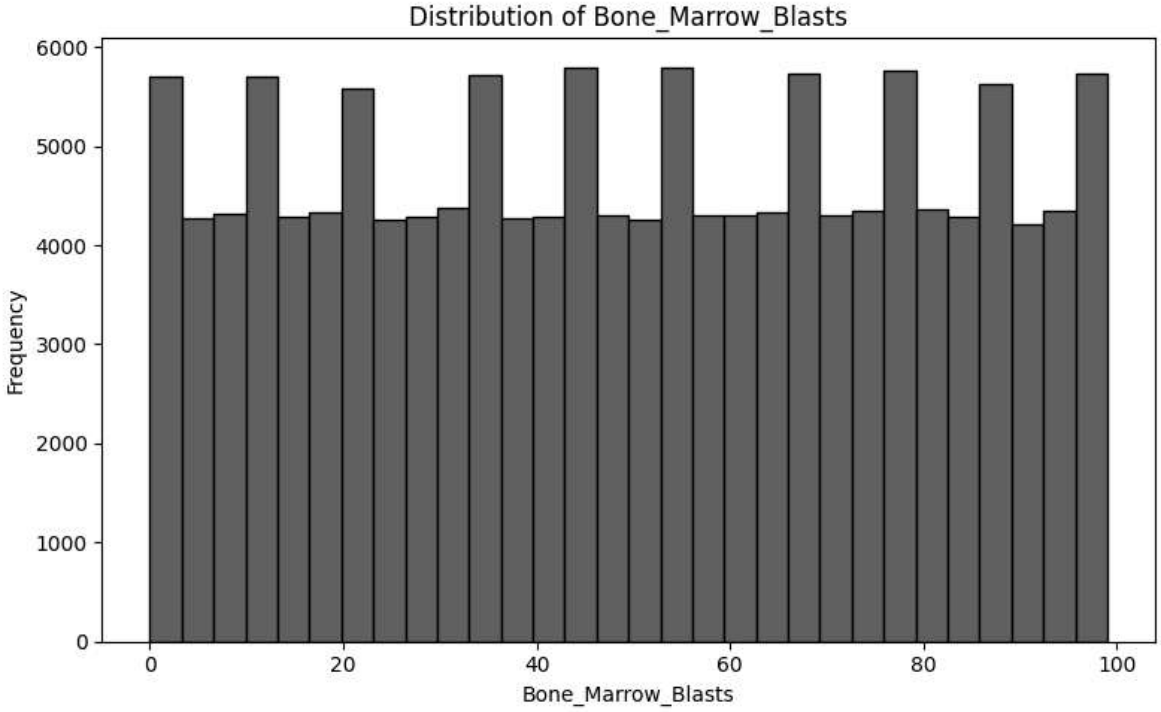


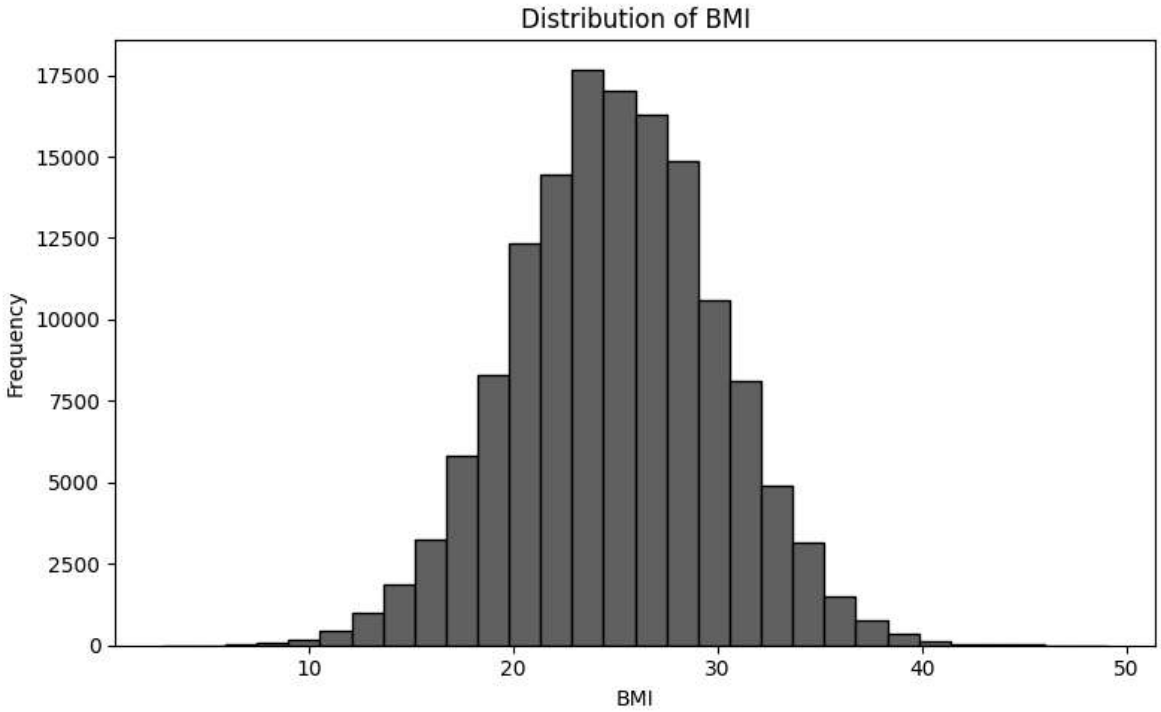



Distribution of RBC_Count
Frequency
RBC_Count
0
2000
4000
6000
8000
10000
12000
14000
16000
3
4
5
6
7


Distribution of Platelet_Count
Frequency
Platelet_Count
0
2000
4000
6000
8000
10000
12000
14000
16000
100000
200000
300000
400000


Distribution of Hemoglobin_Level
Frequency
Hemoglobin_Level
0
2000
4000
6000
8000
10000
12000
14000
16000
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5


Distribution of Bone_Marrow_Blasts
Frequency
Bone_Marrow_Blasts
0
1000
2000
3000
4000
5000
6000
0
20
40
60
80
100



[3] Data Preprocessing

drop Patient ID

In [6]:
```
# Drop Patient_ID และ Country (ไม่ต้องใช้ในการทำนาย)
df = dataset.drop(['Patient_ID', 'Country'], axis=1)
```

Check null data

In [7]:
```
print(df.isnull().sum())
```

```
Age                    0
Gender                 0
WBC_Count              0
RBC_Count              0
Platelet_Count         0
Hemoglobin_Level       0
Bone_Marrow_Blasts     0
Genetic_Mutation       0
Family_History         0
Smoking_Status         0
