Problem Statement: Prostate Cancer Detection

Develop a machine learning model to predict the risk of prostate cancer based on patient health metrics. The model should classify individuals as "Patient suffering from Prostate Cancer" or "The patient doesn't have prostate cancer" based on relevant medical factors such as age, family history, PSA level, screening age, and prostate volume.

```
! kaggle datasets download ankushpanday1/prostate-cancer-prediction-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/ankushpanday1/prostate-cancer-prediction-dataset
License(s): MIT
Downloading prostate-cancer-prediction-dataset.zip to /content
  0% 0.00/665k [00:00<?, ?B/s]
100% 665k/665k [00:00<00:00, 54.1MB/s]
```

```
! unzip prostate-cancer-prediction-dataset
```

```
Archive:  prostate-cancer-prediction-dataset.zip
  inflating: prostate_cancer_prediction.csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('/content/prostate_cancer_prediction.csv')
pd.set_option('display.max_columns',None)
df
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.columns
```

```
Index(['Patient_ID', 'Age', 'Family_History', 'Race_African_Ancestry',
       'PSA_Level', 'DRE_Result', 'Biopsy_Result',
'Difficulty_Urinating',
       'Weak_Urine_Flow', 'Blood_in_Urine', 'Pelvic_Pain',
'Back_Pain',
       'Erectile_Dysfunction', 'Cancer_Stage',
'Treatment_Recommended',
       'Survival_5_Years', 'Exercise_Regularly', 'Healthy_Diet',
```

```
'BMI',
       'Smoking_History', 'Alcohol_Consumption', 'Hypertension',
'Diabetes',
       'Cholesterol_Level', 'Screening_Age', 'Follow_Up_Required',
       'Prostate_Volume', 'Genetic_Risk_Factors',
'Previous_Cancer_History',
       'Early_Detection'],
      dtype='object')
```

Changing column names to more specified column names for better understanding and analysis

```
header = ['Patient ID', 'Age', 'Family History', 'Race African
Ancestry',
       'PSA Level', 'DRE Result', 'Biopsy Result', 'Difficulty
Urinating',
       'Weak Urine Flow', 'Blood in Urine', 'Pelvic Pain', 'Back
Pain',
       'Erectile Dysfunction', 'Cancer Stage', 'Treatment
Recommended',
       'Survival 5 Years', 'Exercise Regularly', 'Healthy Diet',
'BMI',
       'Smoking History', 'Alcohol Consumption', 'Hypertension',
'Diabetes',
       'Cholesterol Level', 'Screening Age', 'Follow Up Required',
       'Prostate Volume', 'Genetic Risk Factors', 'Previous Cancer
History',
       'Early Detection']

df.columns = header

df.columns # Changing column names to more specified names

Index(['Patient ID', 'Age', 'Family History', 'Race African Ancestry',
       'PSA Level', 'DRE Result', 'Biopsy Result', 'Difficulty
Urinating',
       'Weak Urine Flow', 'Blood in Urine', 'Pelvic Pain', 'Back
Pain',
       'Erectile Dysfunction', 'Cancer Stage', 'Treatment
Recommended',
       'Survival 5 Years', 'Exercise Regularly', 'Healthy Diet',
'BMI',
       'Smoking History', 'Alcohol Consumption', 'Hypertension',
'Diabetes',
       'Cholesterol Level', 'Screening Age', 'Follow Up Required',
       'Prostate Volume', 'Genetic Risk Factors', 'Previous Cancer
History',
       'Early Detection'],
      dtype='object')

df.head()
```

{"type":"dataframe","variable_name":"df"}

Checking for null values through out the data since there are no null values we move to next part

```
df.isnull().sum()
```

```
Patient ID                  0
Age                         0
Family History              0
Race African Ancestry       0
PSA Level                   0
DRE Result                  0
Biopsy Result               0
Difficulty Urinating        0
Weak Urine Flow             0
Blood in Urine              0
Pelvic Pain                 0
Back Pain                   0
Erectile Dysfunction        0
Cancer Stage                0
Treatment Recommended       0
Survival 5 Years            0
Exercise Regularly          0
Healthy Diet                0
BMI                         0
Smoking History             0
Alcohol Consumption         0
Hypertension                0
Diabetes                    0
Cholesterol Level           0
Screening Age               0
Follow Up Required          0
Prostate Volume             0
Genetic Risk Factors        0
Previous Cancer History     0
Early Detection             0
dtype: int64
```

```
df.dtypes
```

```
Patient ID                   int64
Age                          int64
Family History              object
Race African Ancestry       object
PSA Level                  float64
DRE Result                  object
Biopsy Result               object
Difficulty Urinating        object
Weak Urine Flow             object
Blood in Urine              object
```

```
Pelvic Pain                  object
Back Pain                    object
Erectile Dysfunction         object
Cancer Stage                 object
Treatment Recommended        object
Survival 5 Years             object
Exercise Regularly           object
Healthy Diet                 object
BMI                         float64
Smoking History              object
Alcohol Consumption          object
Hypertension                 object
Diabetes                     object
Cholesterol Level            object
Screening Age                 int64
Follow Up Required           object
Prostate Volume             float64
Genetic Risk Factors         object
Previous Cancer History      object
Early Detection              object
dtype: object
```

```
df.head()
```


```
{"type":"dataframe","variable_name":"df"}
```


Now its important to observe statistical parameters or descriptive statistics

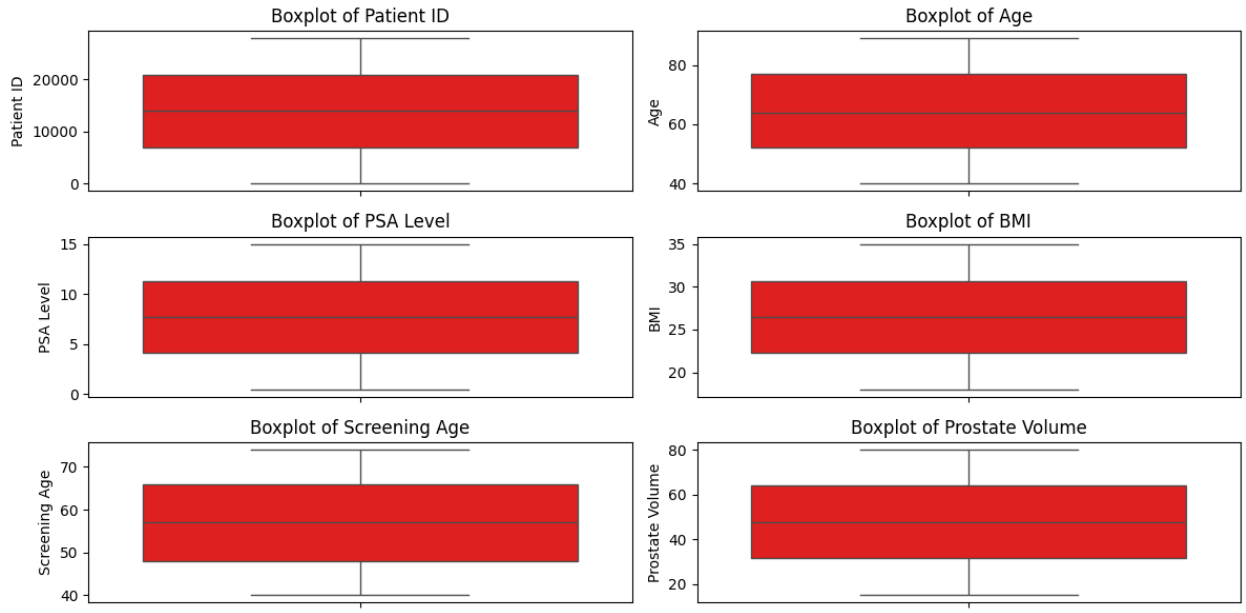
```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Patient ID\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 10071.208405520398,\n
\"min\": 1.0,\n        \"max\": 27945.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          27945.0,\
n          13973.0,\n          20959.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 9859.829264834174,\n
\"min\": 14.404755207372583,\n        \"max\": 27945.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
64.45993916621936,\n          64.0,\n          27945.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PSA Level\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9877.493496137951,\n        \"min\": 0.5,\n        \"max\": 27945.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
7.7515992127393085,\n          7.75,\n          27945.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BMI\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 9871.775503277991,\n
\"min\": 4.888292533977688,\n        \"max\": 27945.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
26.511604938271606,\n          26.5,\n          27945.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Screening Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9862.28878103169,\n        \"min\": 10.11806431543298,\n
\"max\": 27945.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          56.90230810520666,\n          57.0,\n
27945.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Prostate Volume\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 9864.68105305012,\n        \"min\":
15.0,\n        \"max\": 27945.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          47.75577026301664,\n          47.7,\n
27945.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Selecting on numerical columns for outlier detection and after observing these numerical column has on outlier hence we can move to visualization part

```
selected = ['Patient ID',  'Age',     'PSA Level',    'BMI',
     'Screening Age','Prostate Volume']

# from boxplot we can observe that there are no outlier through out
the numerical data

plt.figure(figsize=(12,6))
num_rows=3
num_cols=2

for i, column in enumerate(selected, 1):
  plt.subplot(num_rows, num_cols, i)
  sns.boxplot(y=df[column],color='red')
  plt.title(f'Boxplot of {column}')
  plt.tight_layout()
plt.show()
```



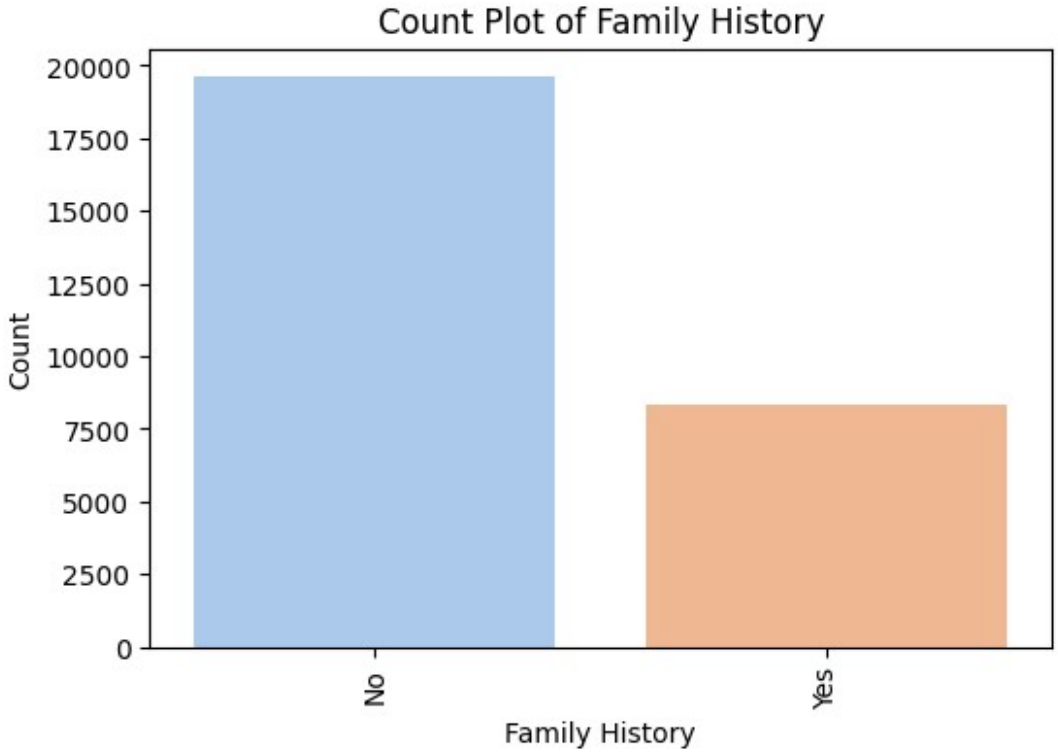

```
df.head()
```

{"type":"dataframe","variable_name":"df"}

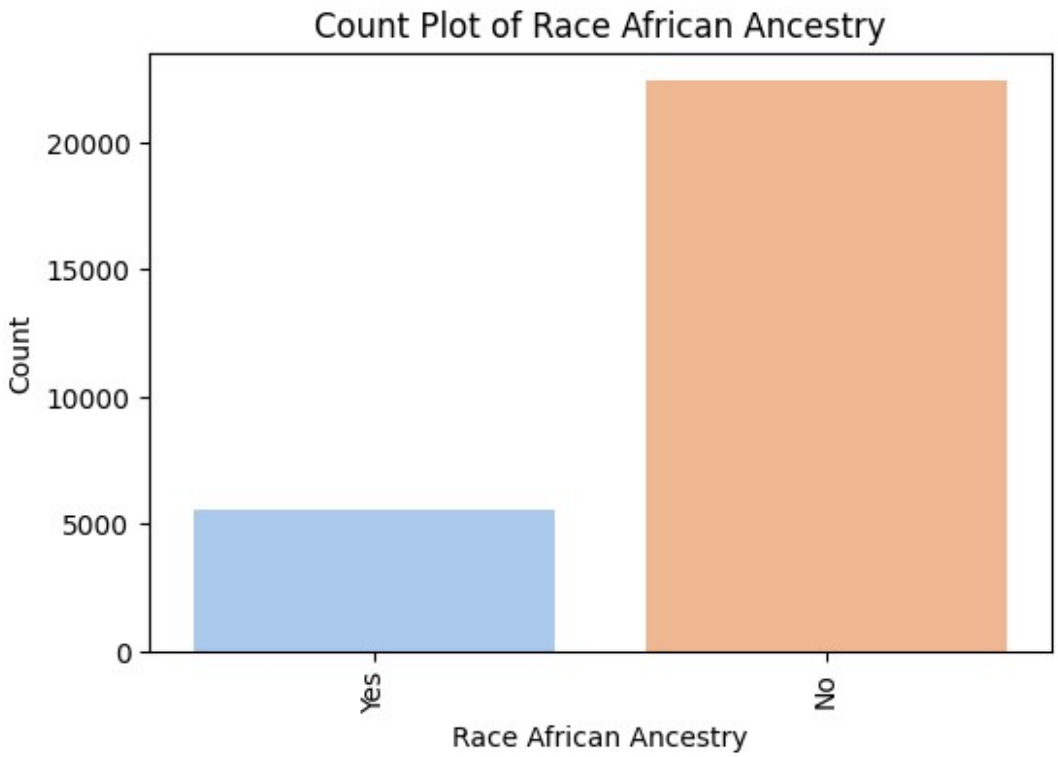
Now we will visualize the categorical column such as Family history, difficulty urnating, blood in urine etc these are the term that will give us deeper idea about the pas treatment plans or if any family member had prostate cancer before which can trigger the cancer in respective patient due to gene similarity.