Alcohol_Consumption    0
Radiation_Exposure     0
Infection_History      0
BMI                    0
Chronic_Illness        0
Immune_Disorders       0
Ethnicity              0
Socioeconomic_Status   0
Urban_Rural            0
Leukemia_Status        0
dtype: int64
```

Encode Categorical Features

In [ ]:
```
#แปลงข้อมูลที่เป็นตัวหนังสือให้เป็นตัวเลข
from sklearn.preprocessing import LabelEncoder

label_cols = ['Gender', 'Genetic_Mutation', 'Family_History', 'Smoking_Status',
              'Alcohol_Consumption', 'Radiation_Exposure', 'Infection_History',
              'Chronic_Illness', 'Immune_Disorders', 'Ethnicity', 'Socioeconomic
              'Urban_Rural', 'Leukemia_Status']

le = LabelEncoder()
for col in label_cols:
    df[col] = le.fit_transform(df[col])
```

In [9]:
```
df.shape
```

Out[9]:
```
(143194, 20)
```

In [10]:
```
df.head()
```

Out[10]:

	Age	Gender	WBC_Count	RBC_Count	Platelet_Count	Hemoglobin_Level	Bone_Mar
0	52	1	2698	5.36	262493	12.2	
1	15	0	4857	4.81	277877	11.9	
2	72	1	9614	5.17	319600	13.4	
3	61	1	6278	5.41	215200	11.6	
4	21	1	8342	4.78	309169	14.3	

Scale Numerical Features

In [ ]:
```
# ทำการ Scale เพื่อที่จะปรับขนาดของข้อมูลตัวเลขให้เหมาะสมสำหรับการฝึก
from sklearn.preprocessing import StandardScaler

num_cols = ['Age', 'WBC_Count', 'RBC_Count', 'Platelet_Count', 'Hemoglobin_Level
            'Bone_Marrow_Blasts', 'BMI']

scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])
```

In [26]:
```
df.head()
```

Out[26]:

	Age	Gender	WBC_Count	RBC_Count	Platelet_Count	Hemoglobin_Level	Bone
0	0.274104	1	-2.143673	0.723185	0.251995	-0.650238	
1	-1.163475	0	-1.068173	-0.376164	0.560650	-0.800485	
2	1.051173	1	1.301515	0.343410	1.397753	-0.049248	
3	0.623785	1	-0.360305	0.823125	-0.696860	-0.950732	
4	-0.930355	1	0.667871	-0.436129	1.188472	0.401494	

train-test split

In [ ]:
```
#ทำข้อมูลมา Split และ drop เป้าหมายสำหรับการ train ออก
from sklearn.model_selection import train_test_split

X = df.drop('Leukemia_Status', axis=1)
y = df['Leukemia_Status']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Handling Class Imbalance using smote

In [ ]:
```
#ใช้ SMOTE เพื่อแก้ปัญหา class imbalance เพื่อให้โมเดลเรียนรู้ได้ดีขึ้น
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

In [29]:
```
# Ensure correct datatypes for TensorFlow
X_train_res = np.array(X_train_res).astype(np.float32)
X_test = np.array(X_test).astype(np.float32)
y_train_res = np.array(y_train_res).astype(np.float32)  # for binary_crossentrop
y_test = np.array(y_test).astype(np.float32)
```