```
category_col = ['Family History', 'Race African Ancestry', 'DRE
Result',
       'Biopsy Result', 'Difficulty Urinating', 'Weak Urine Flow',
       'Blood in Urine', 'Pelvic Pain', 'Back Pain', 'Erectile
Dysfunction',
       'Cancer Stage', 'Treatment Recommended', 'Survival 5 Years',
       'Exercise Regularly', 'Healthy Diet', 'Smoking History',
       'Alcohol Consumption', 'Hypertension', 'Diabetes', 'Cholesterol
Level',
       'Follow Up Required', 'Genetic Risk Factors', 'Previous Cancer
History',
       'Early Detection']

for col in category_col:
    plt.figure(figsize=(6, 4))
    sns.countplot(x=df[col], palette="pastel")
    plt.title(f'Count Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=90)  # Rotate labels if needed
    plt.show()
```

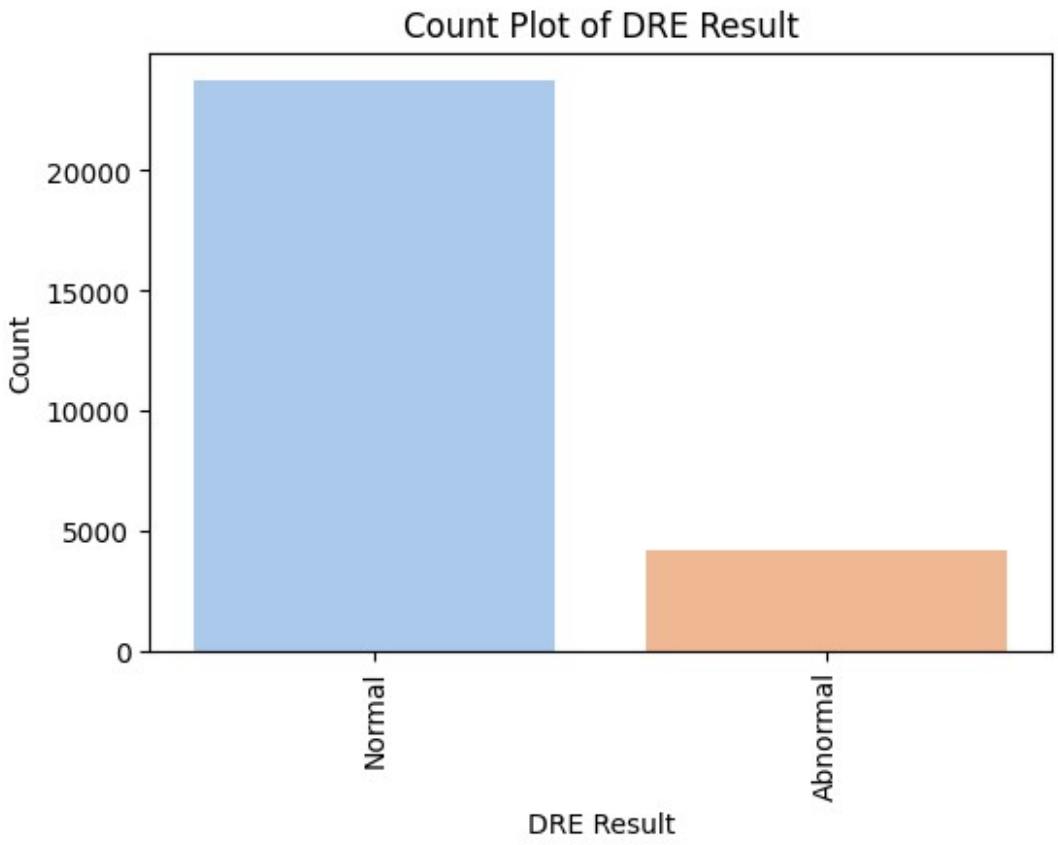
```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



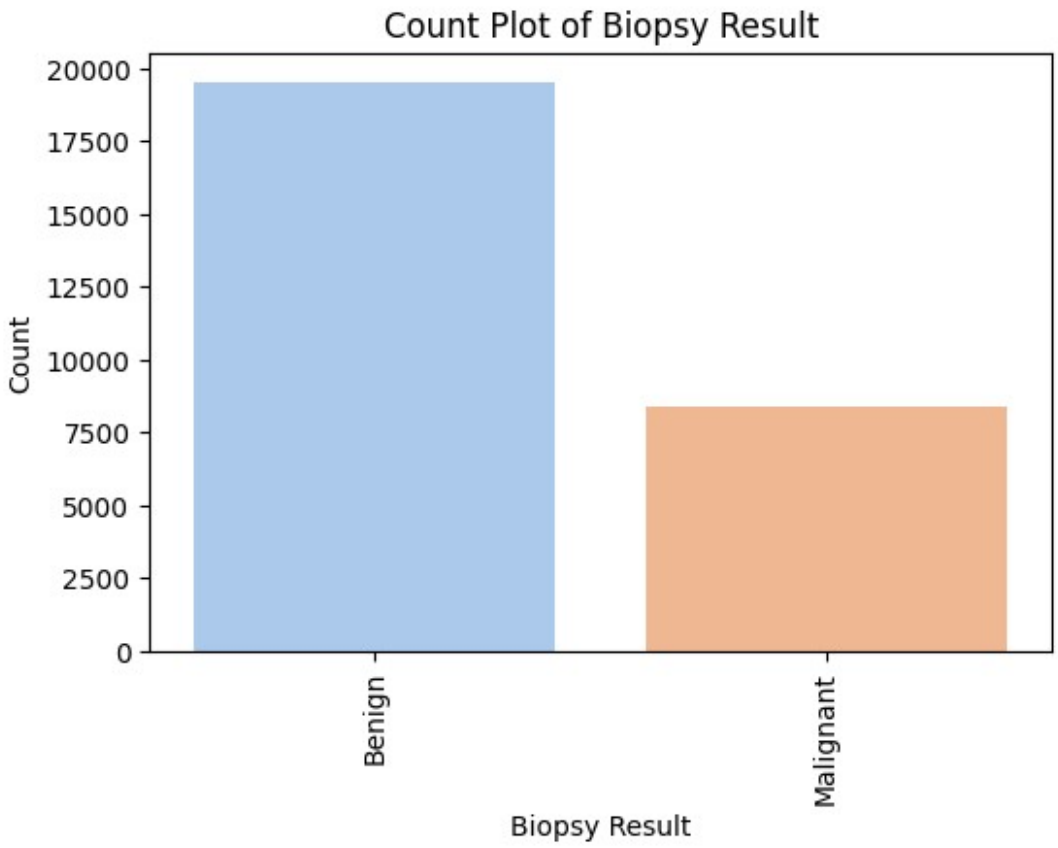

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



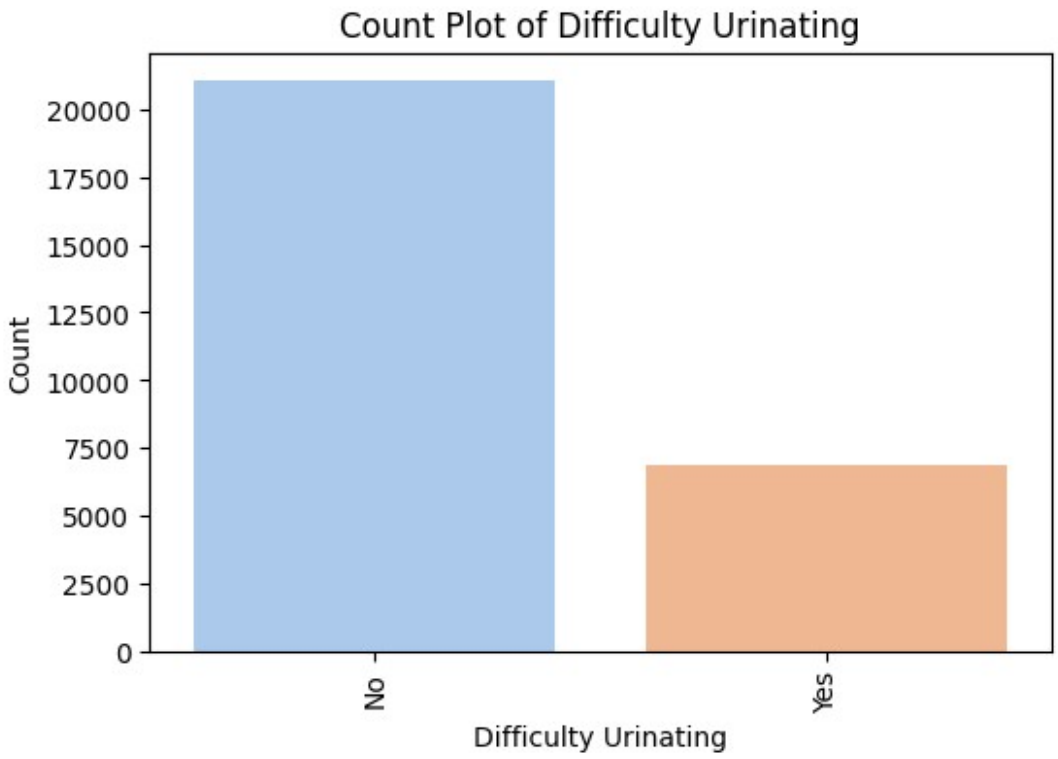

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



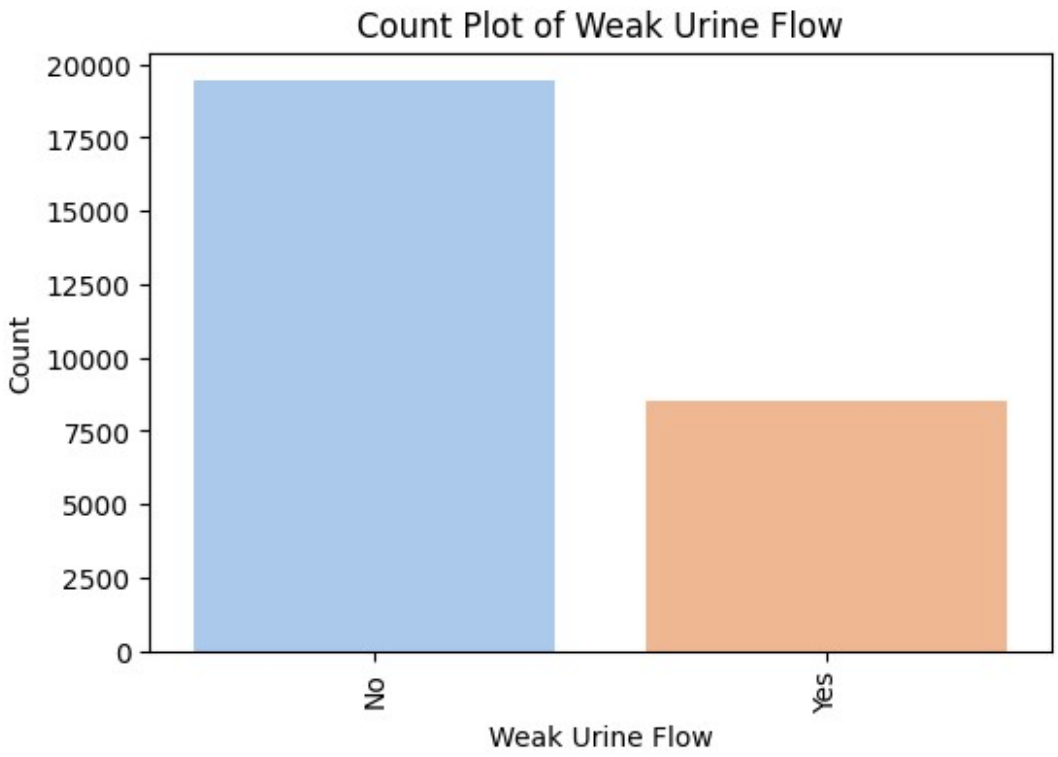

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



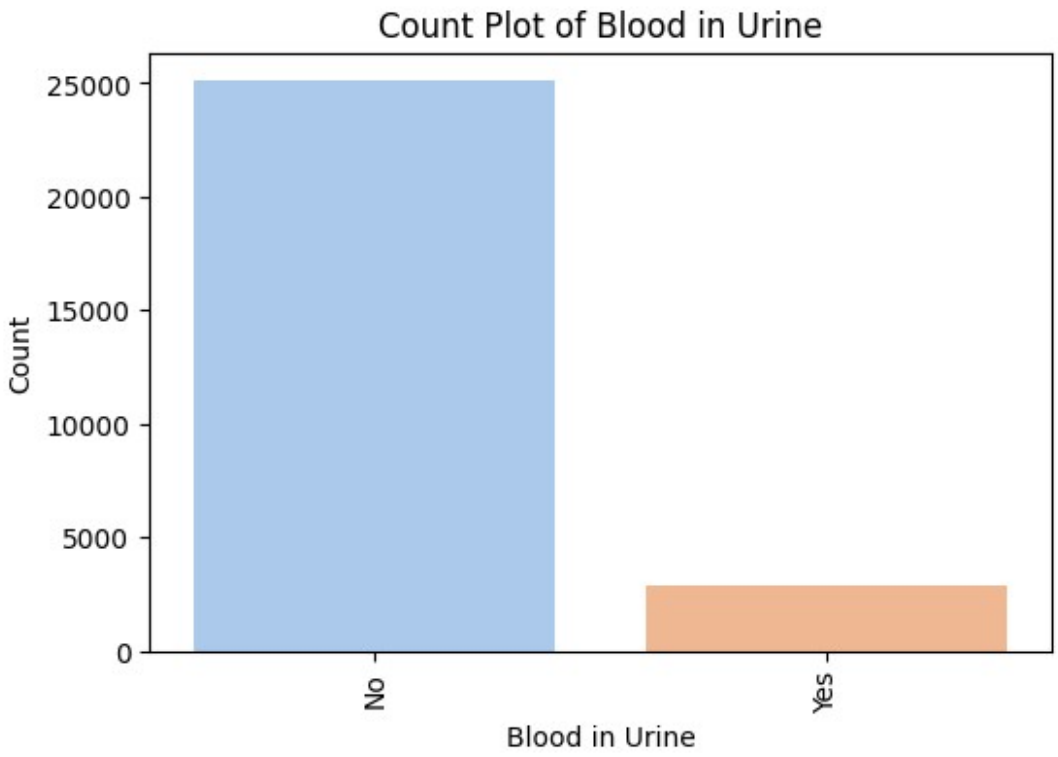

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



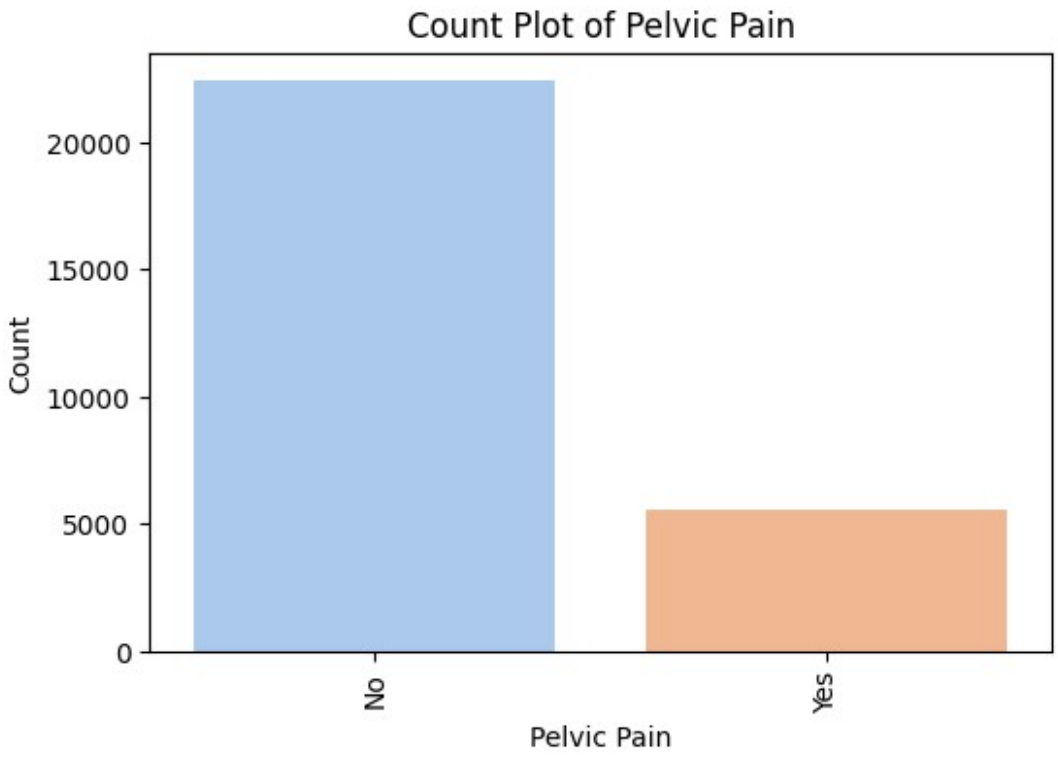

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



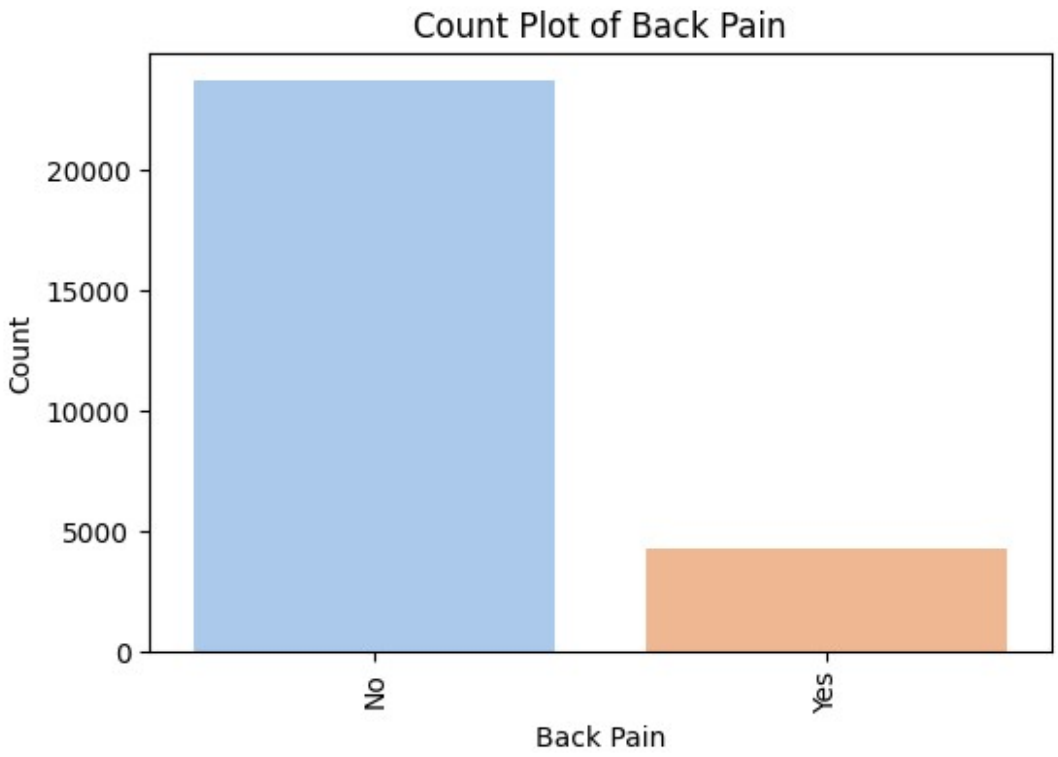

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



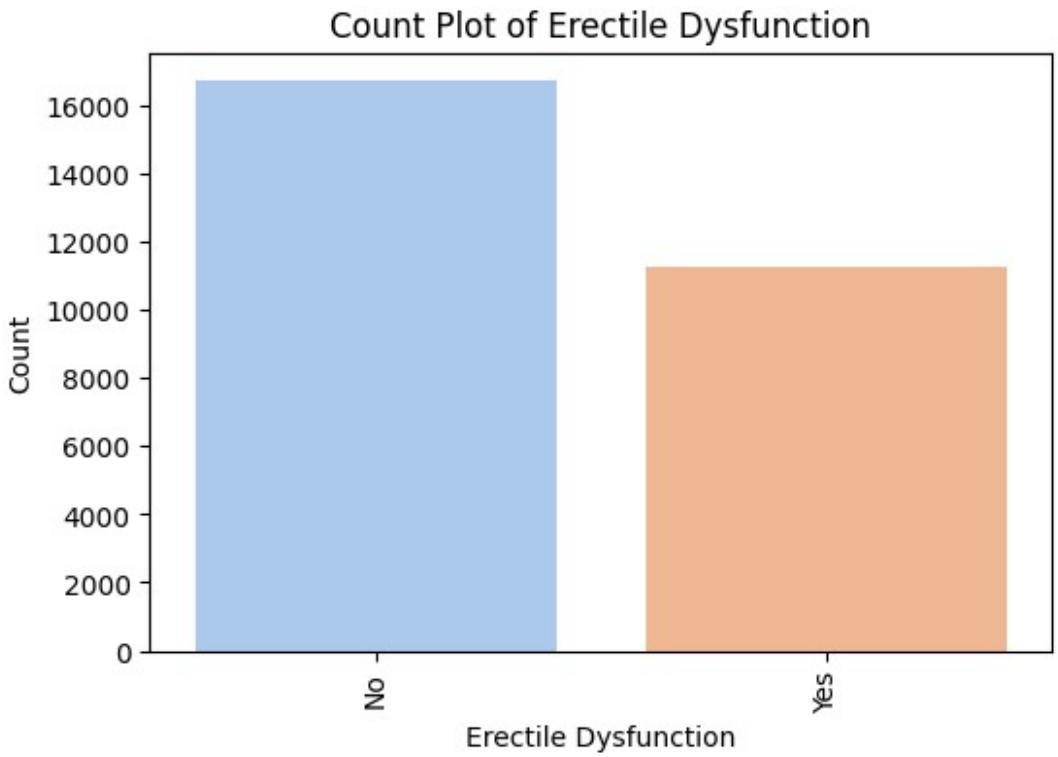

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



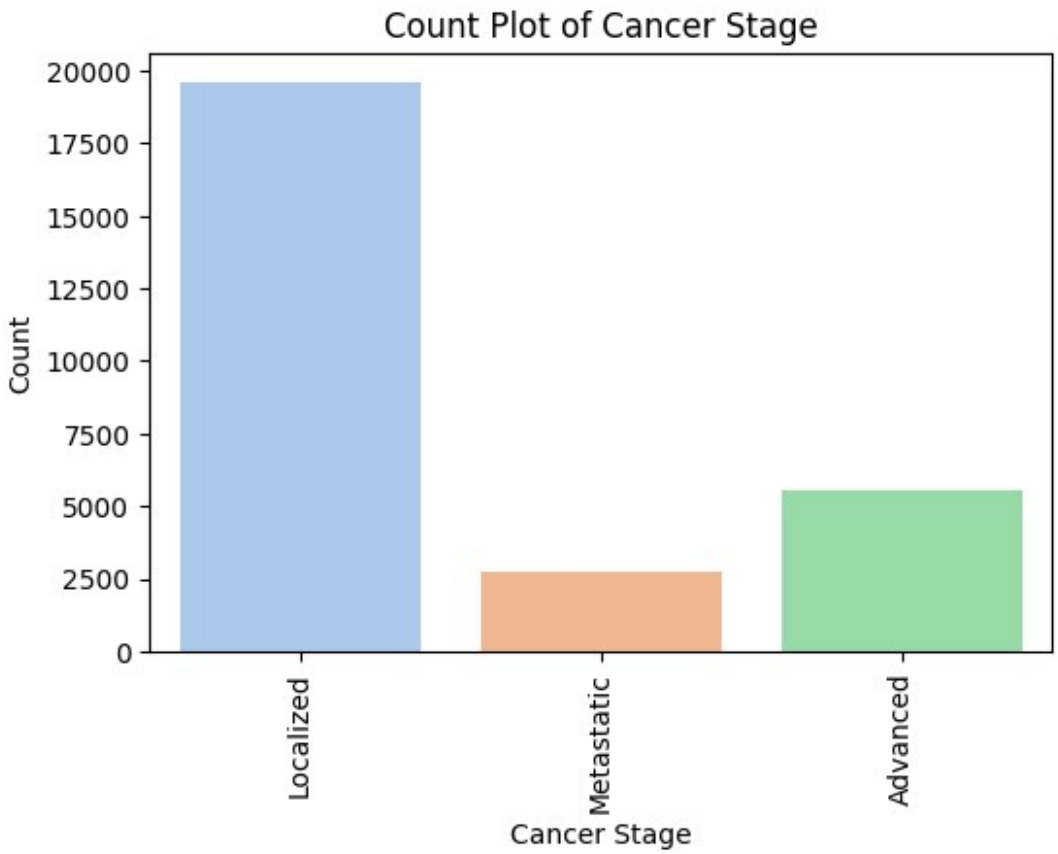

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



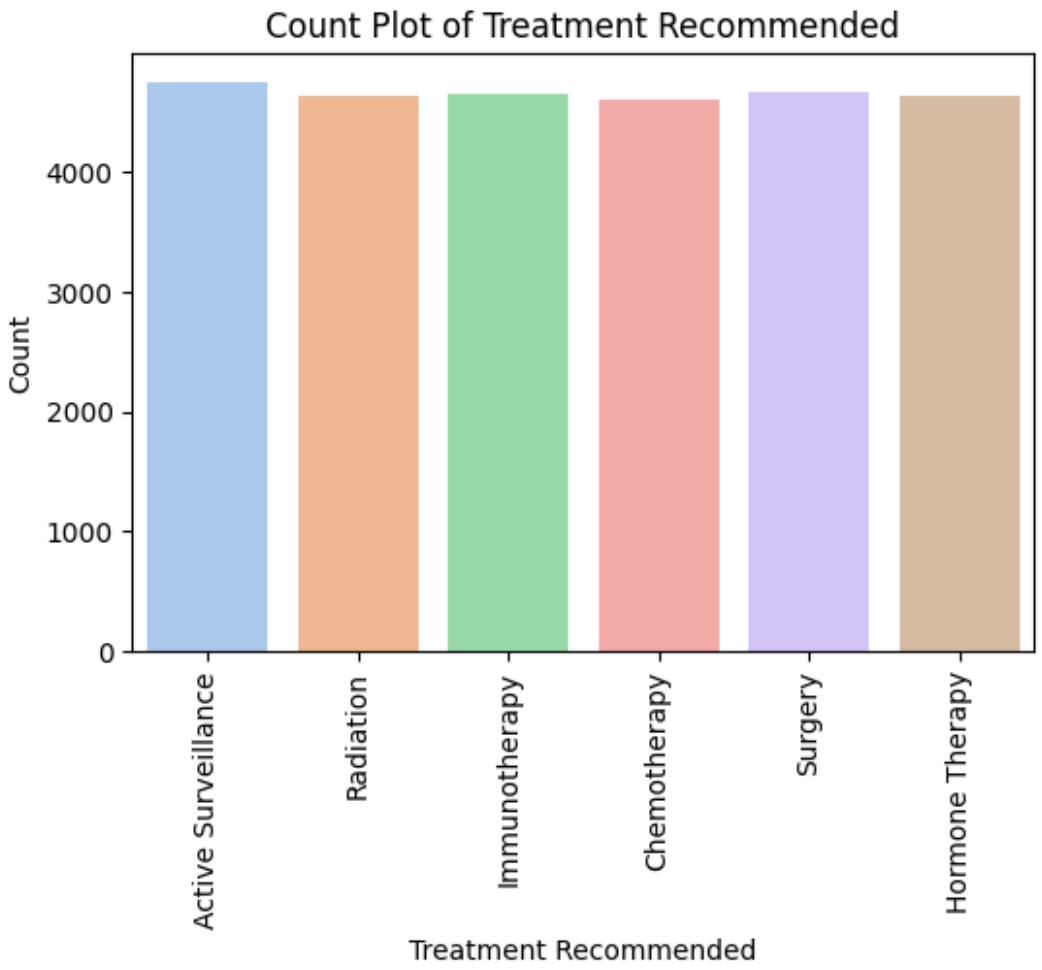

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



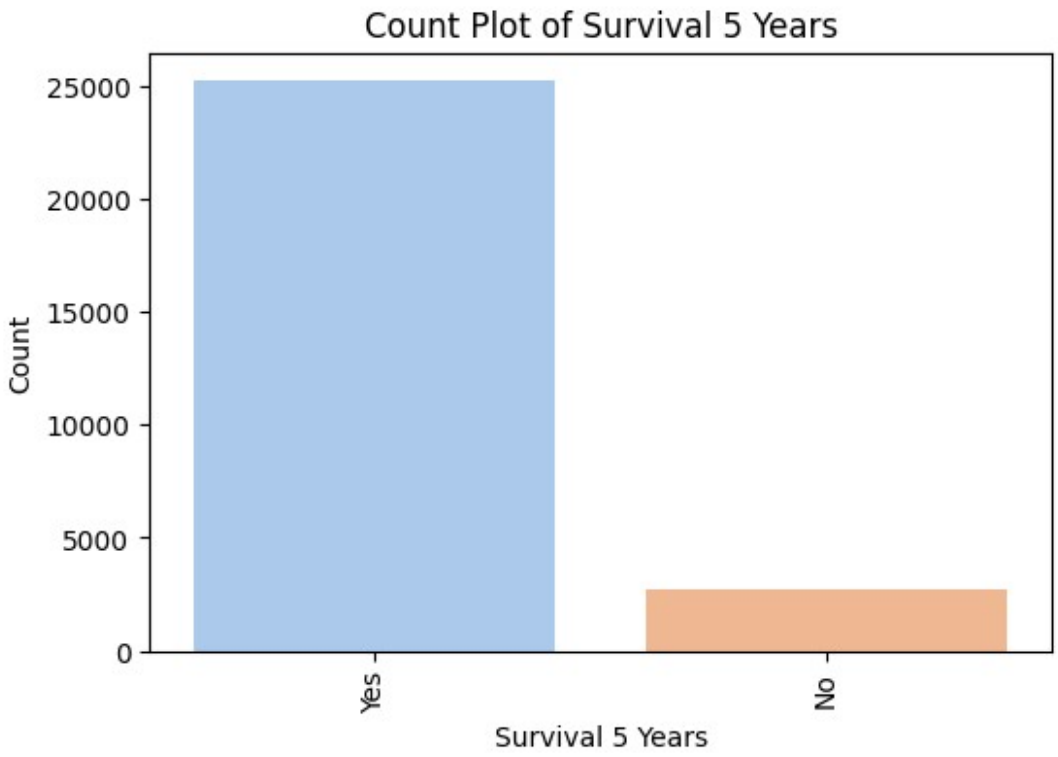

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



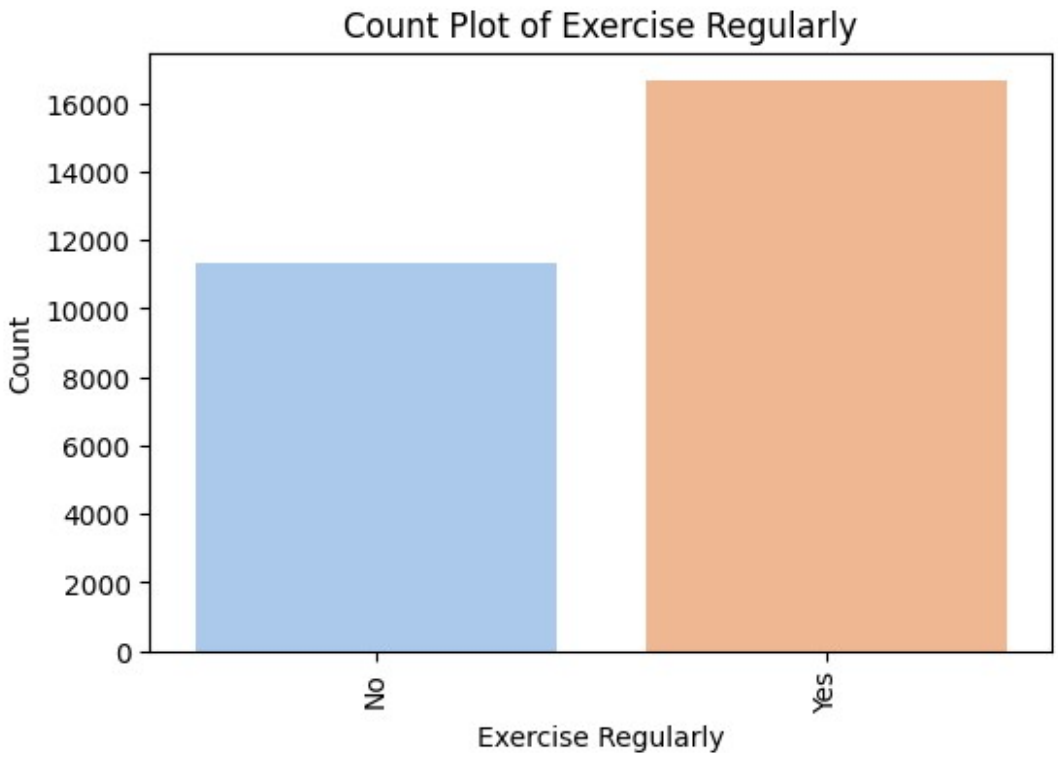

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



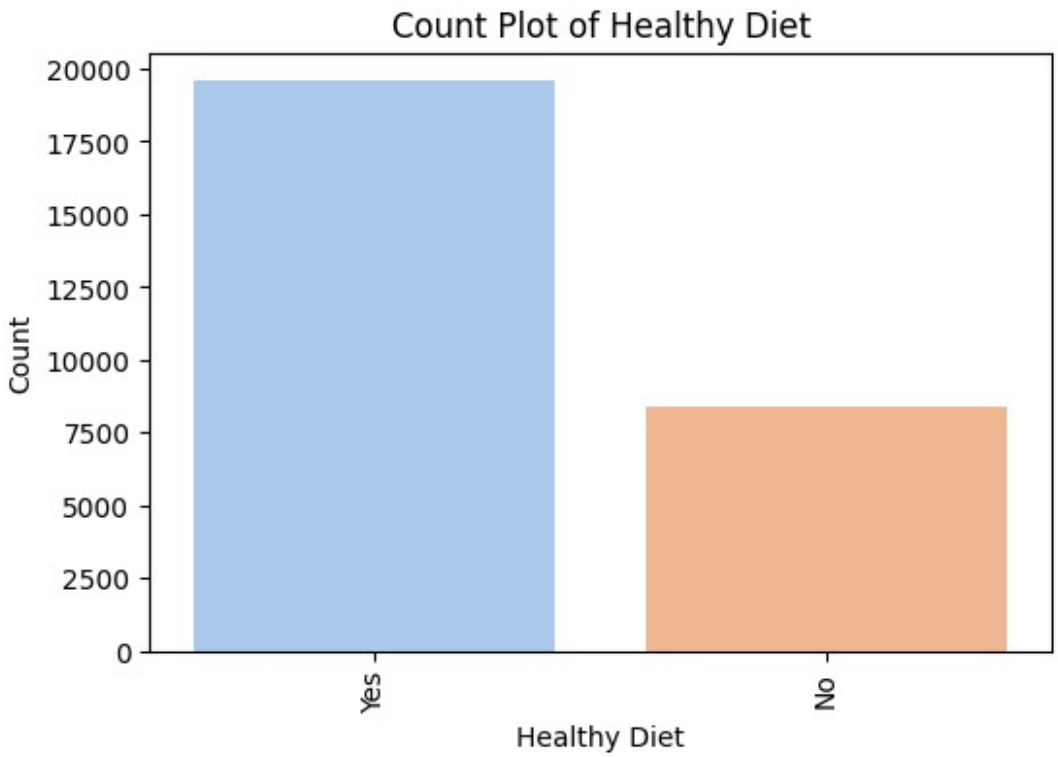

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



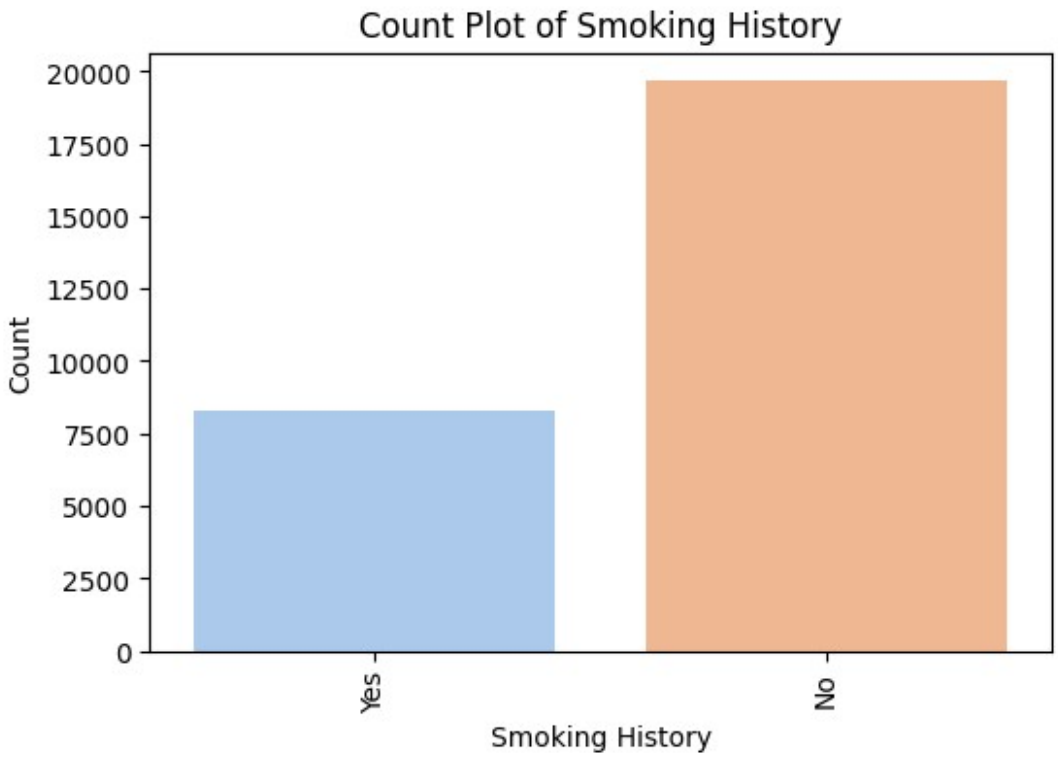

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



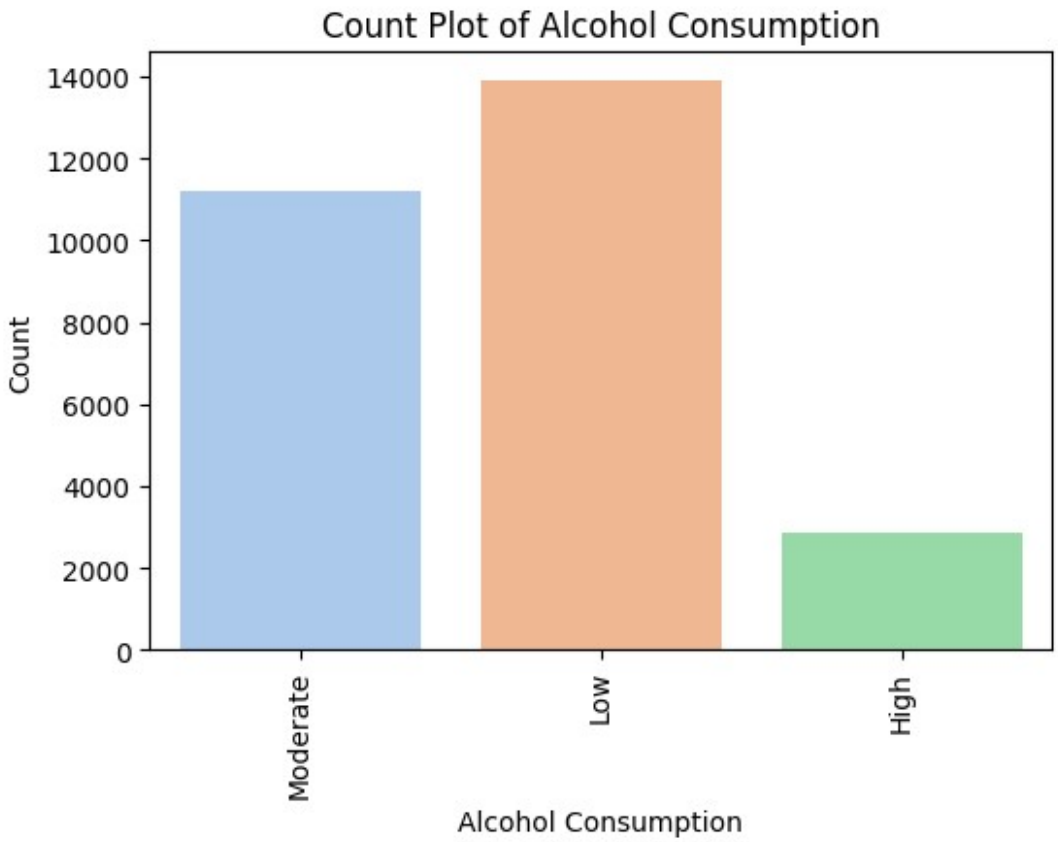

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



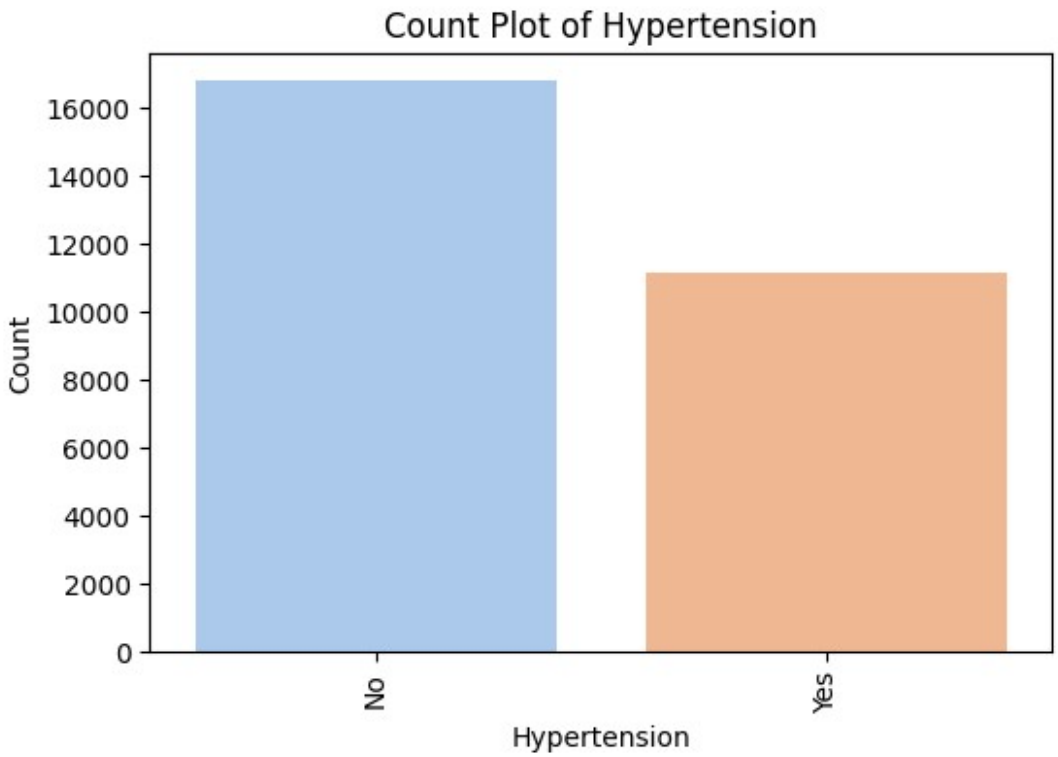

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



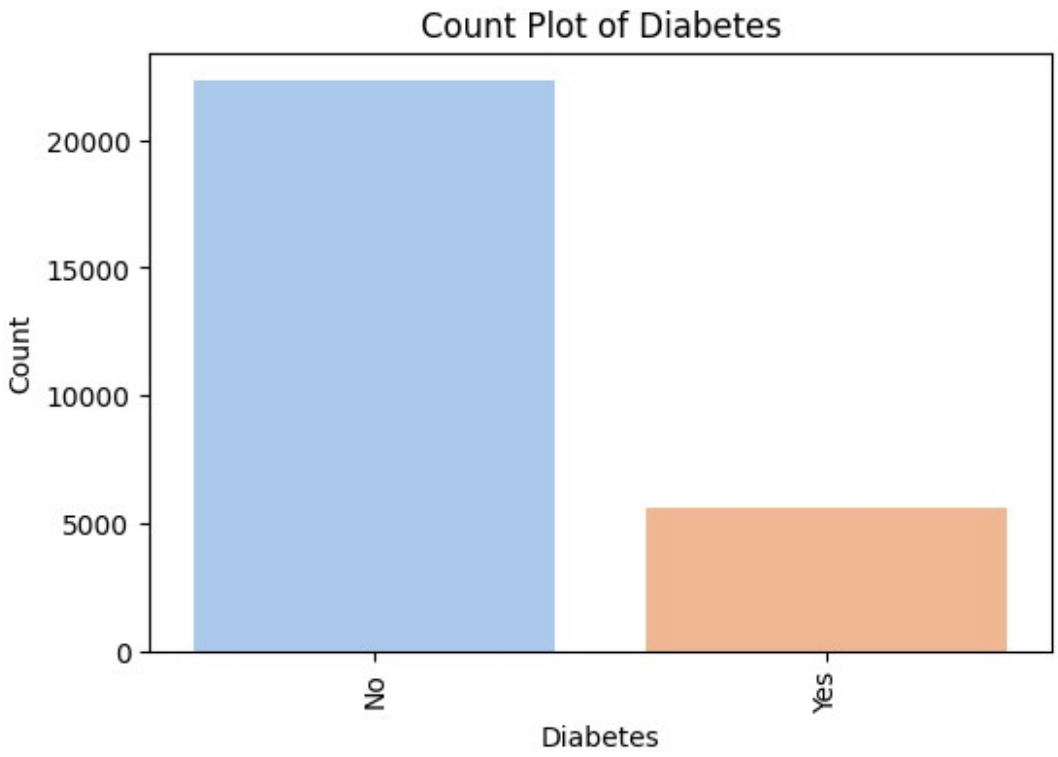

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



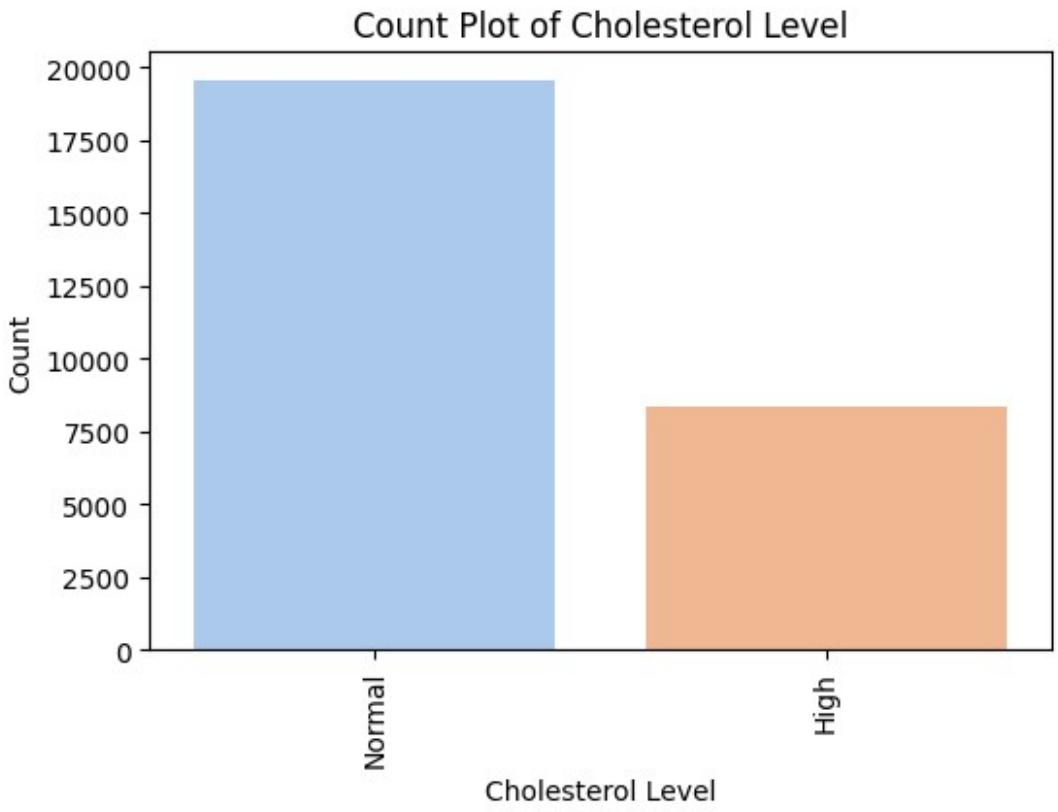