[4] Modeling with SMOTE Data Set

ANN

In [30]:
```
# 1. Import libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc

# 2. Build Model
model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train_res.shape[1],)),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

# 3. Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

# 4. Train Model
history = model.fit(
    X_train_res, y_train_res,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test),
    verbose=1
)

# 5. Predict and Evaluate
y_pred_probs = model.predict(X_test)
```

```
y_pred = (y_pred_probs > 0.5).astype("int32")

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy Score:")
print(f"{accuracy_score(y_test, y_pred):.4f}")
```

```
Epoch 1/50
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 3s 397us/step - accuracy: 0.5989 - loss: 0.6614 -
val_accuracy: 0.5452 - val_loss: 0.7024
Epoch 2/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 389us/step - accuracy: 0.6361 - loss: 0.6402 -
val_accuracy: 0.5410 - val_loss: 0.7052
Epoch 3/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 393us/step - accuracy: 0.6370 - loss: 0.6396 -
val_accuracy: 0.5691 - val_loss: 0.6869
Epoch 4/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 390us/step - accuracy: 0.6400 - loss: 0.6364 -
val_accuracy: 0.5938 - val_loss: 0.6599
Epoch 5/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 382us/step - accuracy: 0.6394 - loss: 0.6369 -
val_accuracy: 0.5713 - val_loss: 0.6869
Epoch 6/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 382us/step - accuracy: 0.6434 - loss: 0.6344 -
val_accuracy: 0.5204 - val_loss: 0.7234
Epoch 7/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 382us/step - accuracy: 0.6442 - loss: 0.6338 -
val_accuracy: 0.5828 - val_loss: 0.6746
Epoch 8/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 383us/step - accuracy: 0.6448 - loss: 0.6324 -
val_accuracy: 0.5868 - val_loss: 0.6728
Epoch 9/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 385us/step - accuracy: 0.6455 - loss: 0.6317 -
val_accuracy: 0.5572 - val_loss: 0.6916
Epoch 10/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 379us/step - accuracy: 0.6474 - loss: 0.6309 -
val_accuracy: 0.5376 - val_loss: 0.7190
Epoch 11/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 377us/step - accuracy: 0.6468 - loss: 0.6314 -
val_accuracy: 0.5698 - val_loss: 0.6860
Epoch 12/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 407us/step - accuracy: 0.6508 - loss: 0.6282 -
val_accuracy: 0.5742 - val_loss: 0.6813
Epoch 13/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 380us/step - accuracy: 0.6485 - loss: 0.6293 -
val_accuracy: 0.5938 - val_loss: 0.6641
Epoch 14/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 382us/step - accuracy: 0.6504 - loss: 0.6273 -
val_accuracy: 0.5865 - val_loss: 0.6694
Epoch 15/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 384us/step - accuracy: 0.6498 - loss: 0.6287 -
val_accuracy: 0.5778 - val_loss: 0.6796
Epoch 16/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 394us/step - accuracy: 0.6517 - loss: 0.6267 -
val_accuracy: 0.5868 - val_loss: 0.6697
Epoch 17/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 378us/step - accuracy: 0.6501 - loss: 0.6281 -
val_accuracy: 0.5879 - val_loss: 0.6698
Epoch 18/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 405us/step - accuracy: 0.6531 - loss: 0.6259 -
val_accuracy: 0.6229 - val_loss: 0.6404
Epoch 19/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 381us/step - accuracy: 0.6534 - loss: 0.6254 -
val_accuracy: 0.5819 - val_loss: 0.6686
Epoch 20/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 381us/step - accuracy: 0.6510 - loss: 0.6272 -
val_accuracy: 0.5741 - val_loss: 0.6783
Epoch 21/50
```

```
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 385us/step - accuracy: 0.6540 - loss: 0.6251 -
val_accuracy: 0.5620 - val_loss: 0.6860
Epoch 22/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 378us/step - accuracy: 0.6515 - loss: 0.6273 -
val_accuracy: 0.5604 - val_loss: 0.6921
Epoch 23/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 379us/step - accuracy: 0.6529 - loss: 0.6257 -
val_accuracy: 0.5641 - val_loss: 0.6936
Epoch 24/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 386us/step - accuracy: 0.6535 - loss: 0.6257 -
val_accuracy: 0.5886 - val_loss: 0.6639
Epoch 25/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 400us/step - accuracy: 0.6545 - loss: 0.6239 -
val_accuracy: 0.5893 - val_loss: 0.6677
Epoch 26/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 384us/step - accuracy: 0.6541 - loss: 0.6253 -
val_accuracy: 0.5998 - val_loss: 0.6610
Epoch 27/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 389us/step - accuracy: 0.6539 - loss: 0.6245 -
val_accuracy: 0.6099 - val_loss: 0.6474
Epoch 28/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 379us/step - accuracy: 0.6522 - loss: 0.6269 -
val_accuracy: 0.5673 - val_loss: 0.6878
Epoch 29/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 379us/step - accuracy: 0.6547 - loss: 0.6247 -
val_accuracy: 0.5758 - val_loss: 0.6817
Epoch 30/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 386us/step - accuracy: 0.6533 - loss: 0.6251 -
val_accuracy: 0.5768 - val_loss: 0.6793
Epoch 31/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 382us/step - accuracy: 0.6545 - loss: 0.6246 -
val_accuracy: 0.5648 - val_loss: 0.6898
Epoch 32/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 387us/step - accuracy: 0.6543 - loss: 0.6252 -
val_accuracy: 0.5638 - val_loss: 0.6906
Epoch 33/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 401us/step - accuracy: 0.6539 - loss: 0.6246 -
val_accuracy: 0.5504 - val_loss: 0.7018
Epoch 34/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 384us/step - accuracy: 0.6562 - loss: 0.6235 -
val_accuracy: 0.5818 - val_loss: 0.6782
Epoch 35/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 384us/step - accuracy: 0.6555 - loss: 0.6231 -
val_accuracy: 0.5727 - val_loss: 0.6815
Epoch 36/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 394us/step - accuracy: 0.6539 - loss: 0.6241 -
val_accuracy: 0.5861 - val_loss: 0.6696
Epoch 37/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 384us/step - accuracy: 0.6551 - loss: 0.6238 -
val_accuracy: 0.5739 - val_loss: 0.6792
Epoch 38/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 384us/step - accuracy: 0.6544 - loss: 0.6240 -
val_accuracy: 0.6001 - val_loss: 0.6568
Epoch 39/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 397us/step - accuracy: 0.6539 - loss: 0.6249 -
val_accuracy: 0.5668 - val_loss: 0.6866
Epoch 40/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 390us/step - accuracy: 0.6536 - loss: 0.6239 -
val_accuracy: 0.5714 - val_loss: 0.6885
Epoch 41/50
```

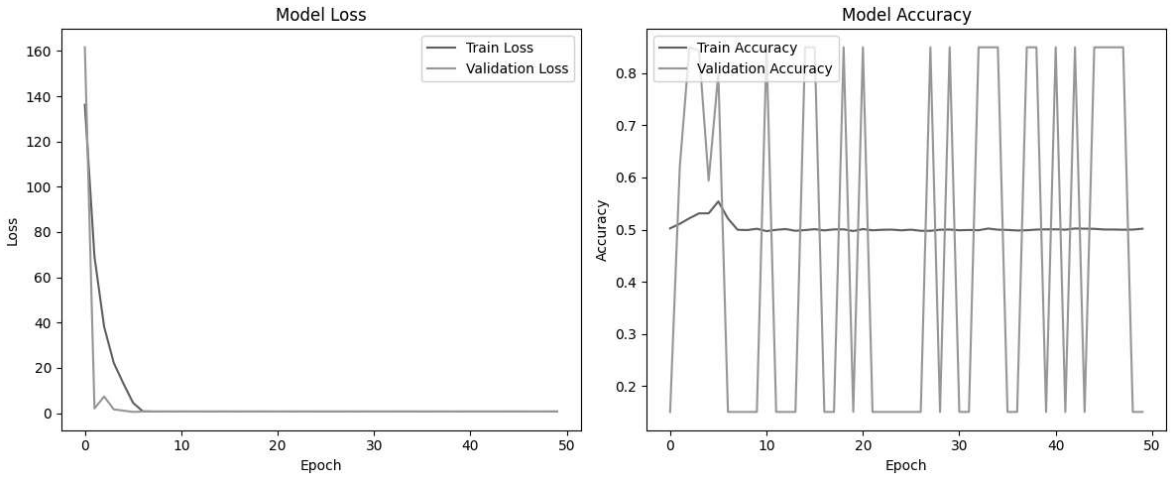
```
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 379us/step - accuracy: 0.6572 - loss: 0.6226 -
val_accuracy: 0.6039 - val_loss: 0.6547
Epoch 42/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 385us/step - accuracy: 0.6545 - loss: 0.6241 -
val_accuracy: 0.5752 - val_loss: 0.6806
Epoch 43/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 376us/step - accuracy: 0.6558 - loss: 0.6231 -
val_accuracy: 0.5785 - val_loss: 0.6797
Epoch 44/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 378us/step - accuracy: 0.6539 - loss: 0.6247 -
val_accuracy: 0.5670 - val_loss: 0.6907
Epoch 45/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 383us/step - accuracy: 0.6546 - loss: 0.6237 -
val_accuracy: 0.5901 - val_loss: 0.6684
Epoch 46/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 391us/step - accuracy: 0.6565 - loss: 0.6226 -
val_accuracy: 0.5804 - val_loss: 0.6769
Epoch 47/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 385us/step - accuracy: 0.6566 - loss: 0.6234 -
val_accuracy: 0.5754 - val_loss: 0.6817
Epoch 48/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 377us/step - accuracy: 0.6559 - loss: 0.6235 -
val_accuracy: 0.5645 - val_loss: 0.6895
Epoch 49/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 383us/step - accuracy: 0.6566 - loss: 0.6222 -
val_accuracy: 0.5532 - val_loss: 0.6993
Epoch 50/50
6092/6092 ━━━━━━━━━━━━━━━━━━━━ 2s 402us/step - accuracy: 0.6527 - loss: 0.6249 -
val_accuracy: 0.5725 - val_loss: 0.6849
895/895 ━━━━━━━━━━━━━━━━━━━━ 0s 207us/step

Confusion Matrix:
[[14680  9648]
 [ 2596  1715]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.60      0.71     24328
         1.0       0.15      0.40      0.22      4311

    accuracy                           0.57     28639
   macro avg       0.50      0.50      0.46     28639
weighted avg       0.74      0.57      0.63     28639


Accuracy Score:
0.5725
```