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



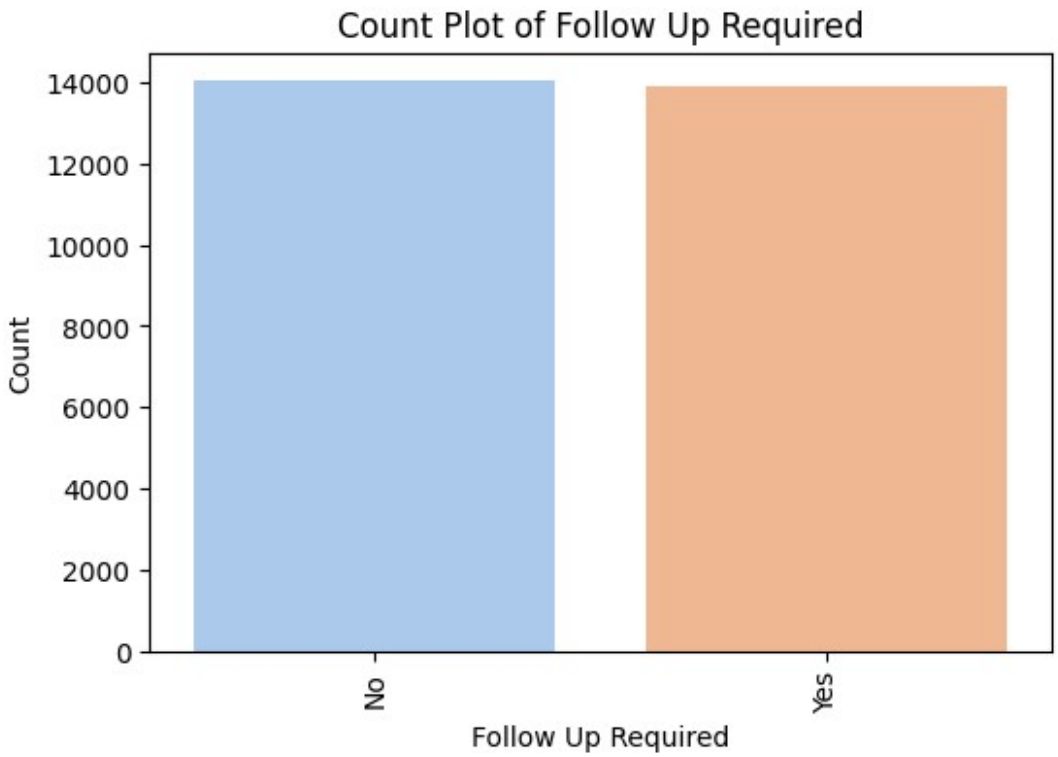

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



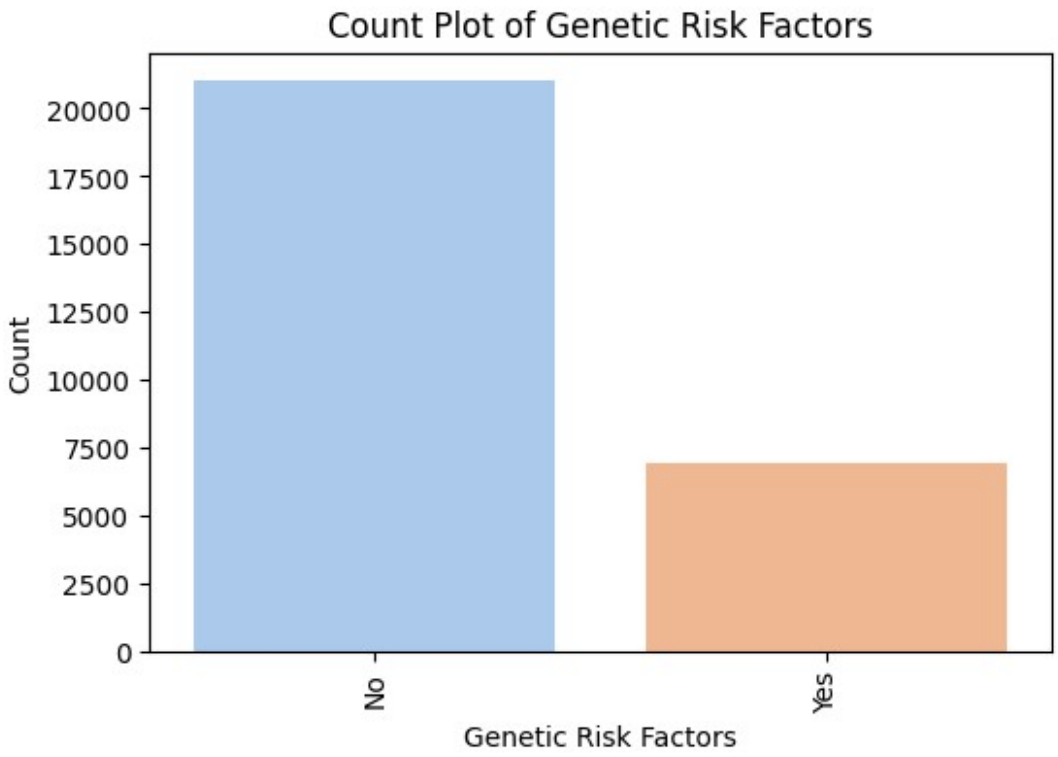

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



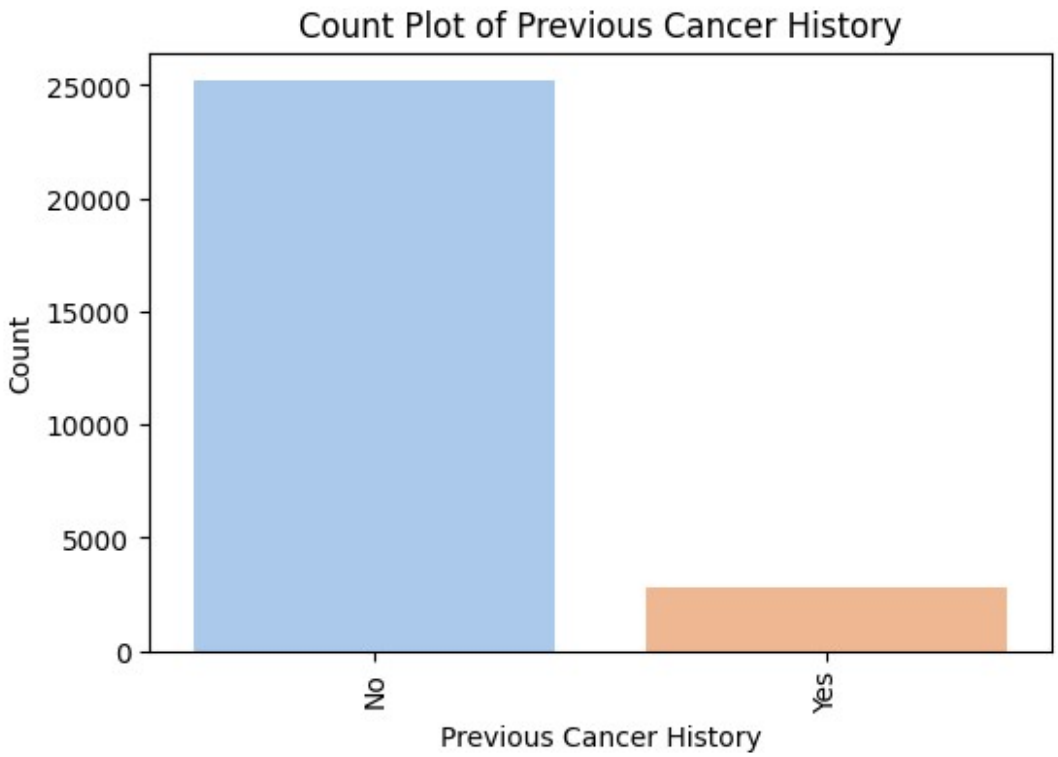

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



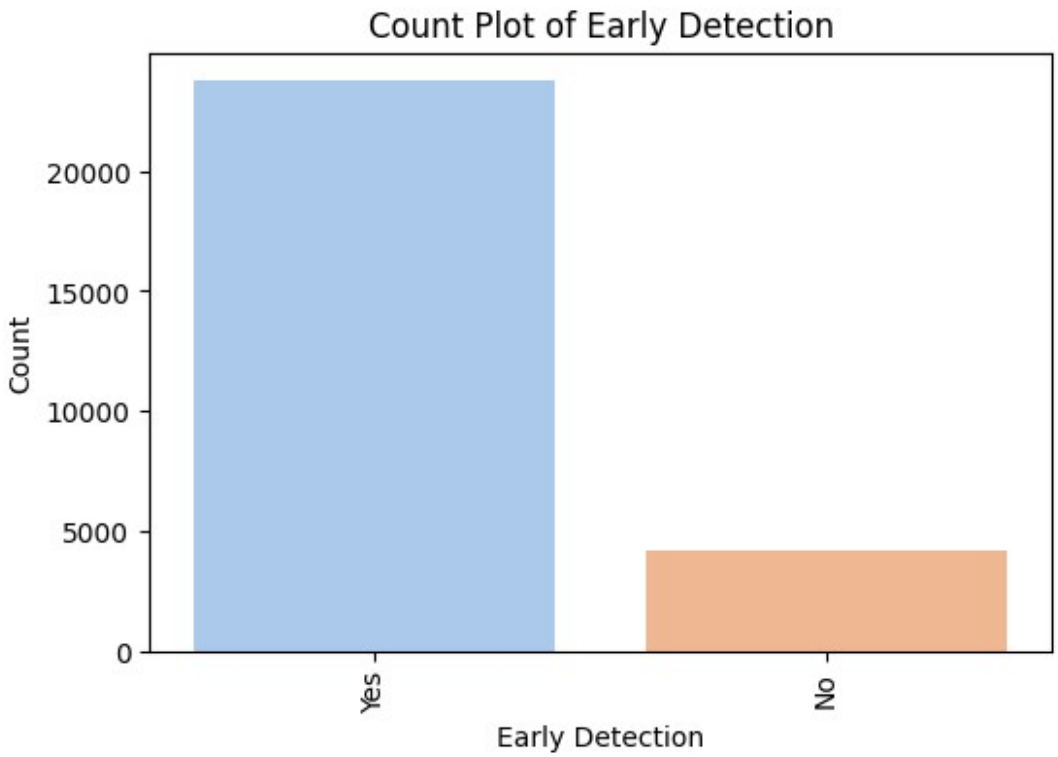

```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```




```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```




```
<ipython-input-19-68771be8f676>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=df[col], palette="pastel")
```



From above graphs the first graph is about Family history most of patient voted no which means patient family member have not been suffering from prostate cancer then next is race African ancestry where again number of patient opted for no. DRE result for maximum number of patient is normal which suggest that there are no lumps or tenderness detected in prostate or surrounding area. Next hows the distribution of biopsy results, with a clear majority classified as benign and a smaller portion classified as malignant. The difference in counts highlights a potential class imbalance in the dataset. The chart displays the frequency of individuals reporting difficulty urinating. A significantly larger number of people reported no difficulty compared to those who reported experiencing it. Visualization provides the balance ratio of the target variable "Weak Urine Flow". It shows a significant class imbalance, with a considerably larger number of individuals reporting "No" weak urine flow compared to those reporting "Yes". Then most of the patient reported "No" for blood in urine, pelvic pain, back pain and Erectile Dysfunction. The chart provides a snapshot of cancer stage distribution, it also raises important questions about potential biases and the need to address class imbalance if the data is used for further analysis or modeling. This is a count plot showing the distribution of recommended treatments. All treatment options appear to have a similar number of recommendations, suggesting a balanced distribution across the different categories. There is no one treatment that is overwhelmingly preferred over the others. Next survival for 5 years most of paitent has voted to yes means they survive more that 5 years through cancer.From Exercise regularly chart we observe that most of the individual exercised regularly. Similarly the rate for "Yes" is selected by individual for healthy diet is more than "No". Also most of the individual prefer not to smoke. Then alcohol consumption is low for maximum number of individual. But the individual with high consumption rate is less. There are less number of individual that are suffering from hypertension and diabetes. Cholesterol level is normal for most of the individual. These disease depends on genetic factor but most of the individual are free from genetic risk factor, also they didn't had any cancer history i.e maximum number of individual has "No" response to cancer

history. Last count plot suggest that early detection is possible after selecting correct parameters. Thus these columns are important for understanding data.

```
'''Here we will convert the age into groups for more visualization and analysis
Age Groups are important because we can under which age group is more likely to have diabetes,cholesterol etc.
'''
df['Age'].unique()
```

```
array([78, 68, 54, 82, 47, 60, 58, 62, 50, 63, 75, 79, 42, 61, 41, 83, 69,
       77, 72, 51, 64, 88, 66, 81, 67, 55, 86, 76, 46, 48, 57, 43, 53, 89,
       65, 59, 74, 56, 45, 73, 49, 70, 87, 84, 80, 40, 44, 52, 71, 85])
```

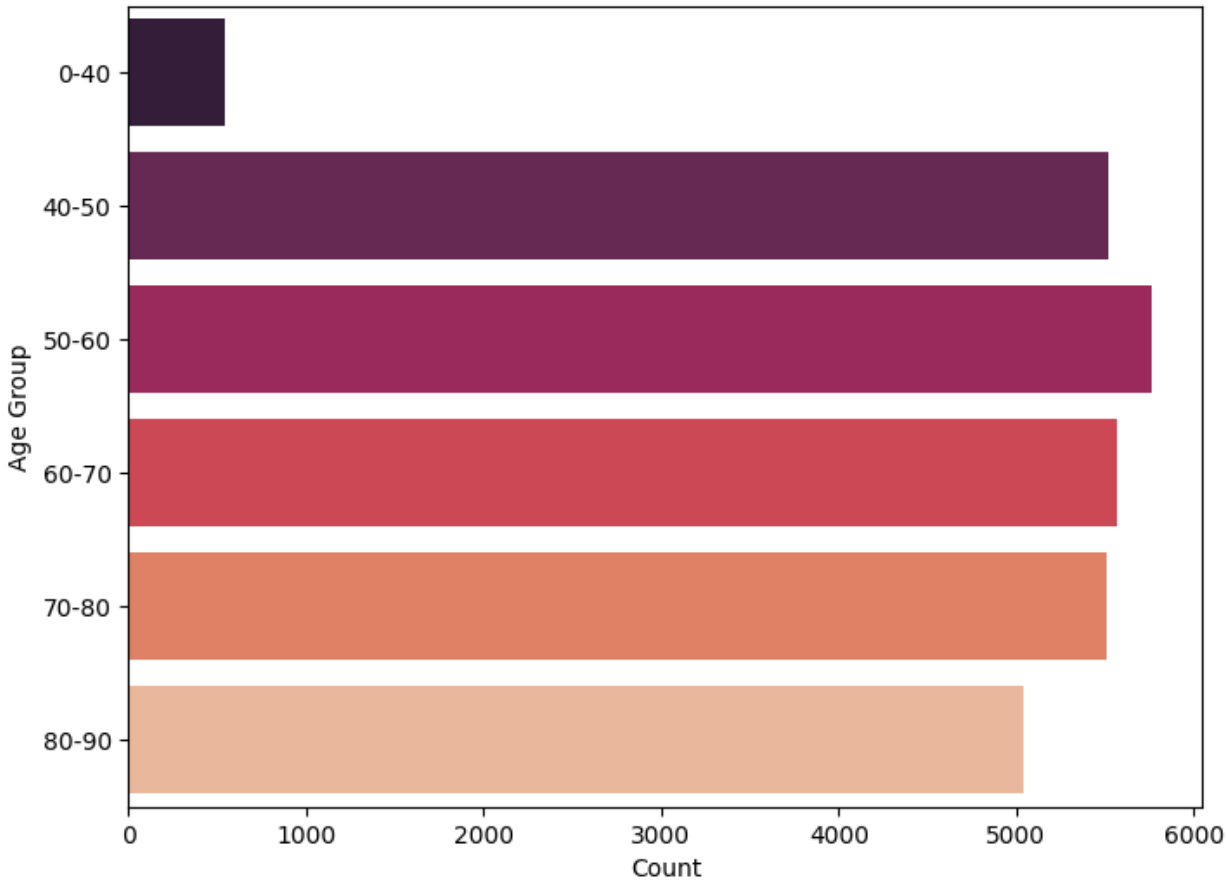
```
bins = [0, 40, 50, 60, 70, 80, 90]

labels = ['0-40', '40-50', '50-60', '60-70', '70-80', '80-90']

df['Age Group'] = pd.cut(df['Age'], bins=bins, labels = labels, right=True)

df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

From this graph we can observe that most of the people are between the age group of 50-60,60-70,70-80 means mostly old age patient from the data that are suffering or not suffering from cancer are between these age groups

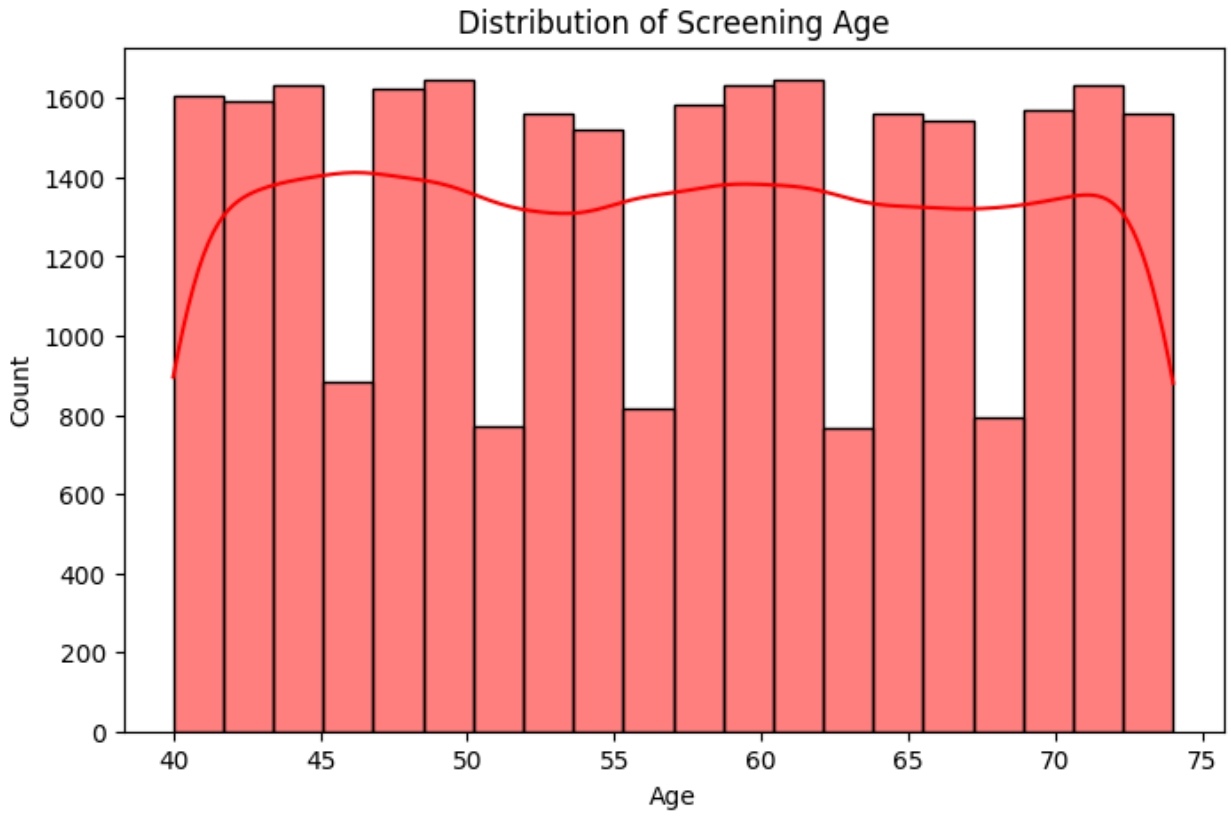
```
plt.figure(figsize=(8,6))
sns.countplot(df['Age Group'],palette='rocket')
plt.xlabel('Count')
plt.ylabel('Age Group')
plt.show()
```

```
<ipython-input-23-38a4a0f84da4>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(df['Age Group'],palette='rocket')
```



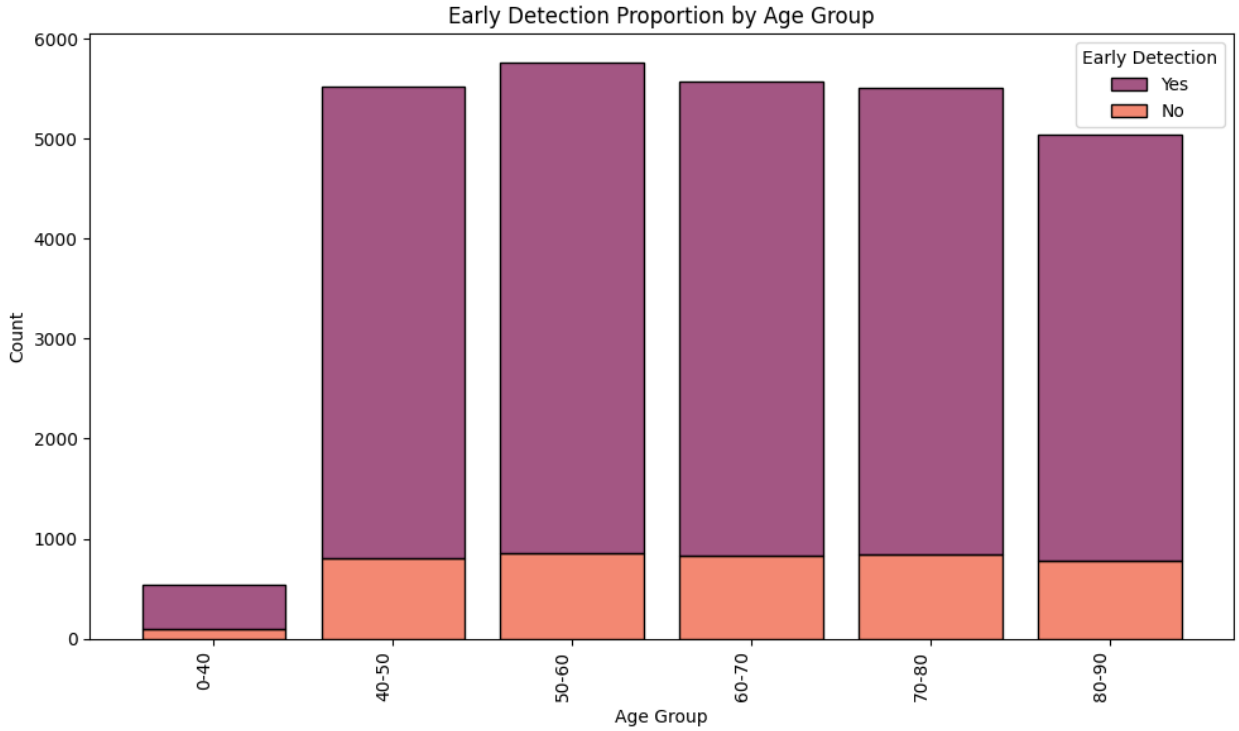
The bars suggest a fairly uniform distribution of screening ages, particularly between 45 and 70 years.There appear to be slight peaks or higher frequencies around ages 50, 55, 60, and 65. This could be due to specific screening guidelines or initiatives targeting these age groups.

```
plt.figure(figsize=(8,5))
sns.histplot(df['Screening Age'], bins=20, kde=True, color='red')
plt.title("Distribution of Screening Age")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```



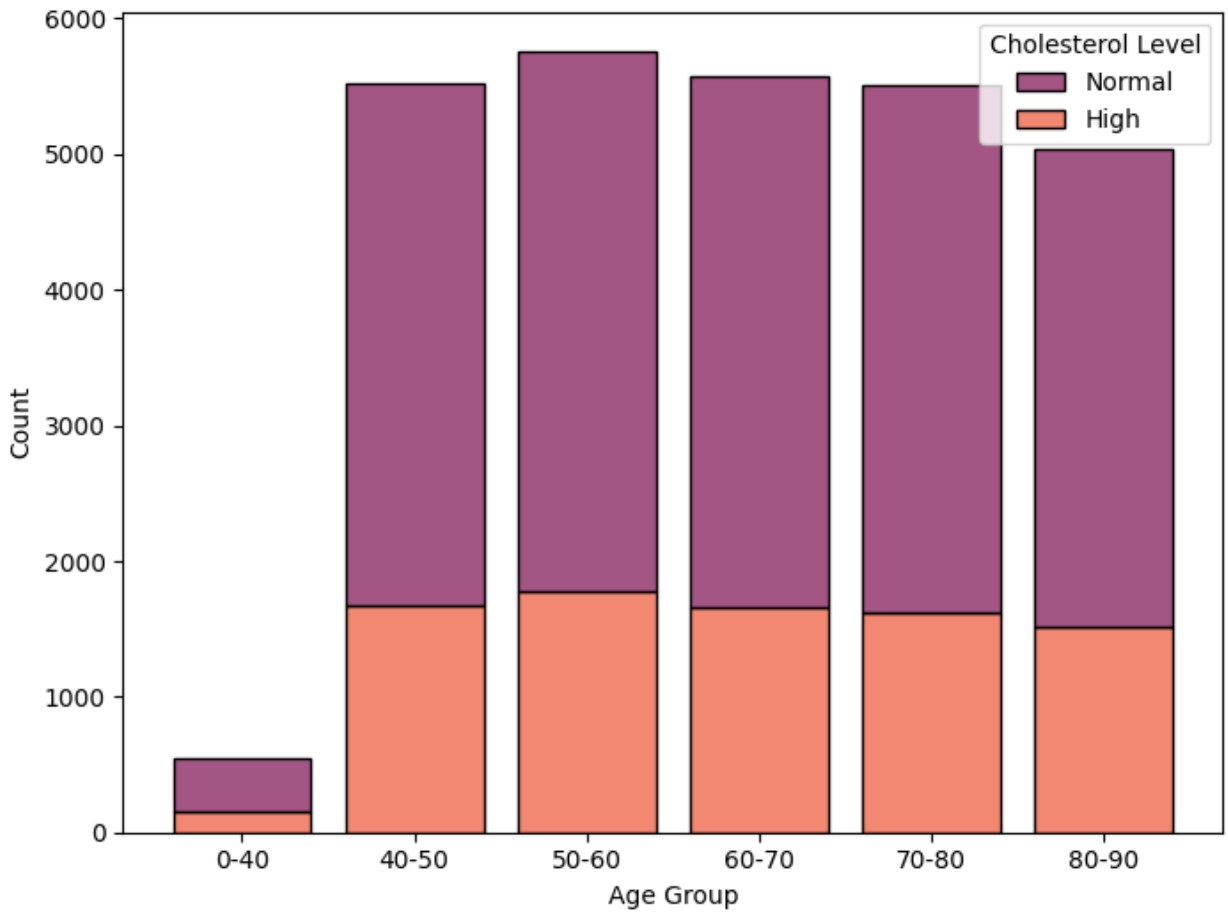
The 0-40 age group shows a significantly lower number of early detections compared to all other age groups. This is likely due to lower incidence rates of the disease in younger individuals. The 40-50, 50-60, 60-70, 70-80, and 80-90 age groups show relatively high and consistent numbers of early detections. This suggests that the screening or detection methods are effectively reaching individuals in these age ranges.The graph suggests that current techiques are effective in detecting the condition in older adults. But efforts might be needed to understand the low detection rate in the youngest group.

```
plt.figure(figsize=(10, 6))
sns.histplot(x='Age Group',hue='Early
Detection',data=df,multiple='stack',palette='rocket',shrink=0.8)
plt.title('Early Detection Proportion by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.xticks(rotation=90, ha='right')
plt.tight_layout()
plt.show()
```



The 40-50, 50-60, 60-70, 70-80, and 80-90 age groups show relatively high and consistent counts of individuals with cholesterol level 1. This suggests a higher prevalence of elevated cholesterol in these age ranges.Whereas count of individuals with cholesterol normal is relatively consistent across the 40-50 to 80-90 age groups.

```
plt.figure(figsize=(8,6))
sns.histplot(x='Age Group',hue='Cholesterol
Level',data=df,multiple='stack',palette='rocket',shrink=0.8)
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.show()
```




```
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df = df.drop('Age Group',axis=1)
```

```
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

Here we will selected the important feature that are necesary is detecting cancer.

(1) AGE is necessary as its important that model should understand what is a specific range of age that is been suffering from prostate cancer.

(2) Family history means if individual family member are suffering or was suffering from cancer.

(3) PSA level is prostate specific antigen is proten produced by both normal and cancer cells. For 60 or older age it should be at or below 4.0 mg/ml and for younger it should be at or below 2.5 mg/ml

(4) Screening test is a medical test or procedure used to detect diseases before symptoms appear. The goal is early detection to improve treatment outcomes.

(5) Prostate volume refers to the size of the prostate gland, it is an important factor in diagnosing and managing prostate-related condition.

(6) Early Detection column in your dataset likely represents whether prostate cancer was detected at early stage.

```
features = ['Age','Family History','PSA Level','Screening
Age','Prostate Volume','Early Detection']

f_df=df[features]

f_df
```


```
{"summary":"{\n  \"name\": \"f_df\",\n  \"rows\": 27945,\n
\"fields\": [\n    {\n      \"column\": \"Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
14,\n        \"min\": 40,\n        \"max\": 89,\n
\"num_unique_values\": 50,\n        \"samples\": [\n          61,\n
73,\n          57\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Family History\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Yes\",\n          \"No\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PSA Level\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.175011539020981,\n        \"min\": 0.5,\n        \"max\": 15.0,\n
\"num_unique_values\": 1451,\n        \"samples\": [\n          5.72,\
n          11.7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Screening Age\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 10,\n        \"min\": 40,\n
\"max\": 74,\n        \"num_unique_values\": 35,\n        \"samples\":
[\n          48,\n          62\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Prostate Volume\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.70428607281604,\n
\"min\": 15.0,\n        \"max\": 80.0,\n        \"num_unique_values\":
651,\n        \"samples\": [\n          17.6,\n          64.9\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Early Detection\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"f_df"}
```


Using label encoding to convert catergorical columns like Family history and Early detection into numerical columns

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

f_df['Family History'] = le.fit_transform(f_df['Family History'])
f_df['Early Detection'] = le.fit_transform(f_df['Early Detection'])

f_df
```

```
<ipython-input-33-0acade6c3dd0>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  f_df['Family History'] = le.fit_transform(f_df['Family History'])
<ipython-input-33-0acade6c3dd0>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  f_df['Early Detection'] = le.fit_transform(f_df['Early Detection'])
```


```
{"summary":"{\n  \"name\": \"f_df\",\n  \"rows\": 27945,\n
\"fields\": [\n    {\n      \"column\": \"Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
14,\n        \"min\": 40,\n        \"max\": 89,\n
\"num_unique_values\": 50,\n        \"samples\": [\n          61,\n
73,\n          57\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Family History\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PSA Level\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.175011539020981,\n
\"min\": 0.5,\n        \"max\": 15.0,\n        \"num_unique_values\":
1451,\n        \"samples\": [\n          5.72,\n          11.7\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Screening Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
10,\n        \"min\": 40,\n        \"max\": 74,\n
\"num_unique_values\": 35,\n        \"samples\": [\n          48,\n
62\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Prostate Volume\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 18.70428607281604,\n        \"min\":
15.0,\n        \"max\": 80.0,\n        \"num_unique_values\": 651,\n
\"samples\": [\n          17.6,\n          64.9\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Early Detection\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"f_df"}
```


Spliting data into x and y i.e independent features and target variable

```
x = f_df.drop('Early Detection',axis=1)
x.head()
```


```
{"summary":"{\n  \"name\": \"x\",\n  \"rows\": 27945,\n  \"fields\":
[\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 14,\n        \"min\": 40,\n
\"max\": 89,\n        \"num_unique_values\": 50,\n        \"samples\":
[\n          61,\n          73,\n          57\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Family History\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"PSA
Level\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4.175011539020981,\n        \"min\": 0.5,\n        \"max\":
15.0,\n        \"num_unique_values\": 1451,\n        \"samples\": [\n
5.72,\n          11.7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Screening Age\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 10,\n        \"min\": 40,\n
\"max\": 74,\n        \"num_unique_values\": 35,\n        \"samples\":
[\n          48,\n          62\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Prostate Volume\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.70428607281604,\n
\"min\": 15.0,\n        \"max\": 80.0,\n        \"num_unique_values\":
651,\n        \"samples\": [\n          17.6,\n          64.9\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"x"}
```


```
y = f_df['Early Detection']
y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: Early Detection, dtype: int64
```

```
x.shape,y.shape
```

```
((27945, 5), (27945,))
```

StandardScaler standardizes features by removing the mean and scaling to unit variance

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

ss_x = ss.fit_transform(x)

ss_x
```

```
array([[ 0.93998842, -0.65228398, -0.64230893, -1.17636345, -
0.09387162],
       [ 0.2457608 , -0.65228398,  0.59603316,  0.80033458,
1.62768974],
       [-0.72615787, -0.65228398,  1.44634542,  0.40499498, -
1.42514112],
       ...,
       [ 0.31518356, -0.65228398, -0.6566804 , -1.27519835, -
0.04040698],
       [-1.00384892, -0.65228398, -0.4890132 ,  0.99800439, -
1.25940074],
       [ 0.59287461, -0.65228398, -1.43992579, -1.47286816, -
0.58039983]])
```

SMOTE (Synthetic Minority Over-sampling Technique) is a technique for generating synthetic samples for the minority class instead of just duplicating existing ones.

This helps balance the dataset, preventing the model from being biased toward the majority class.

```
from imblearn.over_sampling import SMOTE

class_dist = y.value_counts(normalize=True)

smote = SMOTE(random_state=42)

x_sampled,y_sampled = smote.fit_resample(ss_x,y)

x_sampled
```

```
array([[ 0.93998842, -0.65228398, -0.64230893, -1.17636345, -
0.09387162],
       [ 0.2457608 , -0.65228398,  0.59603316,  0.80033458,
1.62768974],
       [-0.72615787, -0.65228398,  1.44634542,  0.40499498, -
1.42514112],
       ...,
       [-1.45465851, -0.65228398, -0.73214577, -0.52173825, -
1.32450358],
       [ 0.87056566, -0.65228398, -0.18872328, -0.48502237, -
0.98133018],
       [-1.49163106,  1.53307459,  0.4504838 , -1.37143818,
0.45098837]])

y_sampled

0        1
1        1
2        1
3        1
4        1
        ..
47489    0
47490    0
47491    0
47492    0
47493    0
Name: Early Detection, Length: 47494, dtype: int64
```

Here data is splitted into training and testing set

```
from sklearn.model_selection import train_test_split,learning_curve

xtrain, xtest, ytrain, ytest = train_test_split(x_sampled, y_sampled,
test_size=0.2,
                                                random_state=42,
stratify=y_sampled)

xtrain.shape,ytrain.shape,xtest.shape,ytest.shape

((37995, 5), (37995,), (9499, 5), (9499,))
```

Training Machine Learning Model

```
# Random Forest Classifier:
'''s an ensemble learning method that builds multiple decision trees
and combines their outputs to improve accuracy and reduce overfitting.
It is widely used for classification tasks due to its robustness and
ability to handle large datasets.
```

```
'''
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42,class_weight='balanced')

rf.fit(xtrain,ytrain)
```

```
RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
rf_pred = rf.predict(xtest)

rf_pred
```

```
array([0, 1, 1, ..., 1, 1, 1])
```

```
ytest
```

```
30871    0
20864    1
23678    1
34406    0
11310    1
        ..
26783    1
33980    0
1727     1
27903    1
997      1
Name: Early Detection, Length: 9499, dtype: int64
```

Strong Performance with a high accuracy of 89%. Balanced Precision & Recall ensures both risk groups are well classified. Slightly lower recall for Class 1 Some high-risk cases might be missed.

```
from sklearn.metrics import accuracy_score,classification_report

print(accuracy_score(rf_pred,ytest))

print(classification_report(rf_pred,ytest))
```

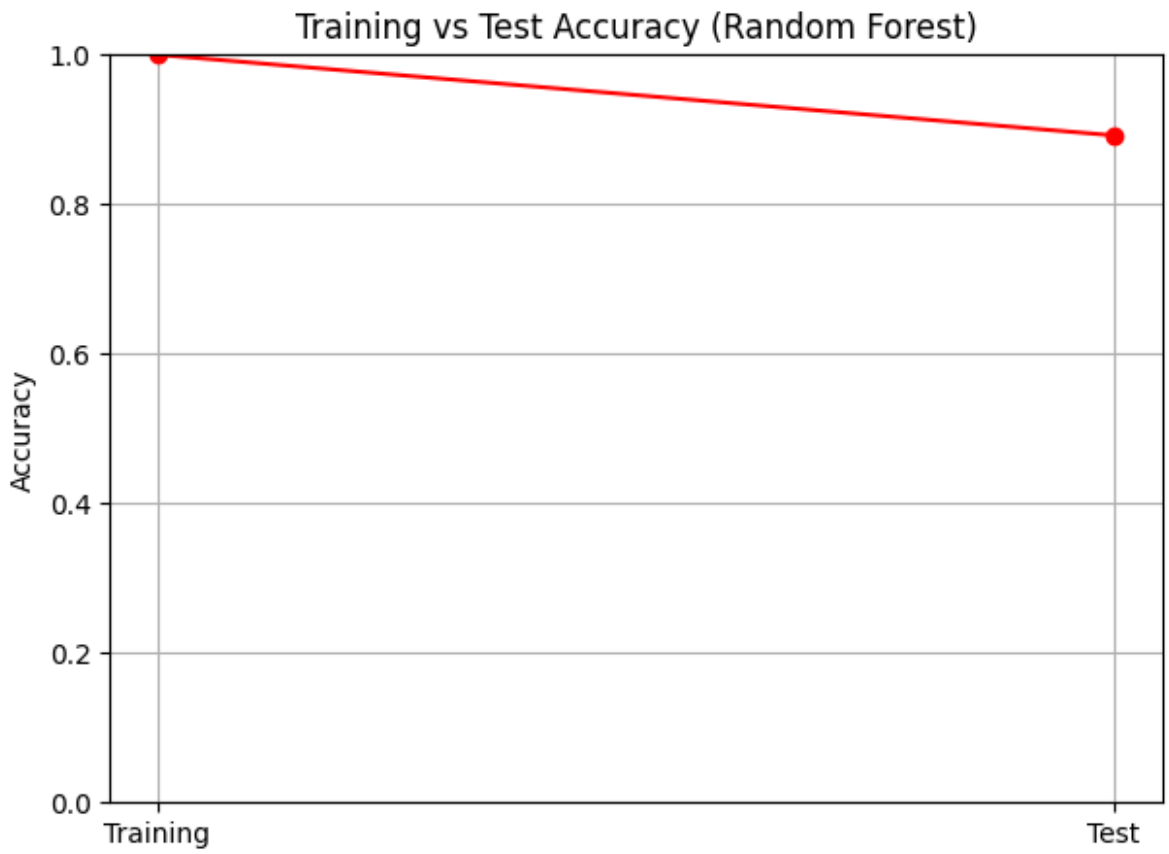
```
0.8919886303821455
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      4528
           1       0.92      0.87      0.89      4971

    accuracy                           0.89      9499
   macro avg       0.89      0.89      0.89      9499
weighted avg       0.89      0.89      0.89      9499
```

```
train_acc = accuracy_score(ytrain, rf.predict(xtrain))

test_acc = accuracy_score(ytest, rf.predict(xtest))

print(train_acc)

print(test_acc)
```

```
1.0
0.8919886303821455
```

The chart compares the training and test accuracy of a Random Forest model. The training accuracy is perfect (1.0), while the test accuracy is significantly lower (around 0.9), indicating potential overfitting.

```
plt.figure(figsize=(7, 5))
plt.plot(["Training", "Test"], [train_acc, test_acc], marker="o",
linestyle="-", color="red")
plt.ylim(0, 1)
plt.ylabel("Accuracy")
plt.title("Training vs Test Accuracy (Random Forest)")
plt.grid(True)
plt.show()
```



```
#XGBoost classifier
'''XGBoost (Extreme Gradient Boosting) is an advanced machine learning
algorithm based on decision trees.
 It is optimized for speed and performance using parallel processing,
regularization, and tree pruning techniques.
 It excels in handling structured data and is widely used for
classification and regression tasks.
'''
```

```
! pip install --upgrade xgboost
```

```
Requirement already satisfied: xgboost in
/usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in
/usr/local/lib/python3.11/dist-packages (from xgboost) (1.13.1)
```

```
import xgboost as xgb

xg =
xgb.XGBClassifier(n_estimators=500,eval_metric='logloss',reg_lambda=15
,reg_alpha=3,min_child_weight=12,

random_state=42,early_stopping_records=50,max_depth=15,tree_method='hi
st',learning_rate=0.05)

eval_set = [(xtrain,ytrain),(xtest,ytest)]

xg.fit(xtrain, ytrain,eval_set=eval_set,verbose=False)
```

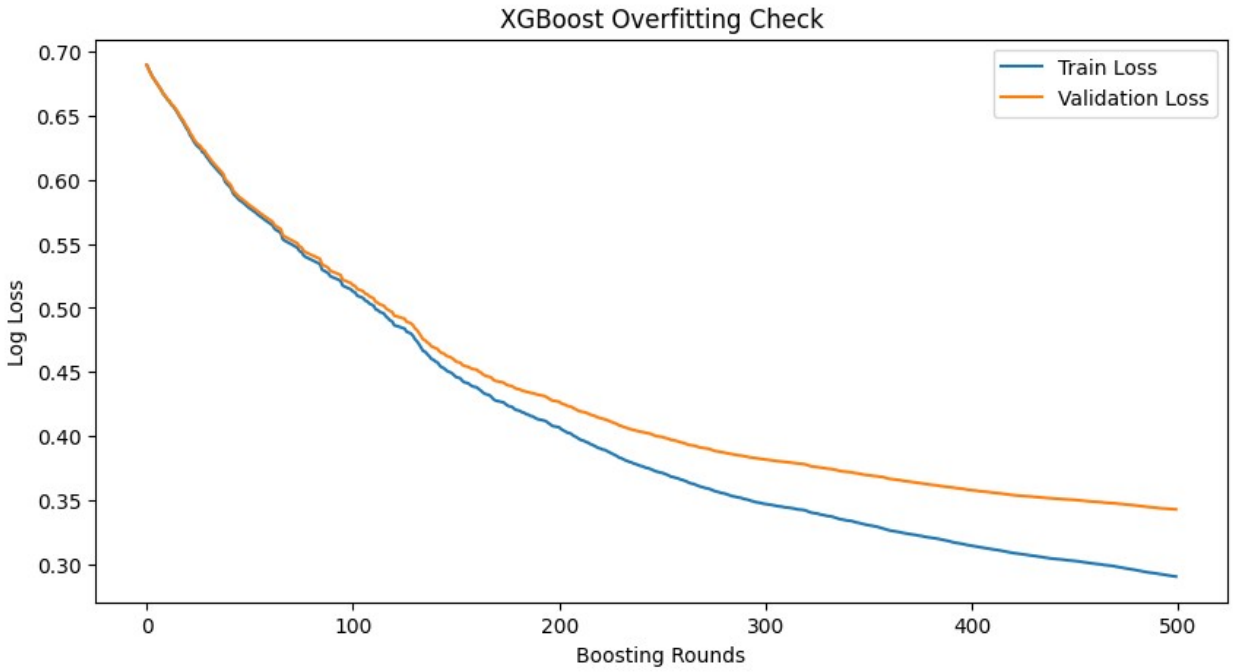
```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158:
UserWarning: [07:32:27] WARNING: /workspace/src/learner.cc:740:
Parameters: { "early_stopping_records" } are not used.

  warnings.warn(smsg, UserWarning)

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_records=50,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric='logloss', feature_types=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=15, max_leaves=None,
              min_child_weight=12, missing=nan,
```

```
monotone_constraints=None,
              multi_strategy=None, n_estimators=500, n_jobs=None,
              num_parallel_tree=None, ...)

xg_pred = xg.predict(xtest)

xg_pred

array([0, 1, 1, ..., 1, 1, 1])

ytest

30871    0
20864    1
23678    1
34406    0
11310    1
        ..
26783    1
33980    0
1727     1
27903    1
997      1
Name: Early Detection, Length: 9499, dtype: int64
```

It is a strong and balanced classifier, though it slightly favors detecting Class 0 over Class 1.

```
print(accuracy_score(xg_pred,ytest))

print(classification_report(xg_pred,ytest))

0.8687230234761554
              precision    recall  f1-score   support

           0       0.76      0.97      0.85      3731
           1       0.98      0.80      0.88      5768

    accuracy                           0.87      9499
   macro avg       0.87      0.89      0.87      9499
weighted avg       0.89      0.87      0.87      9499


results = xg.evals_result()

plt.figure(figsize=(10, 5))
plt.plot(results["validation_0"]["logloss"], label="Train Loss")
plt.plot(results["validation_1"]["logloss"], label="Validation Loss")
plt.xlabel("Boosting Rounds")
plt.ylabel("Log Loss")
plt.title("XGBoost Overfitting Check")
```

```
plt.legend()
plt.show()
```



Initial rapid decrease in both training and validation loss indicates effective learning. Validation loss remains higher than training loss, suggesting some degree of overfitting

Creating Gradio application

```
import pickle

with open("model.pkl", "wb") as f:
    pickle.dump(xg, f)

! pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.17.1-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (3.7.1)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.8-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.7.1 (from gradio)
  Downloading gradio_client-1.7.1-py3-none-any.whl.metadata (7.1 kB)
```

```
Requirement already satisfied: httpx>=0.24.1 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: jinja2<4.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (3.1.5)
Collecting markupsafe~=2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy<3.0,>=1.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (1.26.4)
Requirement already satisfied: orjson~=3.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (3.10.15)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (11.1.0)
Requirement already satisfied: pydantic>=2.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (2.10.6)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8
kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.9.7-py3-none-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata
(9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.0-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in
/usr/local/lib/python3.11/dist-packages (from gradio) (0.15.1)
Requirement already satisfied: typing-extensions~=4.0 in
/usr/local/lib/python3.11/dist-packages (from gradio) (4.12.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.0-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.11/dist-packages (from gradio-client==1.7.1-
>gradio) (2024.10.0)
```

```
Requirement already satisfied: websockets<15.0,>=10.0 in
/usr/local/lib/python3.11/dist-packages (from gradio-client==1.7.1-
>gradio) (14.2)
Requirement already satisfied: idna>=2.8 in
/usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio)
(3.10)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio)
(1.3.1)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.45.3-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: certifi in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio)
(2025.1.31)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio)
(1.0.7)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from httpcore==1.*-
>httpx>=0.24.1->gradio) (0.14.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1-
>gradio) (3.17.0)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1-
>gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in
/usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1-
>gradio) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0-
>gradio) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0-
>gradio) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0-
>gradio) (2025.1)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.0->gradio)
(0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.0->gradio)
(2.27.2)
Requirement already satisfied: click>=8.0.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12-
>gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12-
```

```
>gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in
/usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12-
>gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas<3.0,>=1.0->gradio) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0-
>typer<1.0,>=0.12->gradio) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich>=10.11.0-
>typer<1.0,>=0.12->gradio) (2.18.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface-
hub>=0.28.1->gradio) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->huggingface-
hub>=0.28.1->gradio) (2.3.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0-
>rich>=10.11.0->typer<1.0,>=0.12->gradio) (0.1.2)
Downloading gradio-5.17.1-py3-none-any.whl (62.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.3/62.3 MB 6.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.0/322.0 kB 13.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.8/94.8 kB 4.6 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (28 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.9.7-py3-none-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.6/12.6 MB 24.6 MB/s eta
0:00:00
antic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.45.3-py3-none-any.whl (71 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.5/71.5 kB 3.3 MB/s eta
0:00:00
lkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.0-py3-none-any.whl (62 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.3/62.3 kB 3.3 MB/s eta
0:00:00
py-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-
version, ruff, python-multipart, markupsafe, ffmpy, aiofiles,
starlette, safehttpx, gradio-client, fastapi, gradio
  Attempting uninstall: markupsafe
```

```
    Found existing installation: MarkupSafe 3.0.2
    Uninstalling MarkupSafe-3.0.2:
      Successfully uninstalled MarkupSafe-3.0.2
Successfully installed aiofiles-23.2.1 fastapi-0.115.8 ffmpy-0.5.0
gradio-5.17.1 gradio-client-1.7.1 markupsafe-2.1.5 pydub-0.25.1
python-multipart-0.0.20 ruff-0.9.7 safehttpx-0.1.6 semantic-version-
2.10.0 starlette-0.45.3 tomlkit-0.13.2 uvicorn-0.34.0

import gradio as gr

with open("model.pkl", "rb") as f:
    model = pickle.load(f)

def predict_cancer(Age, Family_History, PSA_Level, Screening_Age,
Prostate_Volume):
    input_data = np.array([[Age, Family_History, PSA_Level,
Screening_Age, Prostate_Volume]])
    input_scaled = ss.transform(input_data)
    prediction = xg.predict(input_scaled)
    return "Yes {The patient has Prostate cancer}" if prediction[0] ==
1 else "No {The patient doesn't have prostate cancer.}"

iface = gr.Interface(
    fn=predict_cancer,
    inputs=[
        gr.Number(label="Age"),
        gr.Number(label="Family History (0 for No, 1 for Yes)"),
        gr.Number(label="PSA Level"),
        gr.Number(label="Screening Age"),
        gr.Number(label="Prostate Volume")
    ],
    outputs=gr.Textbox(label="Early Prediction"),
    title="Prostate Cancer Prediction",
    description="Enter patient details to predict prostate cancer."
)

iface.launch()

Running Gradio in a Colab notebook requires sharing enabled.
Automatically setting `share=True` (you can turn this off by setting
`share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set
debug=True in launch()
* Running on public URL: https://c1970bf554617c7232.gradio.live

This share link expires in 72 hours. For free permanent hosting and
GPU upgrades, run `gradio deploy` from the terminal in the working
directory to deploy to Hugging Face Spaces
(https://huggingface.co/spaces)
```

```
<IPython.core.display.HTML object>
```