In [31]:
```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[14680  9648]
 [ 2596  1715]]
```

In [17]:
```
import matplotlib.pyplot as plt

# Plot training & validation loss
plt.figure(figsize=(12,5))
```

```
# Loss
plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Accuracy
plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

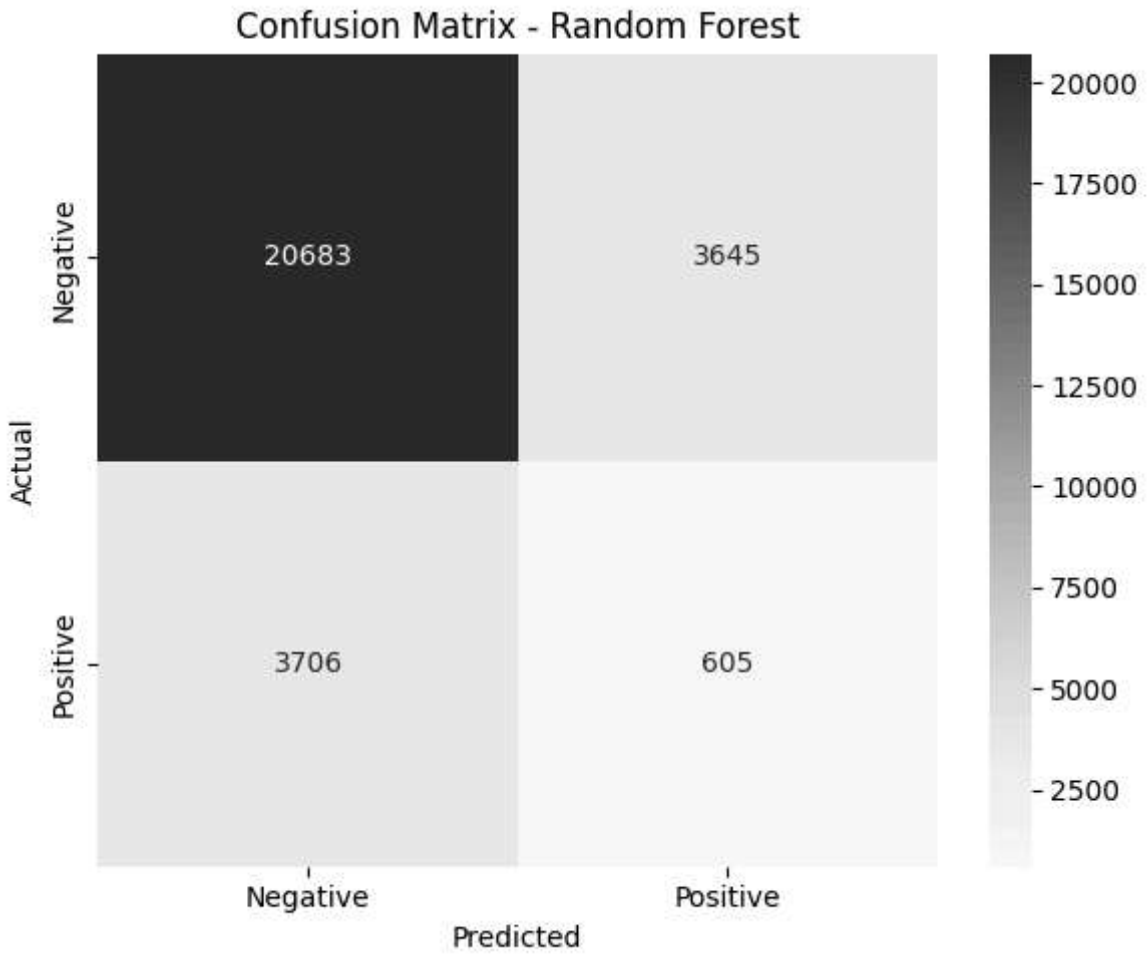


Random Forest

In [18]:
```
# 1. Import libraries
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc

# 2. Create and Train Random Forest Model
rf_model = RandomForestClassifier(
    n_estimators=100,     # number of trees
    random_state=42,
    n_jobs=-1             # use all CPU cores
)
rf_model.fit(X_train_res, y_train_res)

# 3. Predict and Evaluate
y_pred = rf_model.predict(X_test)

# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"✅ Random Forest Model Accuracy: {acc:.4f}")
```

```
# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
conf_mat = confusion_matrix(y_test, y_pred)

# 4. Plot Confusion Matrix
plt.figure(figsize=(6,5))
sns.heatmap(
    conf_mat,
    annot=True,
    fmt='d',
    cmap='Greens',
    xticklabels=['Negative', 'Positive'],
    yticklabels=['Negative', 'Positive']
)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest')
plt.tight_layout()
plt.show()
```

```
Random Forest Model Accuracy: 0.7433

Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.85      0.85     24328
         1.0       0.14      0.14      0.14      4311

    accuracy                           0.74     28639
   macro avg       0.50      0.50      0.50     28639
weighted avg       0.74      0.74      0.74     28639
```



TabNet

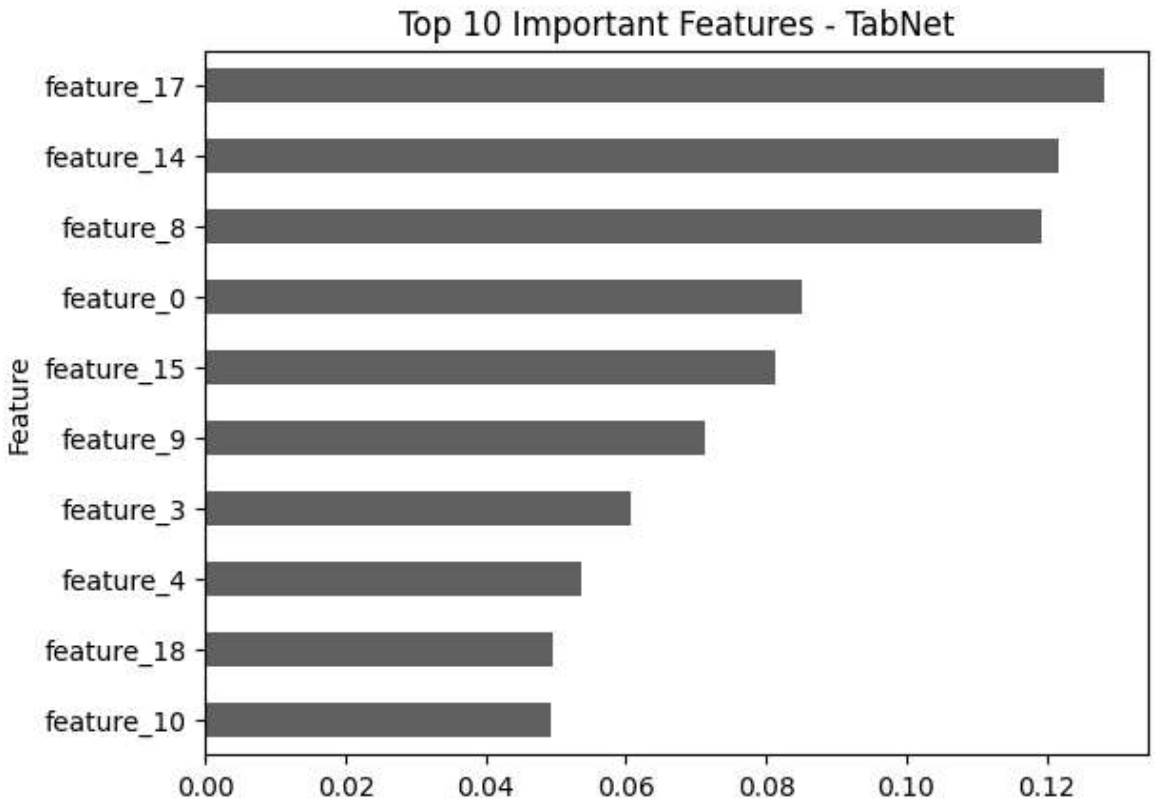
In [19]:
```
# 1. Import libraries
import matplotlib.pyplot as plt
from pytorch_tabnet.tab_model import TabNetClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc

# 2. Create and Train TabNet Model
clf = TabNetClassifier(
    verbose=1,
    seed=42
)

clf.fit(
    X_train_res, y_train_res,
    eval_set=[(X_test, y_test)],
    eval_name=['valid'],
    eval_metric=['accuracy'],
    max_epochs=50,
    patience=10,
    batch_size=1024,
    virtual_batch_size=128
)

# 3. Predict
y_pred = clf.predict(X_test)

# 4. Evaluate
print(f"✅ Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pytorch_tabnet/abstract_model.py:82: UserWarning: Device used : cpu
  warnings.warn(f"Device used : {self.device}")
epoch 0  | loss: 0.5869  | valid_accuracy: 0.78173 |  0:00:03s
epoch 1  | loss: 0.51661 | valid_accuracy: 0.72007 |  0:00:06s
epoch 2  | loss: 0.50692 | valid_accuracy: 0.69112 |  0:00:09s
epoch 3  | loss: 0.50395 | valid_accuracy: 0.695   |  0:00:12s
epoch 4  | loss: 0.49982 | valid_accuracy: 0.69713 |  0:00:15s
epoch 5  | loss: 0.49601 | valid_accuracy: 0.70366 |  0:00:18s
epoch 6  | loss: 0.49221 | valid_accuracy: 0.71259 |  0:00:22s
epoch 7  | loss: 0.49104 | valid_accuracy: 0.72733 |  0:00:25s
epoch 8  | loss: 0.4907  | valid_accuracy: 0.70565 |  0:00:29s
epoch 9  | loss: 0.49004 | valid_accuracy: 0.71176 |  0:00:32s
epoch 10 | loss: 0.49009 | valid_accuracy: 0.70568 |  0:00:36s

Early stopping occurred at epoch 10 with best_epoch = 0 and best_valid_accuracy = 0.78173
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pytorch_tabnet/callbacks.py:172: UserWarning: Best weights from best epoch are automatically used!
  warnings.warn(wrn_msg)
✅ Test Accuracy: 0.7817

Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.91      0.88     24328
         1.0       0.14      0.08      0.10      4311

    accuracy                           0.78     28639
   macro avg       0.49      0.50      0.49     28639
weighted avg       0.74      0.78      0.76     28639


Confusion Matrix:
[[22022  2306]
 [ 3945   366]]
```

In [20]:

```
import pandas as pd

feat_importances = clf.feature_importances_
feature_names = [f'feature_{i}' for i in range(X_train_res.shape[1])]  # or use

feat_imp_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feat_importances
}).sort_values('Importance', ascending=False)
# Plot
feat_imp_df.head(10).plot.barh(x='Feature', y='Importance', legend=False)
plt.title('Top 10 Important Features - TabNet')
plt.gca().invert_yaxis()
plt.show()
```


Top 10 Important Features - TabNet
Feature
feature_17
feature_14
feature_8
feature_0
feature_15
feature_9
feature_3
feature_4
feature_18
feature_10
0.00
0.02
0.04
0.06
0.08
0.10
0.12

In [ ]:

In [ ]: