DSA210 TERM PROJECT

INTRODUCTION

This project will be about investigating how data can be used to determine a patient's lung cancer status based on important medical signs. The goal is to find the most crucial elements that go into a diagnosis by looking for trends in patient data. Understanding which characteristics—such as demographics, mental difficulties, or chronic diseases—are most predictive and how a data-driven strategy might support early detection are the main goals of this study.

Import necessary libraries and pull the data of the first dataset

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
```

In [2]:

```
file_path = "dataset.csv"
df = pd.read_csv(file_path)
df.head()
```

Out[2]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE	FATIGUE	ALLI
0	M	65	1	1	1	2	2	1	
1	F	55	1	2	2	1	1	2	
2	F	78	2	2	1	1	1	2	
3	M	60	2	1	1	1	2	1	
4	F	80	1	1	2	1	1	2	

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   GENDER                 3000 non-null   object
 1   AGE                    3000 non-null   int64 
 2   SMOKING                3000 non-null   int64 
 3   YELLOW_FINGERS         3000 non-null   int64 
 4   ANXIETY                3000 non-null   int64 
 5   PEER_PRESSURE          3000 non-null   int64 
 6   CHRONIC_DISEASE        3000 non-null   int64 
 7   FATIGUE                3000 non-null   int64 
 8   ALLERGY                3000 non-null   int64 
 9   WHEEZING               3000 non-null   int64 
 10  ALCOHOL_CONSUMING      3000 non-null   int64 
 11  COUGHING               3000 non-null   int64 
 12  SHORTNESS_OF_BREATH    3000 non-null   int64 
 13  SWALLOWING_DIFFICULTY  3000 non-null   int64 
 14  CHEST_PAIN             3000 non-null   int64 
 15  LUNG_CANCER            3000 non-null   object
dtypes: int64(14), object(2)
memory usage: 375.1+ KB
```

In [4]:

```
df.describe()
```

Out[4]:

	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE	FATIG
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000
mean	55.169000	1.491000	1.514000	1.494000	1.499000	1.509667	1.489
std	14.723746	0.500002	0.499887	0.500047	0.500082	0.499990	0.499
min	30.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000
25%	42.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000
50%	55.000000	1.000000	2.000000	1.000000	1.000000	2.000000	1.000
75%	68.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000
max	80.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000

Changing the number 1s and 2s into No's and Yes's, M to Male and F to Female for better visualization.

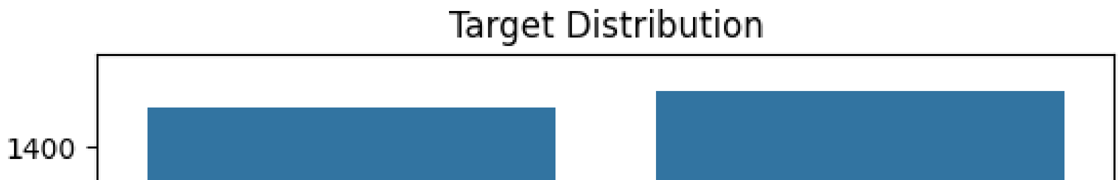
In [5]:

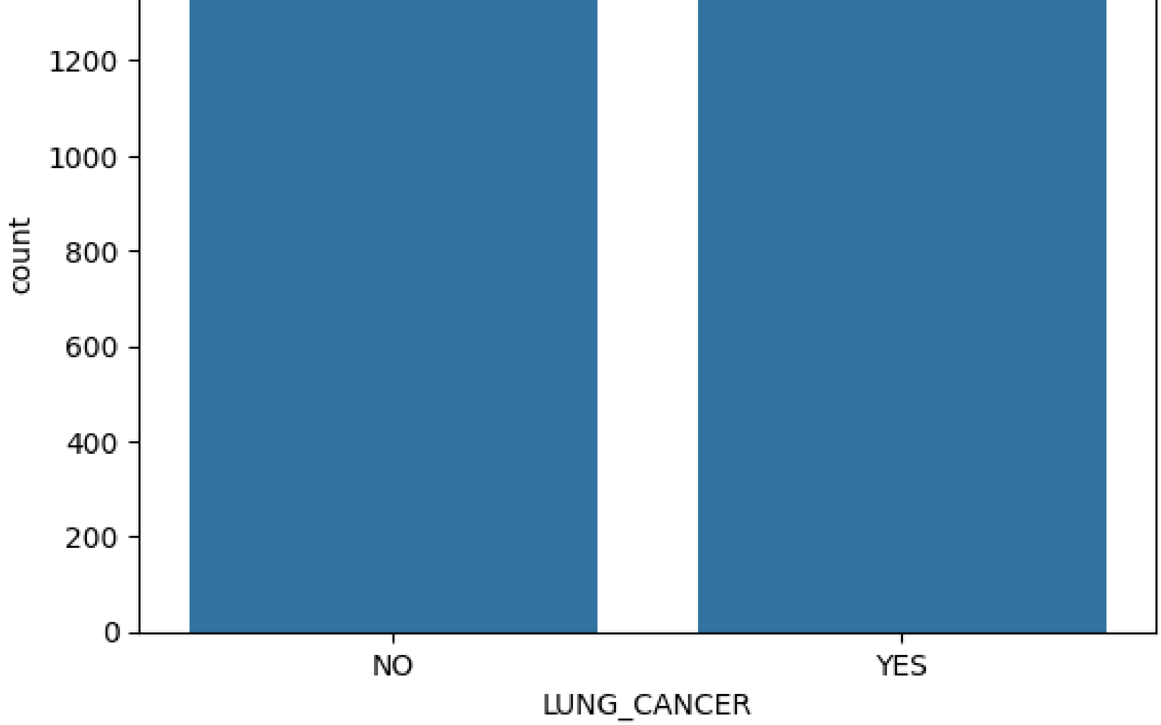
```
binary_columns = [
    "SMOKING", "YELLOW_FINGERS", "PEER_PRESSURE", "CHRONIC_DISEASE",
    "FATIGUE", "ALLERGY", "WHEEZING", "ALCOHOL_CONSUMING", "ANXIETY",
    "COUGHING", "SHORTNESS_OF_BREATH", "SWALLOWING_DIFFICULTY", "CHEST_PAIN"
]

# Map 1 -> 'No', 2 -> 'Yes'
df[binary_columns] = df[binary_columns].replace({1: "No", 2: "Yes"})
df["GENDER"] = df["GENDER"].replace({"M": "Male", "F": "Female"})
```

Total distribution of cancer patients and healthy patients.

In [6]:

```
sns.countplot(x='LUNG_CANCER', data=df,)
plt.title('Target Distribution');
```

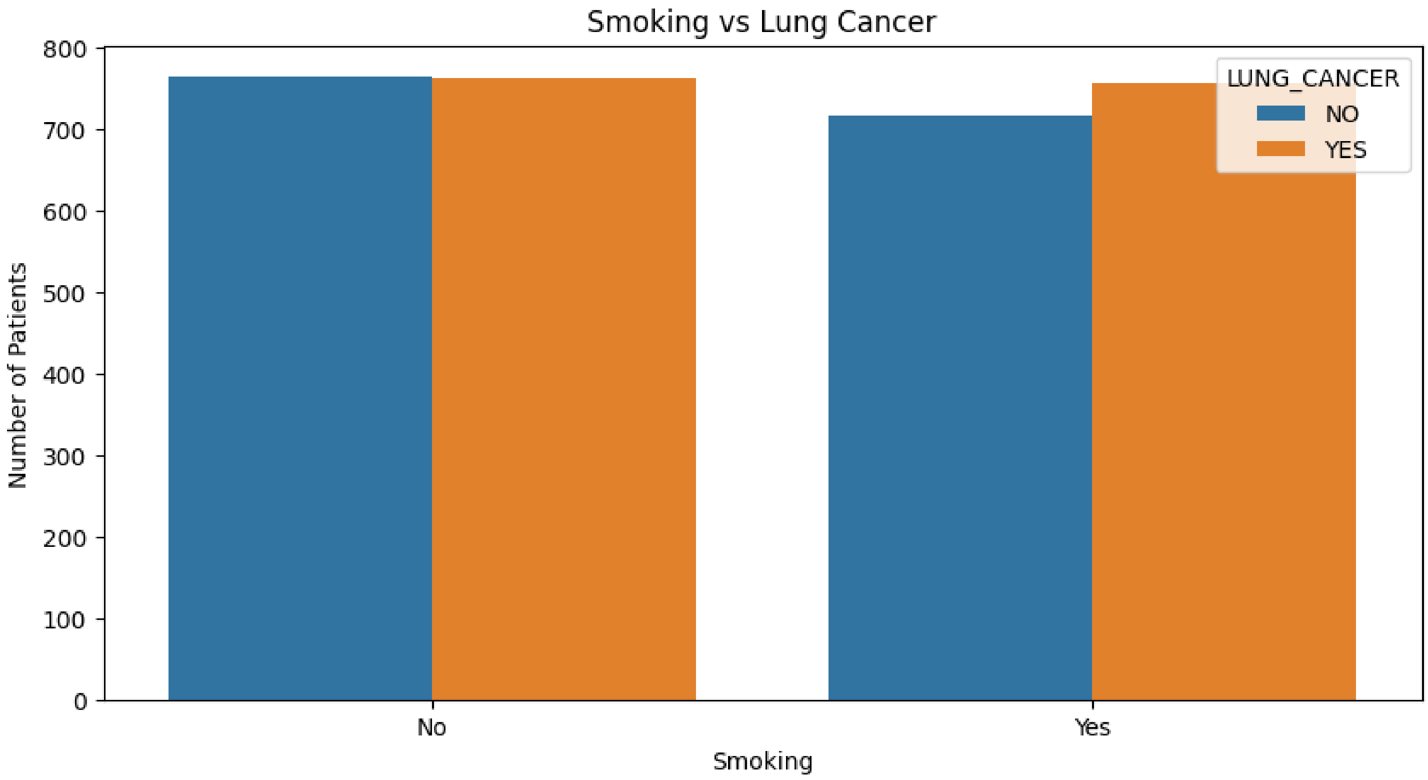


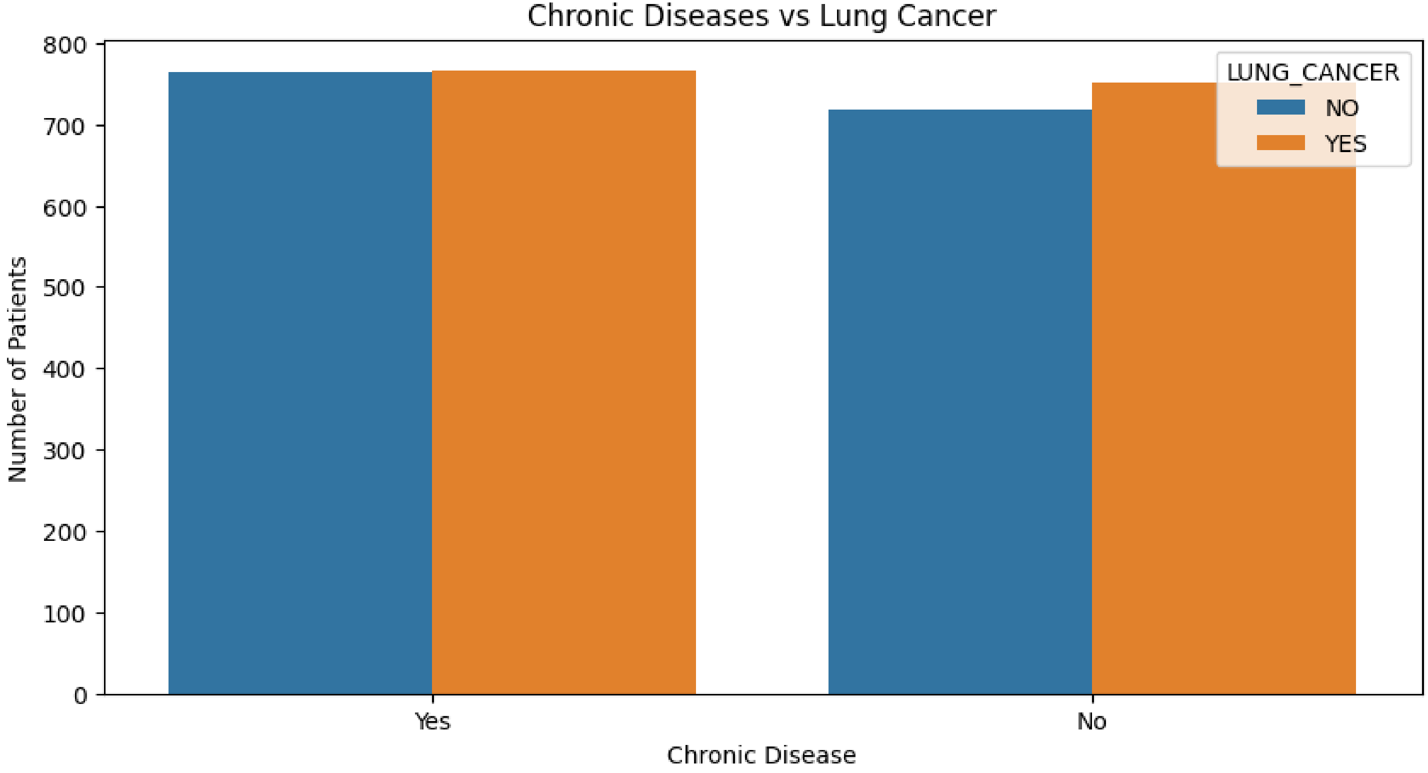


Plotting the most common assumptions for the causes of Lung Cancers.

In [7]:

```
plt.figure(figsize=(10, 5))
sns.countplot(x="SMOKING", hue="LUNG_CANCER", data=df)
plt.title("Smoking vs Lung Cancer")
plt.xlabel("Smoking")
plt.ylabel("Number of Patients")
plt.show()

plt.figure(figsize=(10, 5))
sns.countplot(x="CHRONIC_DISEASE", hue="LUNG_CANCER", data=df)
plt.title("Chronic Diseases vs Lung Cancer")
plt.xlabel("Chronic Disease")
plt.ylabel("Number of Patients")
plt.show()
```

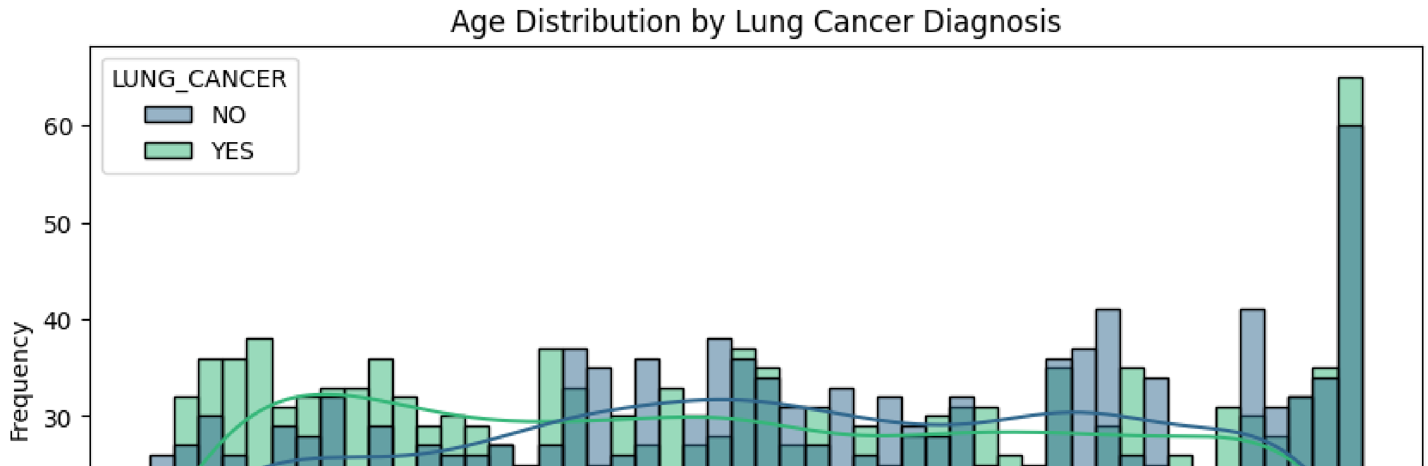


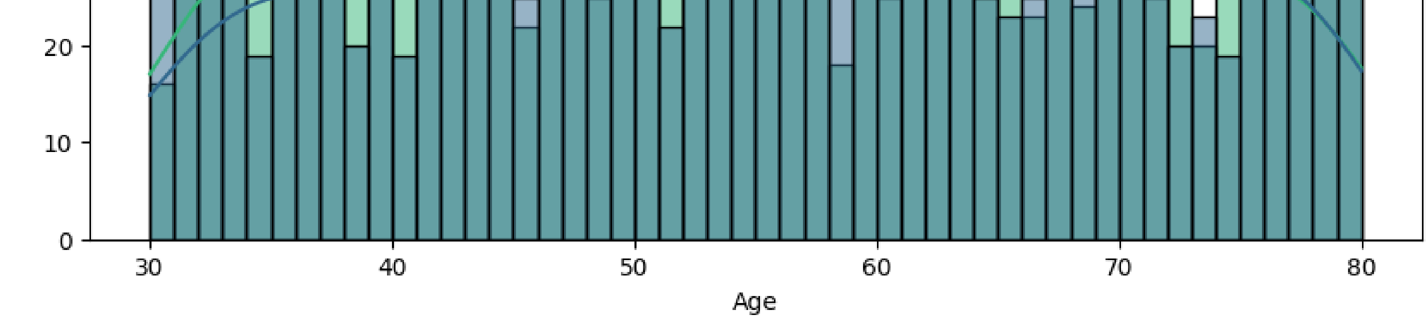


Age distributions of patients for Lung Cancer and Smoking

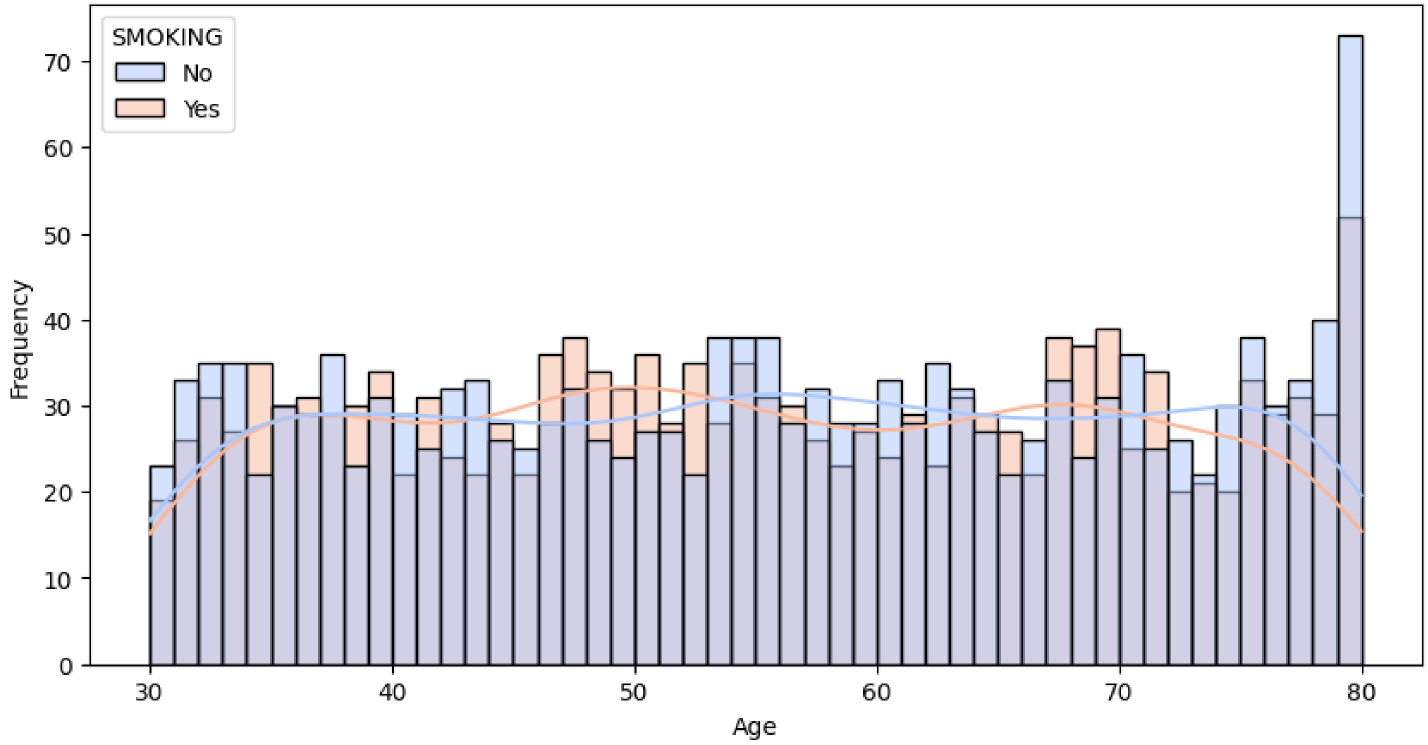
In [8]:

```
plt.figure(figsize=(10, 5))
sns.histplot(data=df, x="AGE", hue="LUNG_CANCER", kde=True, bins=50, palette="viridi
s")
plt.title("Age Distribution by Lung Cancer Diagnosis")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(10, 5))
sns.histplot(data=df, x="AGE", hue="SMOKING", kde=True, bins=50, palette="coolwarm")
plt.title("Age Distribution by Smoking")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()

mean_age = df['AGE'].mean()
median_age = df['AGE'].median()
mode_age = df['AGE'].mode()[0]

print(f"Mean Age: {mean_age}")
print(f"Median Age: {median_age}")
print(f"Mode Age: {mode_age}")
```





Age Distribution by Smoking



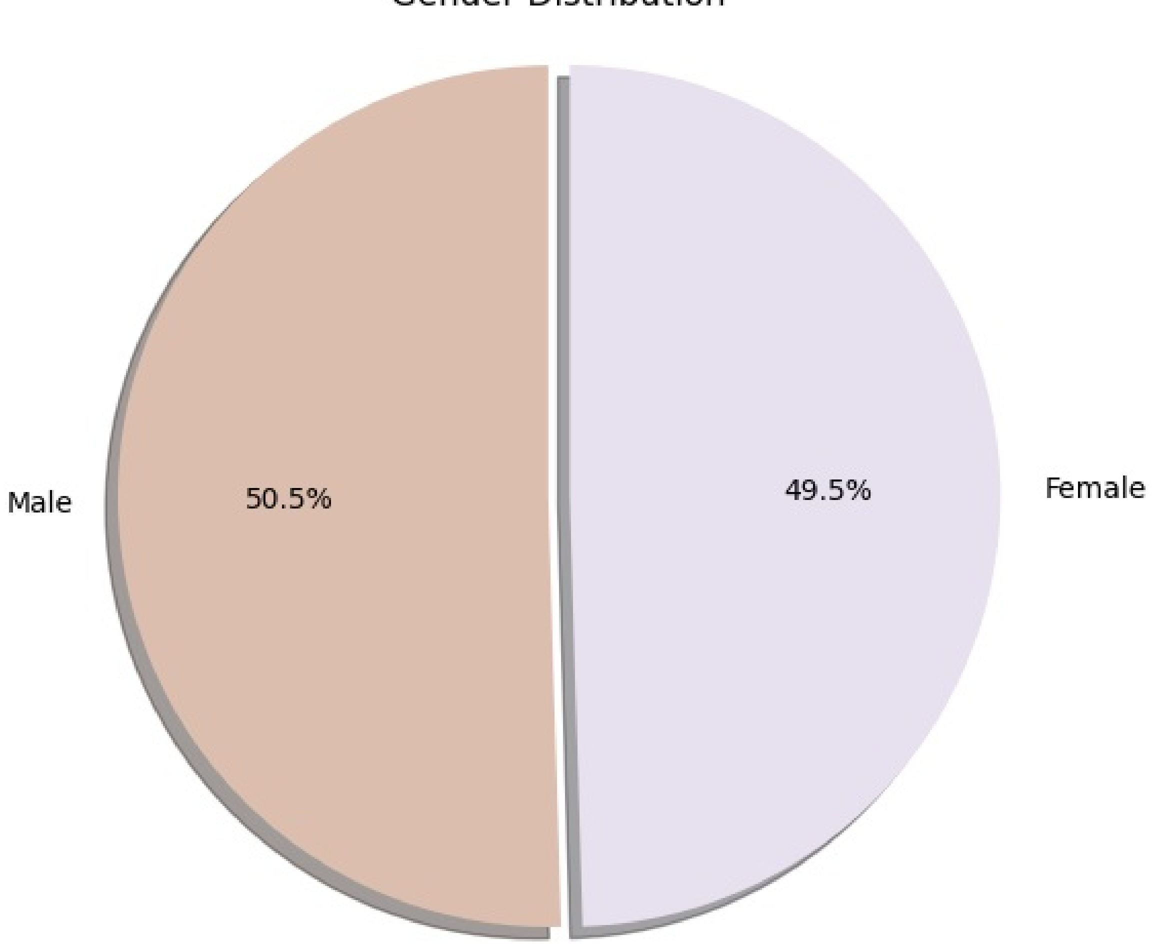
```
Mean Age: 55.169
Median Age: 55.0
Mode Age: 54
```

Male/Female Ratio

In [9]:

```
df["GENDER"] = df["GENDER"].replace({"M": "Male", "F": "Female"})

gender_counts = df["GENDER"].value_counts()

colors = ["#dcbeaf", "#e7e0ee"]
explode = (0.05, 0)

plt.figure(figsize=(6, 6))
plt.pie(
    gender_counts,
    labels=gender_counts.index,
    autopct="%1.1f%%",
    startangle=90,
    colors=colors,
    explode=explode,
    shadow=True
)
plt.title("Gender Distribution")
plt.axis("equal")
plt.show()
```

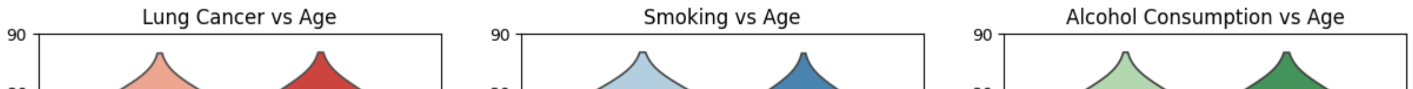
Gender Distribution

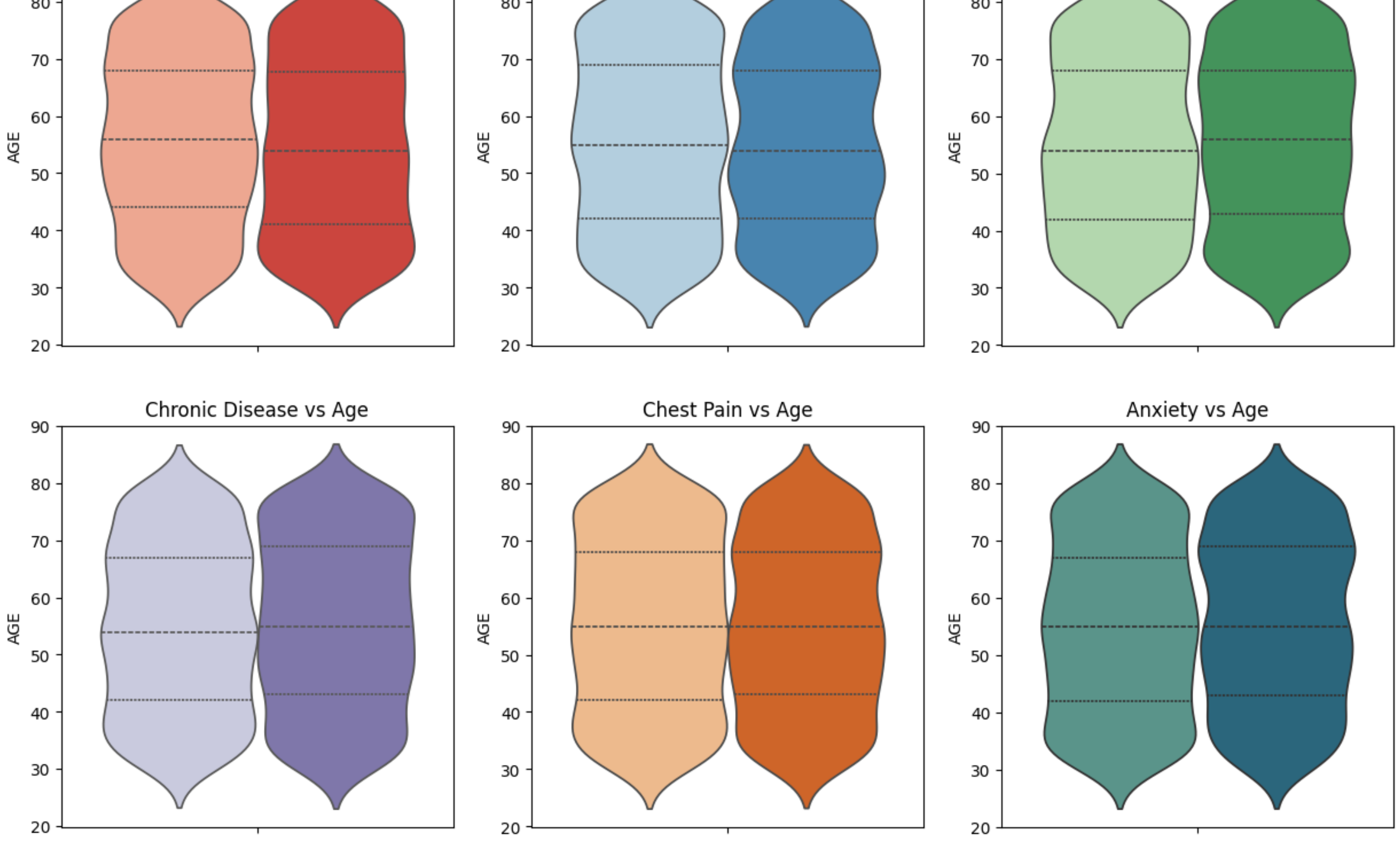


Some Columns of Possible Causes vs Age Using Violin Graphs

In [10]:

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

sns.violinplot(hue="LUNG_CANCER", y="AGE", data=df, inner="quartile", ax=axes[0][0],
palette="Reds", legend=False)
axes[0][0].set_title("Lung Cancer vs Age")

sns.violinplot(hue="SMOKING", y="AGE", data=df, inner="quartile", ax=axes[0][1], pal
ette="Blues", legend=False)
axes[0][1].set_title("Smoking vs Age")

sns.violinplot(hue="ALCOHOL_CONSUMING", y="AGE", data=df, inner="quartile", ax=axes[
0][2], palette="Greens", legend=False)
axes[0][2].set_title("Alcohol Consumption vs Age")

sns.violinplot(hue="CHRONIC_DISEASE", y="AGE", data=df, inner="quartile", ax=axes[1]
[0], palette="Purples", legend=False)
axes[1][0].set_title("Chronic Disease vs Age")

sns.violinplot(hue="CHEST_PAIN", y="AGE", data=df, inner="quartile", ax=axes[1][1],
palette="Oranges", legend=False)
axes[1][1].set_title("Chest Pain vs Age")

sns.violinplot(hue="ANXIETY", y="AGE", data=df, inner="quartile", ax=axes[1][2], pal
ette="crest", legend=False)
axes[1][2].set_title("Anxiety vs Age")

plt.show()
```

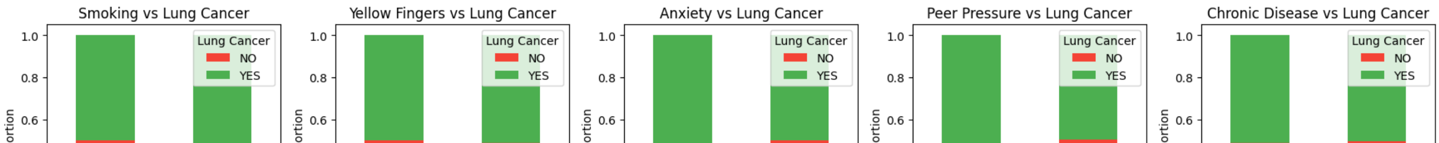


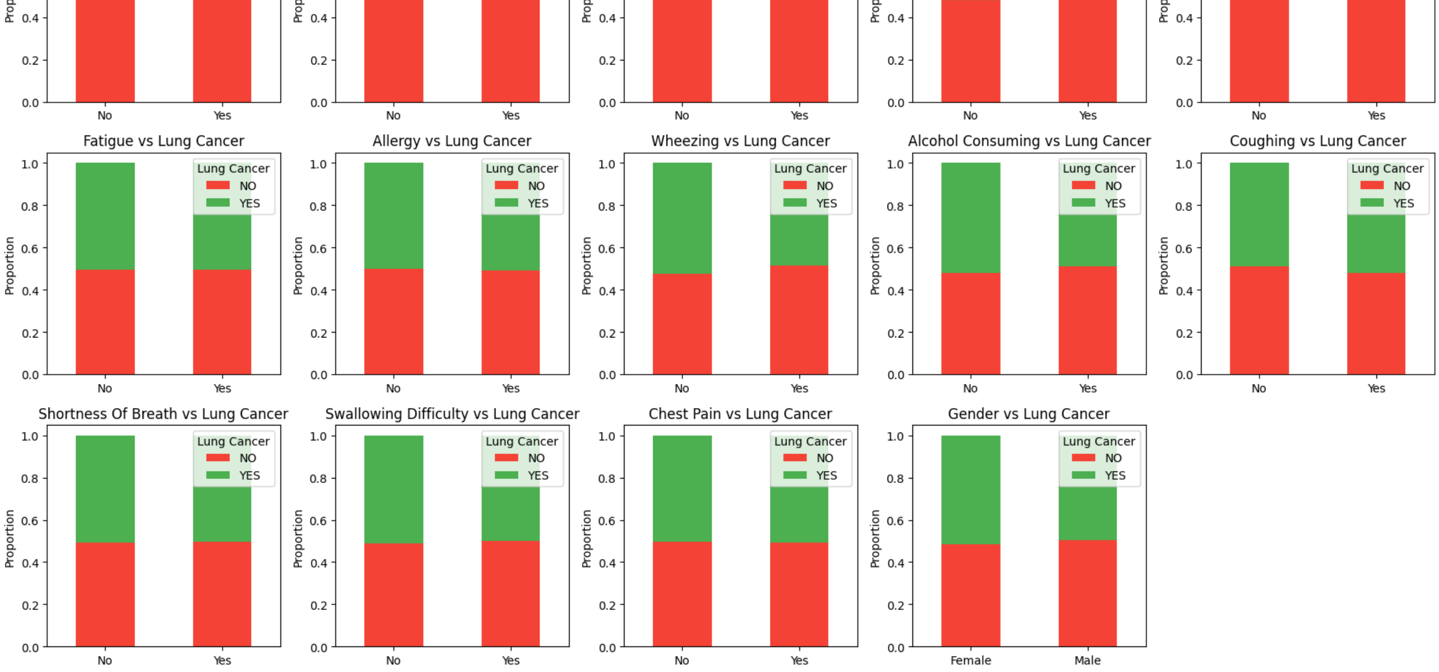


All Columns vs Lung Cancer

In [11]:

```python
features = [
    "SMOKING", "YELLOW_FINGERS", "ANXIETY", "PEER_PRESSURE", "CHRONIC_DISEASE",
    "FATIGUE", "ALLERGY", "WHEEZING", "ALCOHOL_CONSUMING", "COUGHING",
    "SHORTNESS_OF_BREATH", "SWALLOWING_DIFFICULTY", "CHEST_PAIN", "GENDER", "AGE"
]

fig, axes = plt.subplots(3, 5, figsize=(17.5, 10))
axes = axes.flatten()

for i, feature in enumerate(features):
    if feature == "AGE":
        axes[i].axis('off')
        continue
    crosstab = pd.crosstab(df[feature], df["LUNG_CANCER"], normalize='index')
    crosstab.plot(kind='bar', stacked=True, ax=axes[i], color=["#F44336", "#4CAF50"]
)

    axes[i].set_title(f"{feature.replace('_', ' ').title()} vs Lung Cancer")
    axes[i].set_ylabel("Proportion")
    axes[i].set_xlabel("")
    axes[i].legend(title="Lung Cancer", loc='upper right')

    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=0)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

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

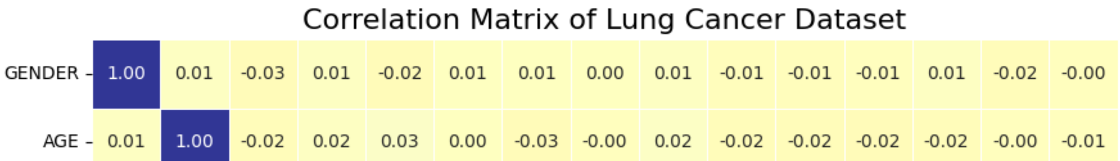


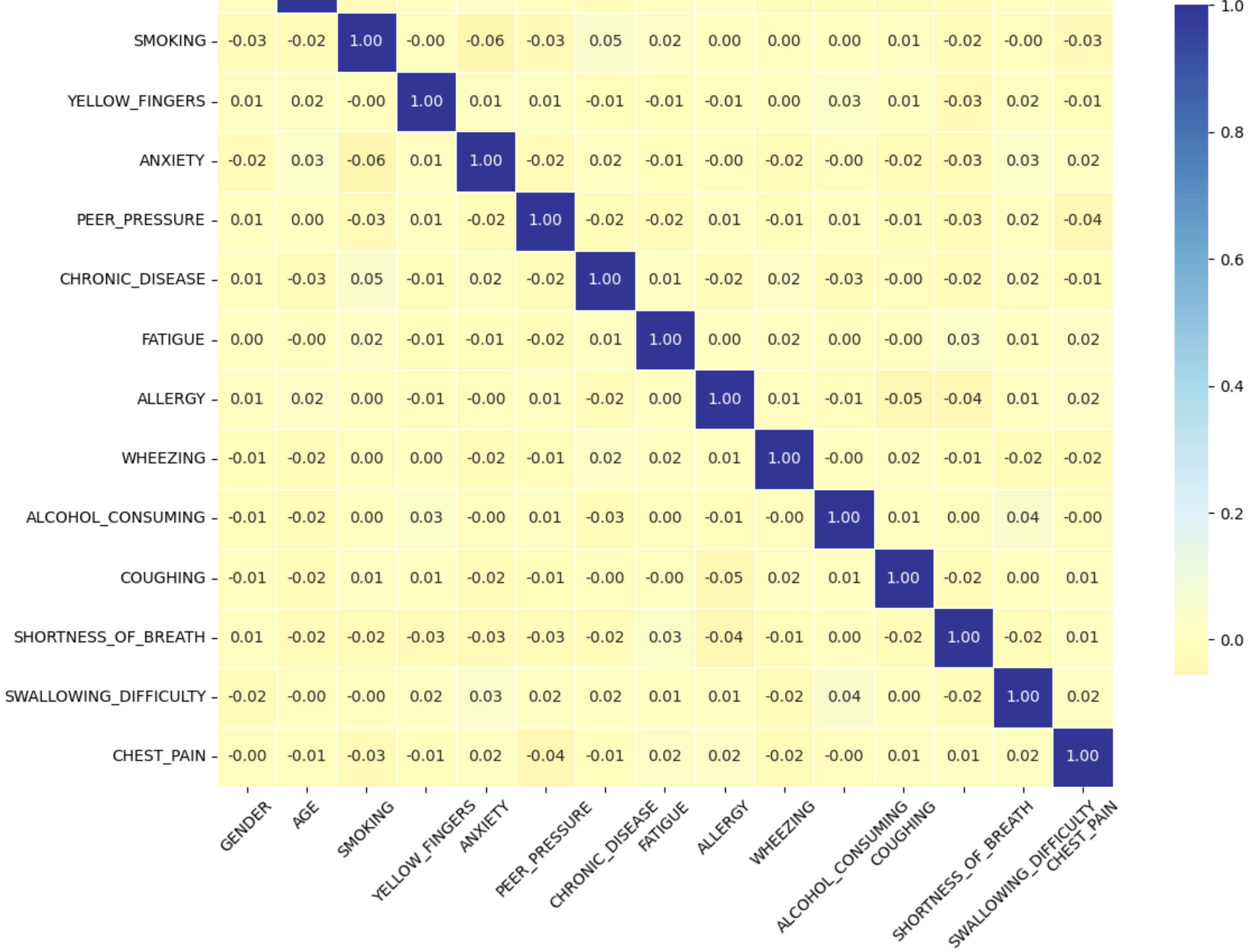


Correlation Matrix of Lung Cancer Dataset 1

In [12]:

```
df_encoded = df.copy()
yes_no_map = {"Yes": 1, "No": 0}
gender_map = {"Male": 1, "Female": 0}

yes_no_columns = [
    "SMOKING", "YELLOW_FINGERS", "ANXIETY", "PEER_PRESSURE",
    "CHRONIC_DISEASE", "FATIGUE", "ALLERGY", "WHEEZING", "ALCOHOL_CONSUMING",
    "COUGHING", "SHORTNESS_OF_BREATH", "SWALLOWING_DIFFICULTY", "CHEST_PAIN", "LUNG_
CANCER"
]

df_encoded[yes_no_columns] = df_encoded[yes_no_columns].map(yes_no_map.get)
df_encoded["GENDER"] = df_encoded["GENDER"].map(gender_map)

df_numeric = df_encoded.select_dtypes(include=["number"])

corr_matrix = df_numeric.corr()

plt.figure(figsize=(13, 10))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="RdYlBu",
    center=0,
    linewidths=0.5,
    square=True,
    cbar_kws={"shrink": 0.75}
)
plt.title("Correlation Matrix of Lung Cancer Dataset", fontsize=16)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```





Pulling the data from the second dataset and visualizing it

In [13]:

```
file_path2 = "dataset2.csv"
df2 = pd.read_csv(file_path2)
df2.head()
```

Out[13]:

	index	Patient Id	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	...	Fatigue	Weight Loss
0	0	P1	33	1	2	4	5	4	3	2	...	3	4
1	1	P10	17	1	3	1	5	3	4	2	...	1	3
2	2	P100	35	1	4	5	6	5	5	4	...	8	7
3	3	P1000	37	1	7	7	7	7	6	7	...	4	2
4	4	P101	46	1	6	8	7	7	7	6	...	3	2

5 rows × 26 columns

Removing the index and Patient Id columns

In [14]:

```
df2 = df2.iloc[:, 2:]
```

In [15]:

```
df2.head()
```

Out[15]:

	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	Balanced Diet	Obesity	...	Fatigue	Weig Los
0	33	1	2	4	5	4	3	2	2	4	...	3	
1	17	1	3	1	5	3	4	2	2	2	...	1	
2	35	1	4	5	6	5	5	4	6	7	...	8	
3	37	1	7	7	7	7	6	7	7	7	...	4	
4	46	1	6	8	7	7	7	6	7	7	...	3	

5 rows × 24 columns

In [16]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Age                        1000 non-null   int64 
 1   Gender                     1000 non-null   int64 
 2   Air Pollution              1000 non-null   int64 
 3   Alcohol use                1000 non-null   int64 
 4   Dust Allergy               1000 non-null   int64 
 5   OccuPational Hazards       1000 non-null   int64 
 6   Genetic Risk               1000 non-null   int64 
 7   chronic Lung Disease       1000 non-null   int64 
 8   Balanced Diet              1000 non-null   int64 
 9   Obesity                    1000 non-null   int64 
 10  Smoking                    1000 non-null   int64 
 11  Passive Smoker             1000 non-null   int64 
 12  Chest Pain                 1000 non-null   int64 
 13  Coughing of Blood          1000 non-null   int64 
 14  Fatigue                    1000 non-null   int64 
 15  Weight Loss                1000 non-null   int64 
 16  Shortness of Breath        1000 non-null   int64 
 17  Wheezing                   1000 non-null   int64 
 18  Swallowing Difficulty      1000 non-null   int64 
 19  Clubbing of Finger Nails   1000 non-null   int64 
 20  Frequent Cold              1000 non-null   int64 
 21  Dry Cough                  1000 non-null   int64 
 22  Snoring                    1000 non-null   int64 
 23  Level                      1000 non-null   object
dtypes: int64(23), object(1)
memory usage: 187.6+ KB
```

In [17]:

```
df2.iloc[:, 0:11].describe()
```

Out[17]:

	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	
count	1000.000000	1000.000000	1000.0000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1(
mean	37.174000	1.402000	3.8400	4.563000	5.165000	4.840000	4.580000	4.380000	
std	12.005493	0.490547	2.0304	2.620477	1.980833	2.107805	2.126999	1.848518	
min	14.000000	1.000000	1.0000	1.000000	1.000000	1.000000	1.000000	1.000000	
25%	27.750000	1.000000	2.0000	2.000000	4.000000	3.000000	2.000000	3.000000	
50%	36.000000	1.000000	3.0000	5.000000	6.000000	5.000000	5.000000	4.000000	
75%	45.000000	2.000000	6.0000	7.000000	7.000000	7.000000	7.000000	6.000000	
max	73.000000	2.000000	8.0000	8.000000	8.000000	8.000000	7.000000	7.000000	

In [18]:

```
df2.iloc[:, 12:].describe()
```

Out[18]:

	Chest Pain	Coughing of Blood	Fatigue	Weight Loss	Shortness of Breath	Wheezing	Swallowing Difficulty	Clubbing of Finger Nails	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1
mean	4.438000	4.859000	3.856000	3.855000	4.240000	3.777000	3.746000	3.923000	
std	2.280209	2.427965	2.244616	2.206546	2.285087	2.041921	2.270383	2.388048	
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	
25%	2.000000	3.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	
50%	4.000000	4.000000	3.000000	3.000000	4.000000	4.000000	4.000000	4.000000	
75%	7.000000	7.000000	5.000000	6.000000	6.000000	5.000000	5.000000	5.000000	
max	9.000000	9.000000	9.000000	8.000000	9.000000	8.000000	8.000000	9.000000	

In [19]:

```
df2.shape
```

Out[19]:

```
(1000, 24)
```

Preparing the second dataset for visualization

In [20]:

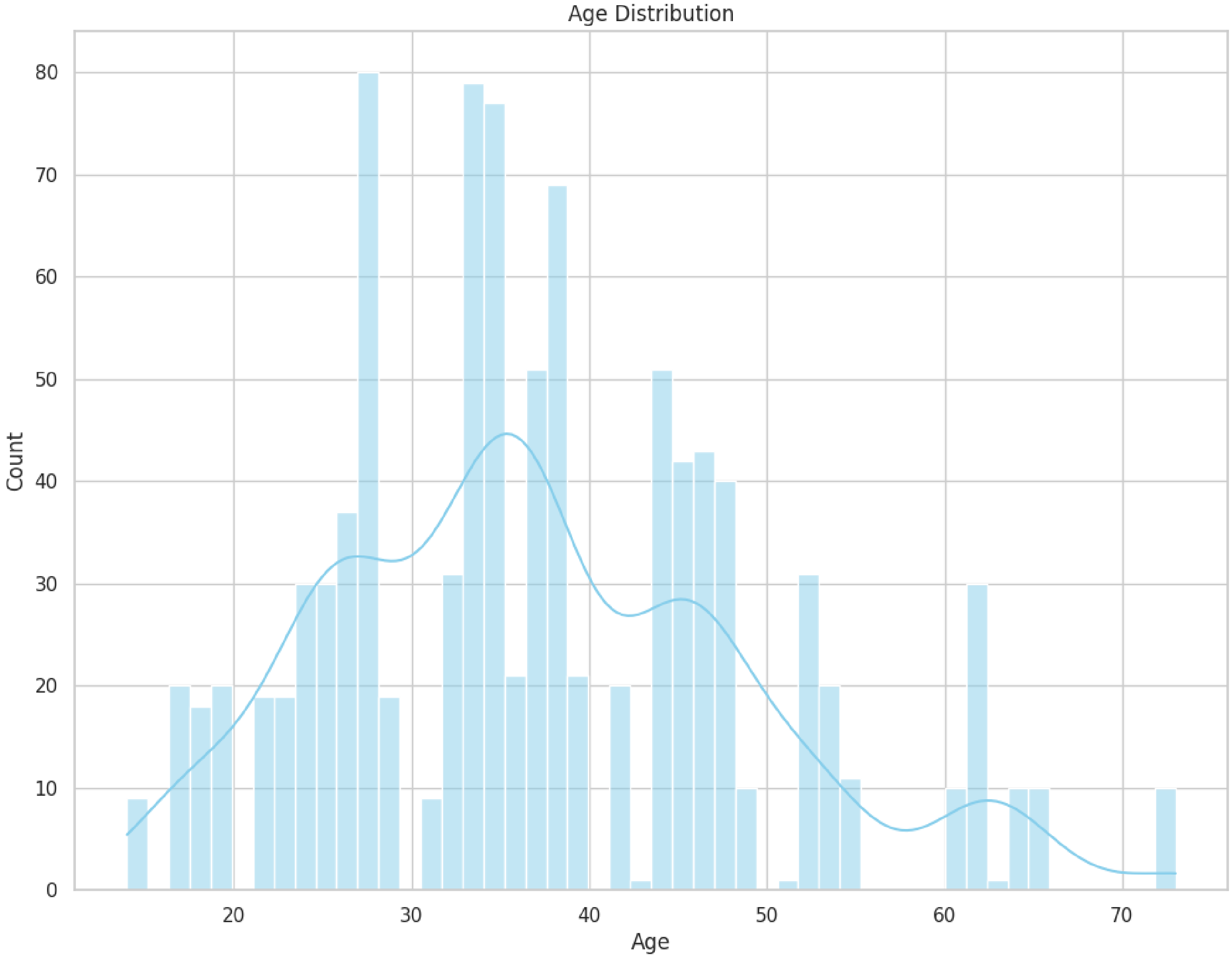
```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
df2['Gender'] = df2['Gender'].map({1: 'Male', 2: 'Female'})
df2['Level'] = df2['Level'].astype("category")
```

```
<Figure size 1000x600 with 0 Axes>
```

Plotting the age distribution of the patients

In [21]:

```
plt.figure(figsize=(12, 9))
sns.histplot(df2['Age'], bins=50, kde=True, color='skyblue')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()

mean_age = df2['Age'].mean()
median_age = df2['Age'].median()
mode_age = df2['Age'].mode()[0]

print(f"Mean Age: {mean_age}")
print(f"Median Age: {median_age}")
print(f"Mode Age: {mode_age}")
```



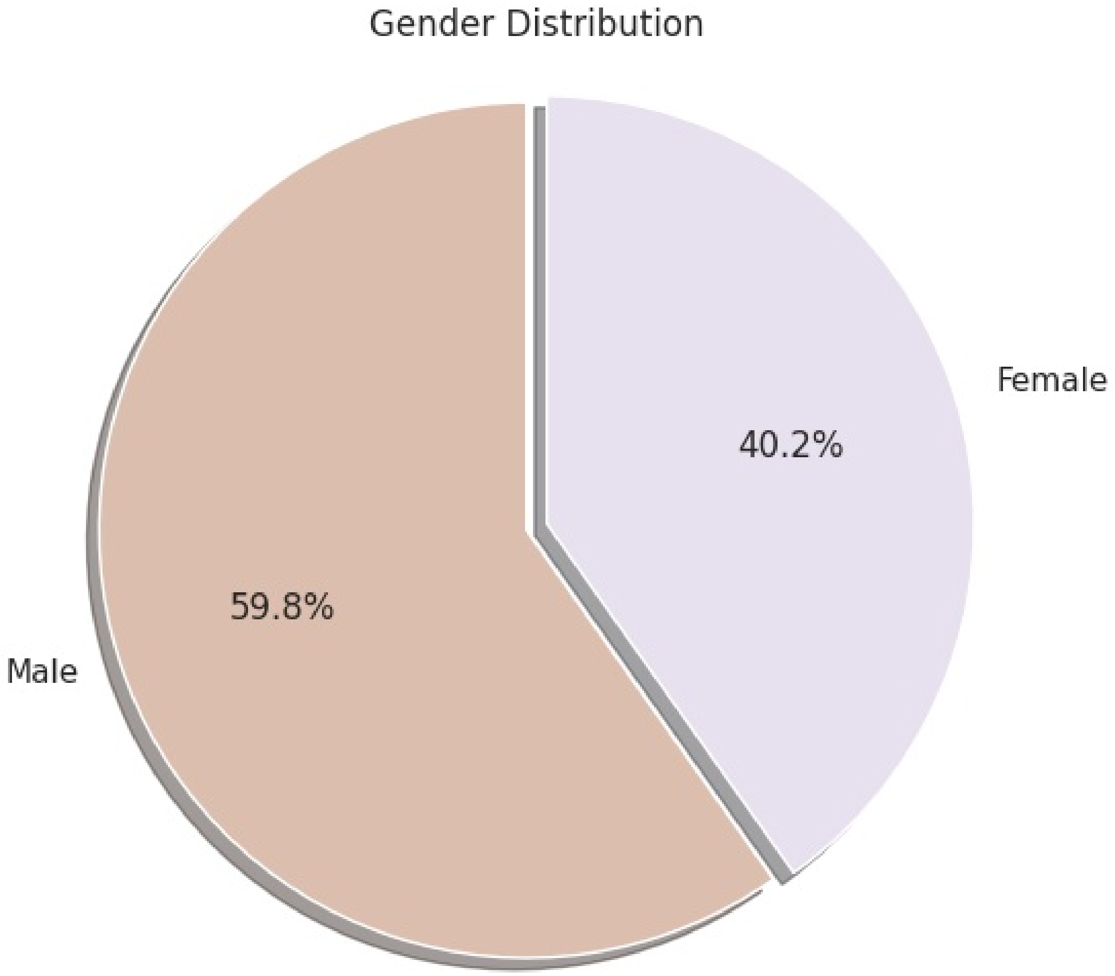
```
Mean Age: 37.174
Median Age: 36.0
Mode Age: 35
```

Plotting the gender distributions of the patients

In [22]:

```
gender_counts = df2["Gender"].value_counts()

colors = ["#dcbeaf", "#e7e0ee"]
explode = (0.05, 0)
```

```
plt.figure(figsize=(6, 6))
plt.pie(
    gender_counts,
    labels=gender_counts.index,
    autopct="%1.1f%%",
    startangle=90,
    colors=colors,
    explode=explode,
    shadow=True
)
plt.title("Gender Distribution")
plt.axis("equal")
plt.show()
```

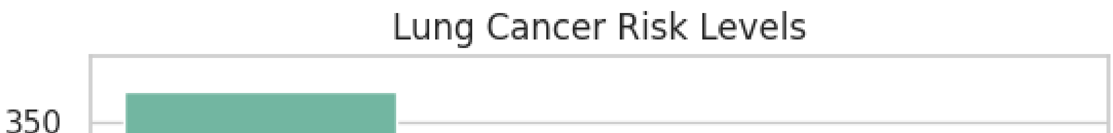


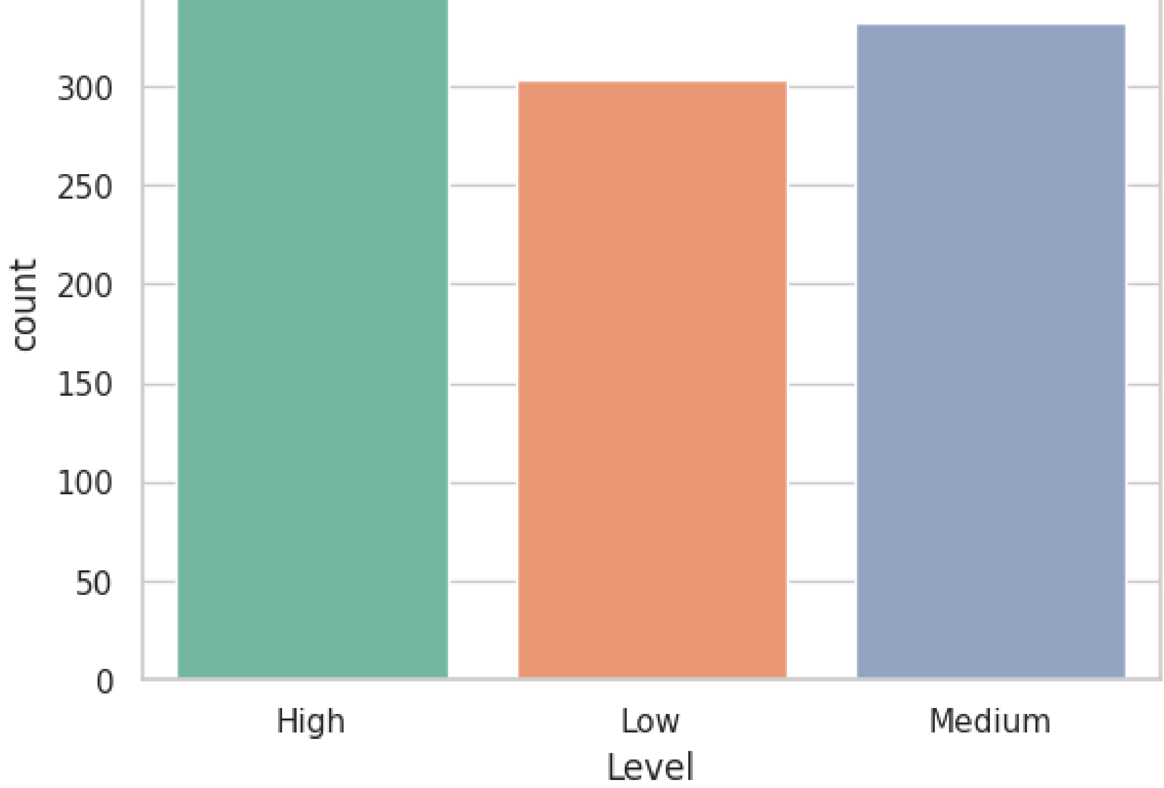
Plotting the risks of Lung Cancer of the patients

In [23]:

```
sns.countplot(data=df2, x='Level', palette='Set2')
plt.title('Lung Cancer Risk Levels')
plt.show();
```

```
<ipython-input-23-80d4d314cecc>:1: 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(data=df2, x='Level', palette='Set2')
```



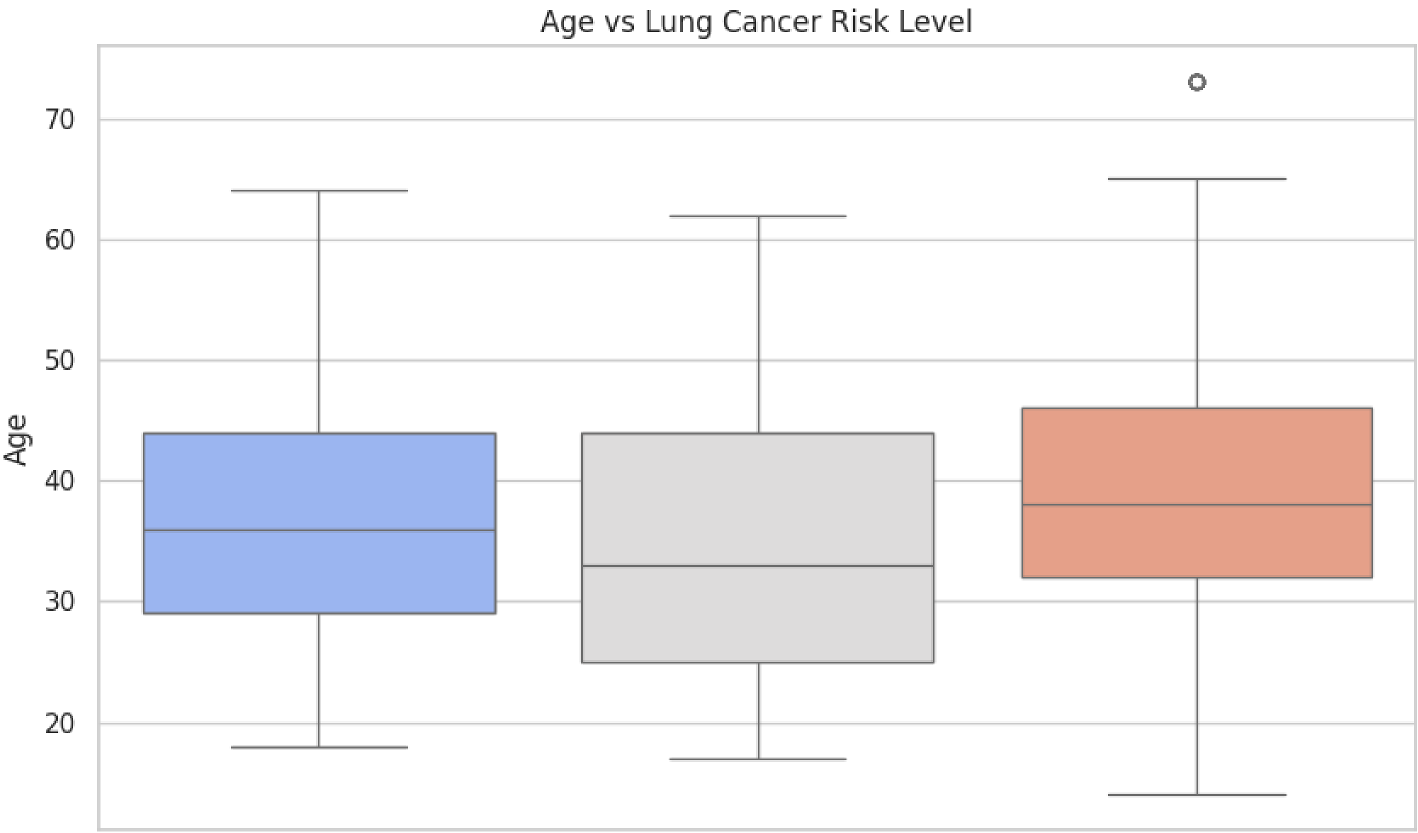


Age vs Lung Cancer Boxplot

In [24]:

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df2, x='Level', y='Age', palette='coolwarm')
plt.title('Age vs Lung Cancer Risk Level')
plt.show()
```

```
<ipython-input-24-db9e4c8384cb>:2: 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.boxplot(data=df2, x='Level', y='Age', palette='coolwarm')
```

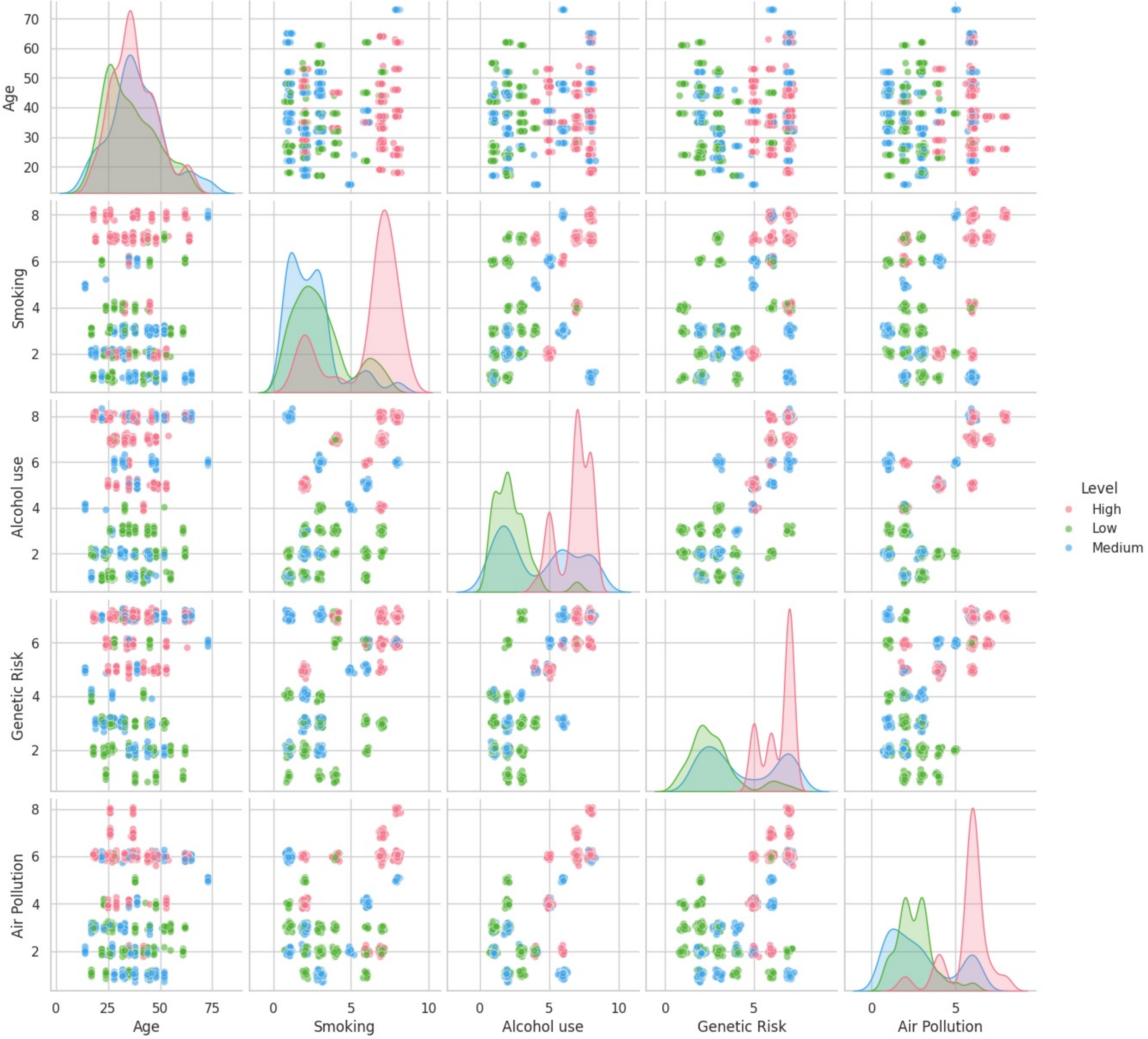


High Low Medium
Level

Pairplot of the selected columns

In [25]:

```
selected = ['Age', 'Smoking', 'Alcohol use', 'Genetic Risk', 'Air Pollution', 'Level']
df_jittered = df2.copy()
jitter_cols = ['Smoking', 'Alcohol use', 'Genetic Risk', 'Air Pollution']

for col in jitter_cols:
    df_jittered[col] = df_jittered[col] + np.random.normal(0, 0.1, size=len(df2))

sns.pairplot(df_jittered[selected], hue='Level', palette='husl', plot_kws={'alpha': 0.6});
```

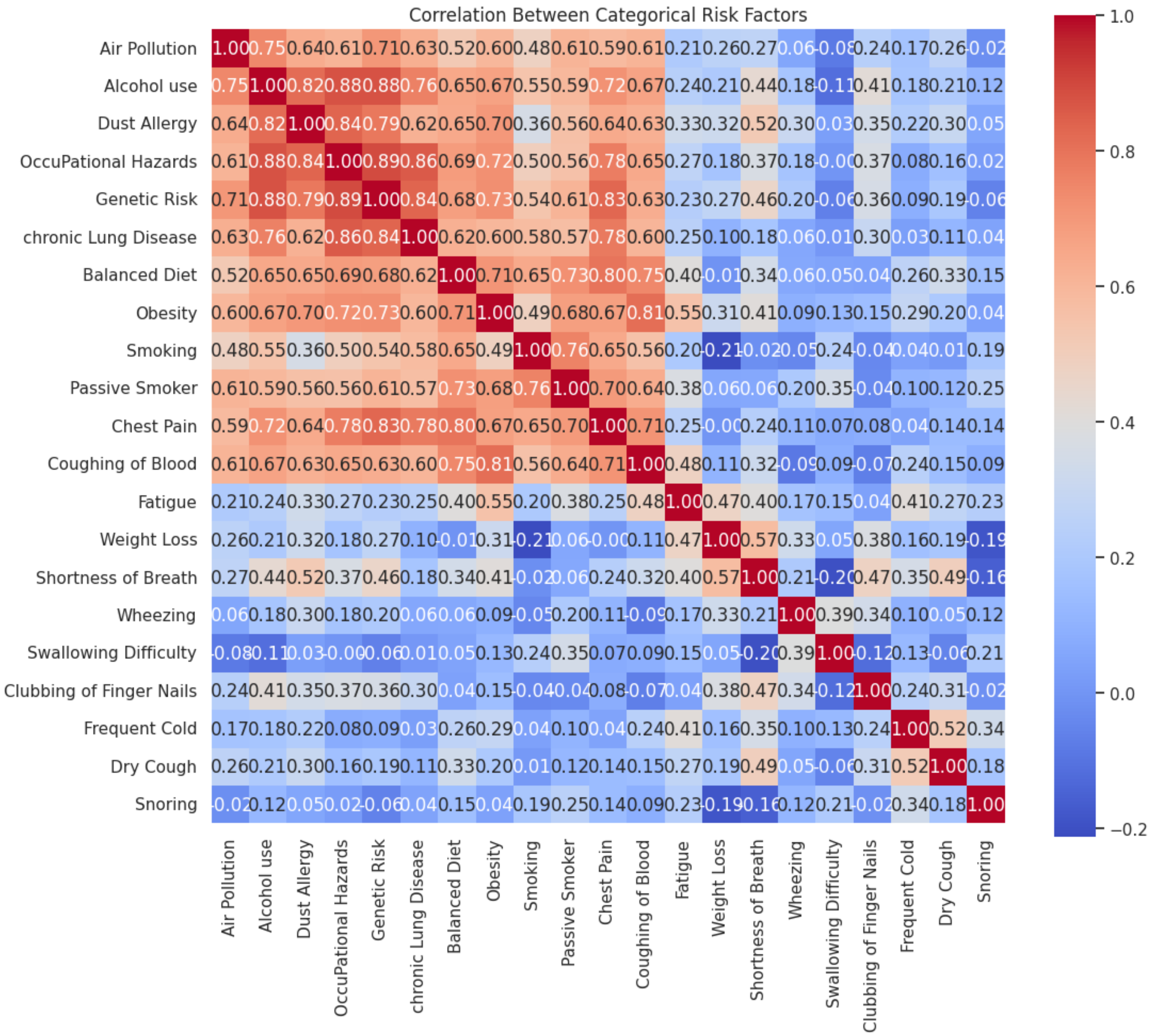


Correlation between the columns

In [26]:

```
categorical_cols = df2.drop(columns=['Age', 'Gender', 'Level']).columns
plt.figure(figsize=(12, 10))
```

```
corr_matrix = df2[categorical_cols].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title("Correlation Between Categorical Risk Factors")
plt.show()
```



Hypothesis Tests

This section will be about some hypothesis on the causes of lung cancer. The two datasets will be compared using hypothesis tests on the same hypotheses.

1. **Hypothesis: Smoking regularly affects the risk of lung cancer.**

- **Null Hypothesis (H₀)**: Smoking and lung cancer are independent
- **Alternative Hypothesis (H₁)**: Smoking and lung cancer are associated

Dataset 1

In [27]:

```
df["SMOKING"] = df["SMOKING"].map({"Yes": 1, "No": 0})
df["LUNG_CANCER"] = df["LUNG_CANCER"].map({"YES": 1, "NO": 0})
```

```
contingency_table = pd.crosstab(df["SMOKING"], df["LUNG_CANCER"])

# --- Run chi-square test ---
chi2, p, dof, expected = chi2_contingency(contingency_table)

# --- Print results ---
print("Contingency Table:\n", contingency_table)
print("\nChi-square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-value:", p)

# --- Interpret result ---
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: Smoking is associated with lung cancer.")
else:
    print("Fail to reject the null hypothesis: No significant association between smoking and lung cancer.")
```

```
Contingency Table:
 LUNG_CANCER    0    1
SMOKING
0             765  762
1             717  756

Chi-square Statistic: 0.5510111488430194
Degrees of Freedom: 1
P-value: 0.45790483338496013
Fail to reject the null hypothesis: No significant association between smoking and lung cancer.
```

Dataset 2

In [28]:

```
df2["Level"] = df2["Level"].map({"Low": 0, "Medium": 1, "High": 2})
contingency_table = pd.crosstab(df2['Smoking'], df2['Level'])

# Perform the Chi-Square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Output results
print("Contingency Table:\n", contingency_table)
print("Chi-Square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-Value:", p_value)

# Interpret the result
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: Smoking is associated with lung cancer.")
else:
    print("Fail to reject the null hypothesis: No significant association between smoking and lung cancer.")
```

```
Contingency Table:
 Level     2   0    1
Smoking
1         0  61  120
2        70  81   71
3         0  71  101
4        19  40    0
5         0   0   10
```

```
6          10   30   20
7         187   20    0
8          79    0   10
Chi-Square Statistic: 684.4965209399204
Degrees of Freedom: 14
P-Value: 5.246169667913276e-137
Reject the null hypothesis: Smoking is associated with lung cancer.
```

According to the first dataset, smoking regularly does not have a significant relation with lung cancer. However, the test for the second dataset tells us the otherwise.

1. **Hypothesis: Having chronic lung diseases affect the risk of lung cancer.**

- **Null Hypothesis (H_0): Chronic lung diseases and lung cancer are independent**
- **Alternative Hypothesis (H_1): Chronic lung diseases and lung cancer are associated**

Dataset 1

In [29]:

```
df["CHRONIC_DISEASE"] = df["CHRONIC_DISEASE"].map({"Yes": 1, "No": 0})

contingency_table = pd.crosstab(df["CHRONIC_DISEASE"], df["LUNG_CANCER"])

# --- Run chi-square test ---
chi2, p, dof, expected = chi2_contingency(contingency_table)

# --- Print results ---
print("Contingency Table:\n", contingency_table)
print("\nChi-square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-value:", p)

# --- Interpret result ---
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: Chronic diseases are associated with lung cancer.")
else:
    print("Fail to reject the null hypothesis: No significant association between chronic diseases and lung cancer.")
```

```
Contingency Table:
 LUNG_CANCER        0    1
CHRONIC_DISEASE
0                719  752
1                763  766

Chi-square Statistic: 0.27462898733847857
Degrees of Freedom: 1
P-value: 0.6002433756617297
Fail to reject the null hypothesis: No significant association between chronic
diseases and lung cancer.
```

Dataset 2

In [30]:

```
contingency_table = pd.crosstab(df2['chronic Lung Disease'], df2['Level'])

# Perform the Chi-Square test
```

```
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Output results
print("Contingency Table:\n", contingency_table)
print("Chi-Square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-Value:", p_value)

# Interpret the result
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: Chronic diseases are associated with lung 
cancer.")
else:
    print("Fail to reject the null hypothesis: No significant association between 
chronic diseases and lung cancer.")
```

```
Contingency Table:
 Level                  2   0    1
chronic Lung Disease
1                      0  50    0
2                      0  82   91
3                      0  81   60
4                     70  20   51
5                     10  40   30
6                    198  10  100
7                     87  20    0
Chi-Square Statistic: 585.1295399409582
Degrees of Freedom: 12
P-Value: 1.5853456732363079e-117
Reject the null hypothesis: Chronic diseases are associated with lung cancer.
```

According to the first dataset, chronic lung diseases regularly do not have a significant relation with lung cancer. However, the test for the second dataset tells us the otherwise.

1. **Hypothesis: Gender has a relation with lung cancer.**

- **Null Hypothesis (H_0): Gender and lung cancer is independent**
- **Alternative Hypothesis (H_1): Gender and lung cancer is associated**

Dataset 1

In [31]:

```
df["GENDER"] = df["GENDER"].map({"Male": 1, "Female": 2})

contingency_table = pd.crosstab(df["GENDER"], df["LUNG_CANCER"])

# --- Run chi-square test ---
chi2, p, dof, expected = chi2_contingency(contingency_table)

# --- Print results ---
print("Contingency Table:\n", contingency_table)
print("\nChi-square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-value:", p)

# --- Interpret result ---
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: Gender is associated with lung cancer.")
```

```
else:
    print("Fail to reject the null hypothesis: No significant association between
gender and lung cancer.")
```

```
Contingency Table:
 LUNG_CANCER    0    1
GENDER
1             760  754
2             722  764

Chi-square Statistic: 0.7158403950228966
Degrees of Freedom: 1
P-value: 0.3975117338177119
Fail to reject the null hypothesis: No significant association between gender and l
ung cancer.
```

Dataset 2

In [32]:

```
contingency_table = pd.crosstab(df2['Gender'], df2['Level'])

# Perform the Chi-Square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Output results
print("Contingency Table:\n", contingency_table)
print("Chi-Square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-Value:", p_value)

# Interpret the result
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: Gender is associated with lung cancer.")
else:
    print("Fail to reject the null hypothesis: No significant association between
gender and lung cancer.")
```

```
Contingency Table:
 Level     2    0    1
Gender
Female  113  154  135
Male    252  149  197
Chi-Square Statistic: 27.22494142912069
Degrees of Freedom: 2
P-Value: 1.2251212802771069e-06
Reject the null hypothesis: Gender is associated with lung cancer.
```

According to the first dataset, gender regularly do not have a significant relation with lung cancer. However, the test for the second dataset tells us the otherwise.

Even though all of the test results were different for both datasets, the reason may only be the sizes of them. If both of the datasets had 10k+ patients, the result might have been similar.

Creating machine learning models using sklearn

In [33]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
roc_curve, auc
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Set seaborn style
sns.set(style="whitegrid")
```

In [34]:

```
df.head()
```

Out[34]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE	FATIGUE	ALLI
0	1	65	0	No	No	Yes	1	No	
1	2	55	0	Yes	Yes	No	0	Yes	
2	2	78	1	Yes	No	No	0	Yes	
3	1	60	1	No	No	No	1	No	
4	2	80	0	No	Yes	No	0	Yes	

In [35]:

```
df2.head()
```

Out[35]:

	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	Balanced Diet	Obesity	...	Fatigue	Weig Los
0	33	Male	2	4	5	4	3	2	2	4	...	3	
1	17	Male	3	1	5	3	4	2	2	2	...	1	
2	35	Male	4	5	6	5	5	4	6	7	...	8	
3	37	Male	7	7	7	7	6	7	7	7	...	4	
4	46	Male	6	8	7	7	7	6	7	7	...	3	

5 rows × 24 columns

In [36]:

```
# Replace categorical 'Yes'/'No' and '1'/'2' with 1/0
df = df.replace({"Yes": 1, "No": 0, "2": 1, "1": 0})

# Split features (X) and target (y)
X = df.drop("LUNG_CANCER", axis=1)
y = df["LUNG_CANCER"]

# Check the distribution of the target variable
print(f"Target distribution:\n{y.value_counts()}")
```

```
Target distribution:
LUNG_CANCER
1     1518
```

```
1    1518
0    1482
Name: count, dtype: int64
```

```
<ipython-input-36-e98bae2b7c38>:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df = df.replace({"Yes": 1, "No": 0, "2": 1, "1": 0})
```

In [37]:

```
df.head()
```

Out[37]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC_DISEASE	FATIGUE	ALLI
0	1	65	0	0	0	1	1	0	
1	2	55	0	1	1	0	0	1	
2	2	78	1	1	0	0	0	1	
3	1	60	1	0	0	0	1	0	
4	2	80	0	0	1	0	0	1	

In [38]:

```
# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Check the shape of train/test datasets
print(f"Training set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")
```

```
Training set: (2400, 15)
Testing set: (600, 15)
```

This section of the code focuses on training and evaluating several common machine learning classification models using the prepared data. The goal is to see which model performs best at predicting the target variable, which in this case is likely related to lung cancer status.

In [39]:

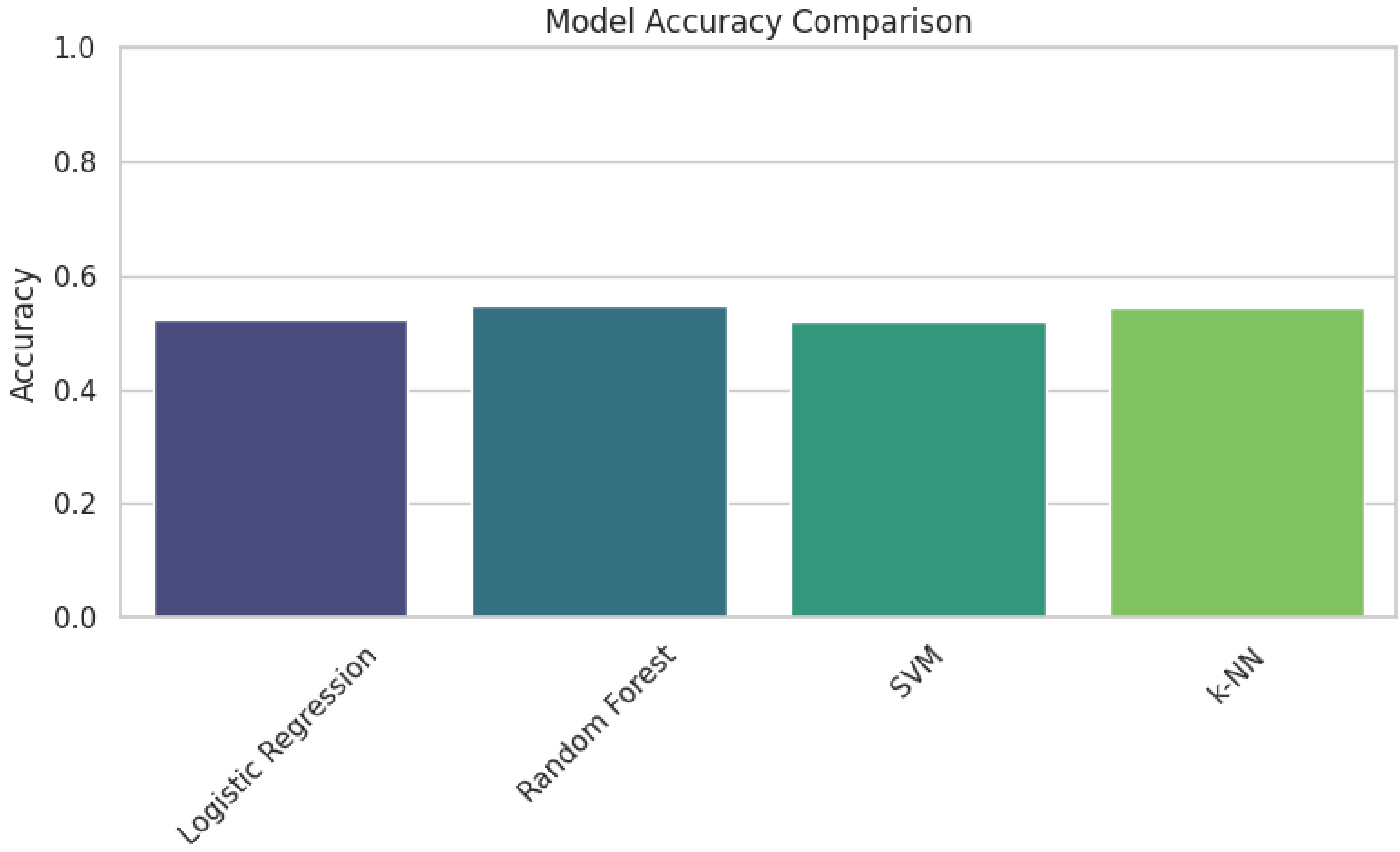
```
%%time
# Define the classification models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(probability=True),
    "k-NN": KNeighborsClassifier()
}

# Store results
results = {}
y_probs = {}

# Train each model and evaluate performance
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    results[name] = {
        "accuracy": acc,
        "classification_report": report,
        "conf_matrix": confusion_matrix(y_test, y_pred)
    }

    if hasattr(model, "predict_proba"):
        y_probs[name] = model.predict_proba(X_test)[:, 1]

    print(f"\n{name}")
    print(f"Accuracy: {acc:.2f}")
    print("Classification Report:\n", report)
```

```
Logistic Regression
Accuracy: 0.52
Classification Report:
               precision    recall  f1-score   support

           0       0.52      0.45      0.48       296
           1       0.53      0.59      0.56       304

    accuracy                           0.52       600
   macro avg       0.52      0.52      0.52       600
weighted avg       0.52      0.52      0.52       600


Random Forest
Accuracy: 0.55
Classification Report:
               precision    recall  f1-score   support

           0       0.54      0.54      0.54       296
           1       0.55      0.56      0.56       304

    accuracy                           0.55       600
   macro avg       0.55      0.55      0.55       600
weighted avg       0.55      0.55      0.55       600


SVM
Accuracy: 0.52
Classification Report:
               precision    recall  f1-score   support

           0       0.52      0.44      0.48       296
           1       0.52      0.60      0.56       304

    accuracy                           0.52       600
   macro avg       0.52      0.52      0.52       600
weighted avg       0.52      0.52      0.52       600


k-NN
Accuracy: 0.55
Classification Report:
               precision    recall  f1-score   support

           0       0.54      0.56      0.55       296
           1       0.55      0.54      0.55       304

    accuracy                           0.55       600
   macro avg       0.55      0.55      0.55       600
weighted avg       0.55      0.55      0.55       600

CPU times: user 2.34 s, sys: 79 ms, total: 2.41 s
Wall time: 2.29 s
```

In [40]:

```
# Plotting model accuracy comparison
accuracies = [results[m]["accuracy"] for m in models]
plt.figure(figsize=(8, 5))
sns.barplot(x=list(models.keys()), y=accuracies, palette="viridis")
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

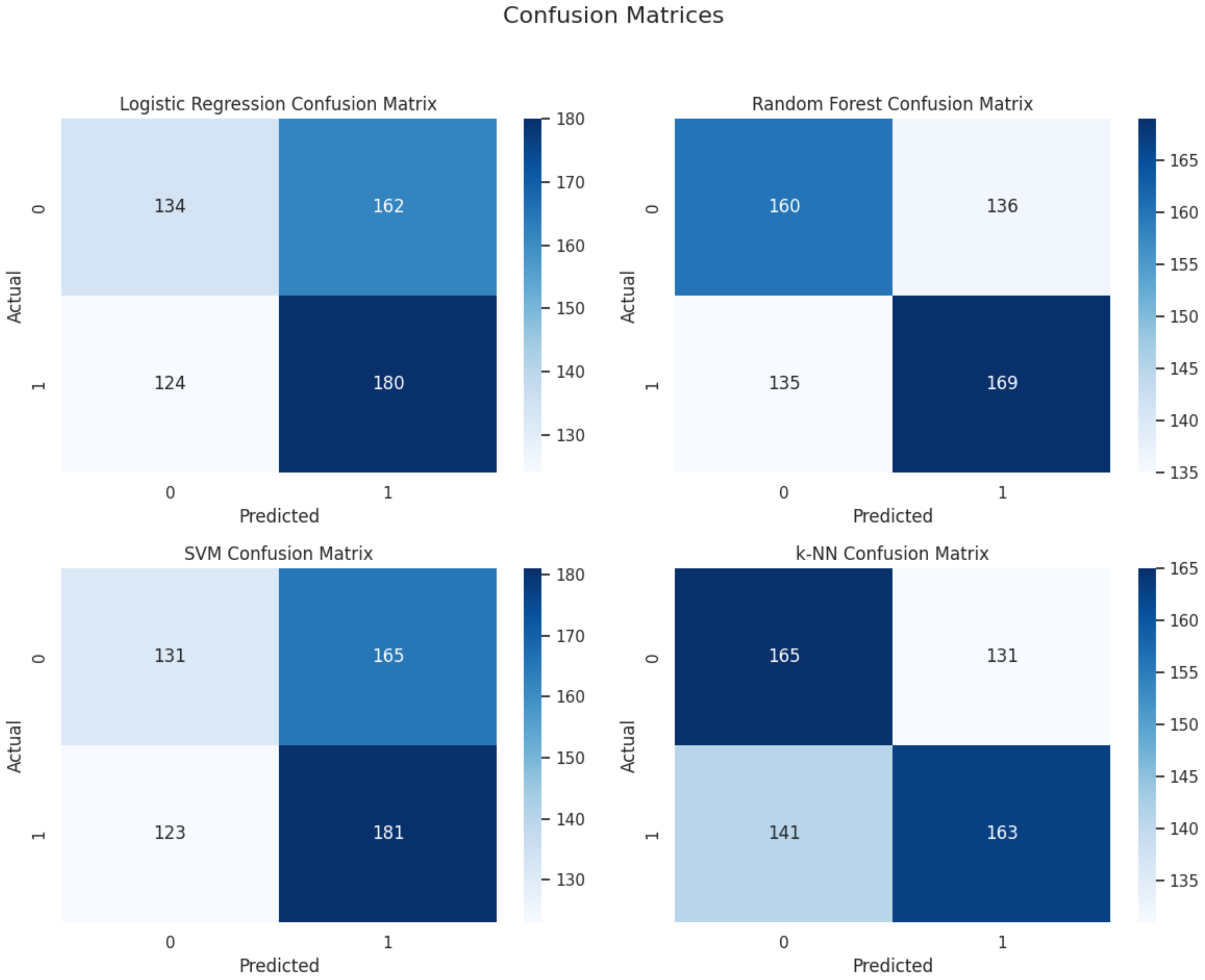
```
<ipython-input-40-5f2253f25700>:4: FutureWarning:

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.barplot(x=list(models.keys()), y=accuracies, palette="viridis")
```



In [41]:

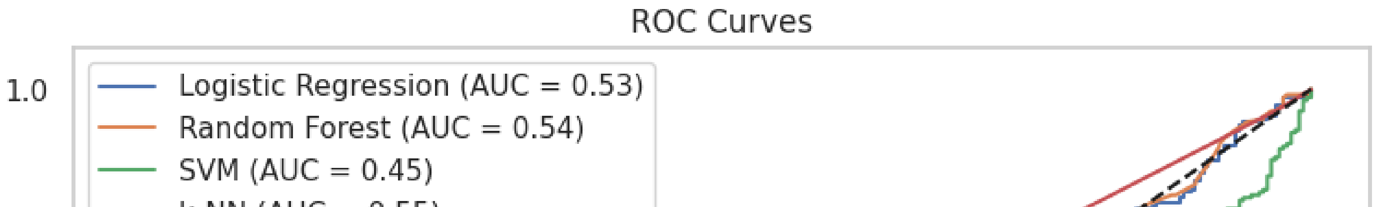
```
# Plot confusion matrices for each model
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()

for idx, name in enumerate(models):
    sns.heatmap(results[name]["conf_matrix"], annot=True, fmt="d", cmap="Blues", ax=
axes[idx])
    axes[idx].set_title(f"{name} Confusion Matrix")
    axes[idx].set_xlabel("Predicted")
    axes[idx].set_ylabel("Actual")
```

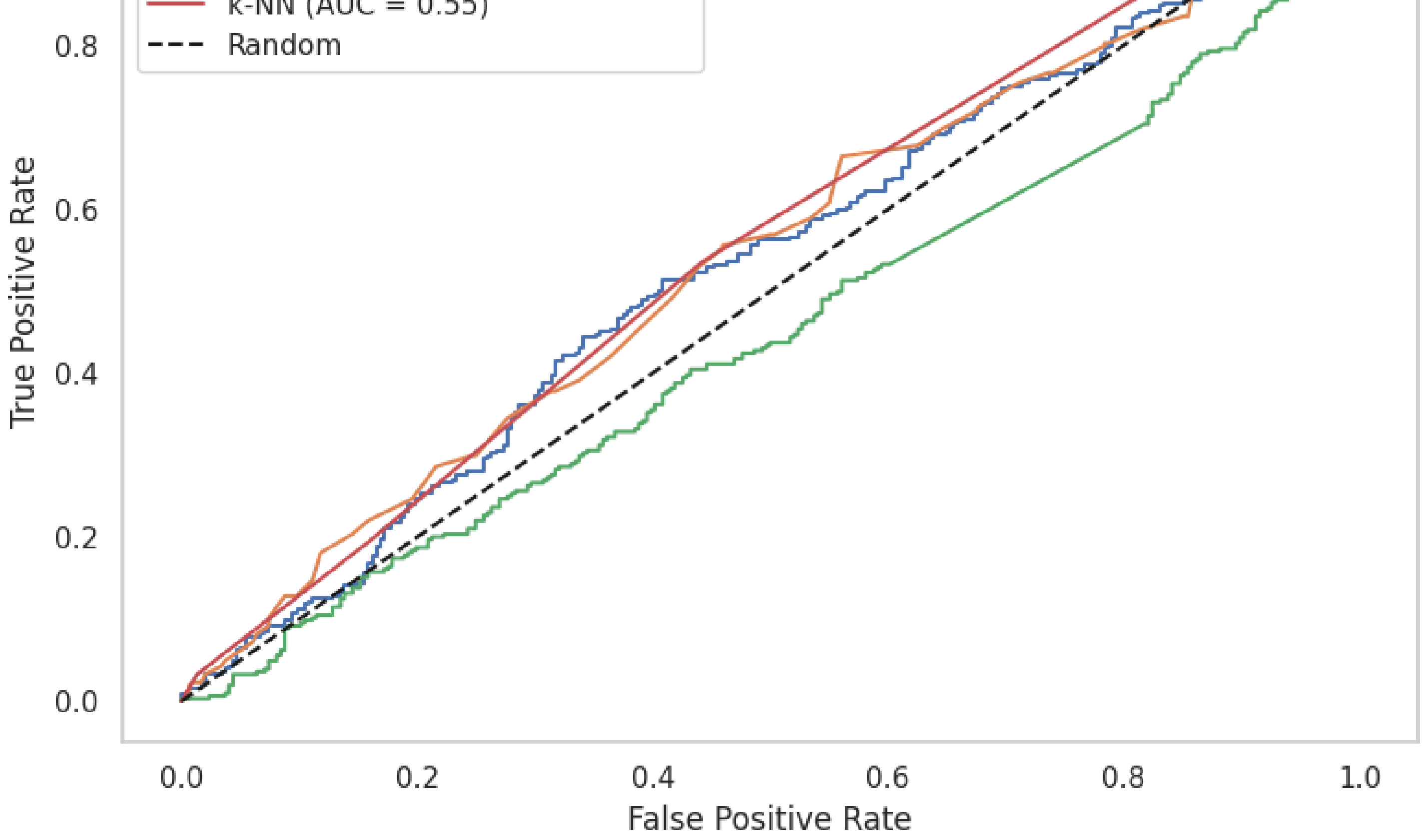
```
plt.suptitle("Confusion Matrices", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



In [42]:

```
# Plot ROC curves for each model
plt.figure(figsize=(8, 6))
for name, probs in y_probs.items():
    fpr, tpr, _ = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")

# Random line for comparison
plt.plot([0, 1], [0, 1], "k--", label="Random")
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```





Overall results show that different types of training models of the first dataset have between 0.50-0.55 accuracy, highest being 0.55 usign k-NN as a the base model. Since the first dataset was relatively simple (as we can see from the correlation matrix), this output was expected.

In [43]:

```
df2['Gender'] = df2['Gender'].map({'Male': 1, 'Female': 2})

# Split features (X) and target (y)
X = df2.drop("Level", axis=1)
y = df2["Level"]

# Check the distribution of the target variable
print(f"Target distribution:\n{y.value_counts()}")
```

```
Target distribution:
Level
2    365
1    332
0    303
Name: count, dtype: int64
```

In [44]:

```
# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Check the shape of train/test datasets
print(f"Training set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")
```

```
Training set: (800, 23)
Testing set: (200, 23)
```

This section of the code trains and evaluates several different machine learning models on the second data. The goal is to see how well each model can predict the target variable.

data. The goal is to see how well each model can predict the target variable.

In [45]:

```
%%time
# Define the classification models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(probability=True),
    "k-NN": KNeighborsClassifier()
}

# Store results
results = {}
y_probs = {}

# Train each model and evaluate performance
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    results[name] = {
        "accuracy": acc,
        "classification_report": report,
        "conf_matrix": confusion_matrix(y_test, y_pred)
    }

    if hasattr(model, "predict_proba"):
        y_probs[name] = model.predict_proba(X_test)[:, 1]

    print(f"\n{name}")
    print(f"Accuracy: {acc:.2f}")
    print("Classification Report:\n", report)
```

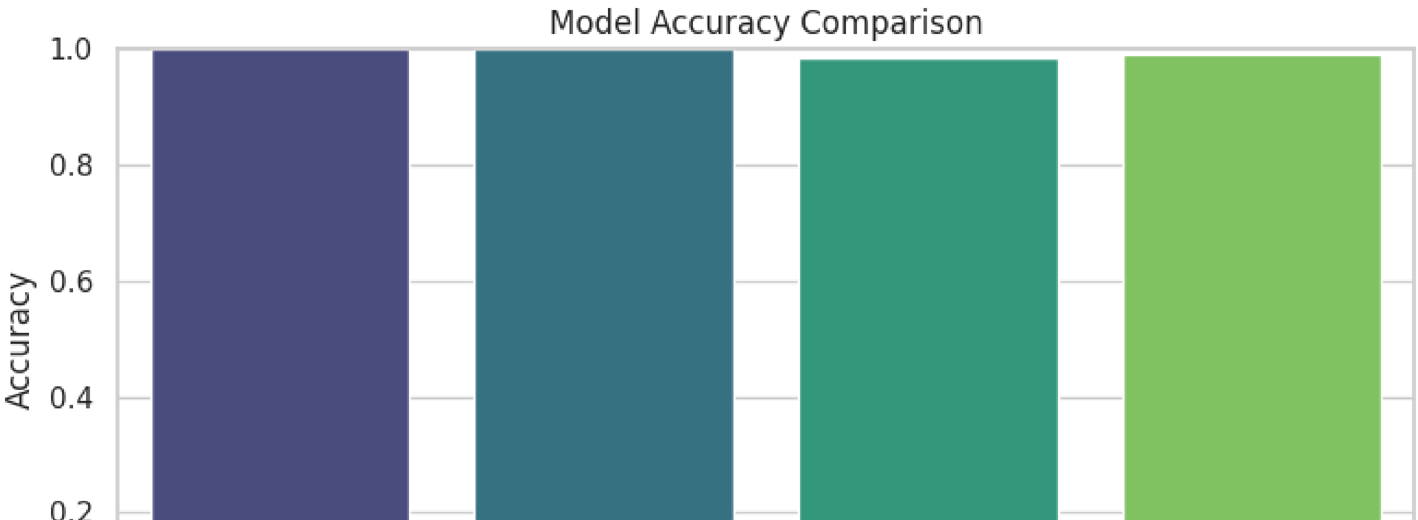
```
Logistic Regression
Accuracy: 1.00
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        61
           1       1.00      1.00      1.00        66
           2       1.00      1.00      1.00        73

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200


Random Forest
Accuracy: 1.00
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        61
           1       1.00      1.00      1.00        66
           2       1.00      1.00      1.00        73

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

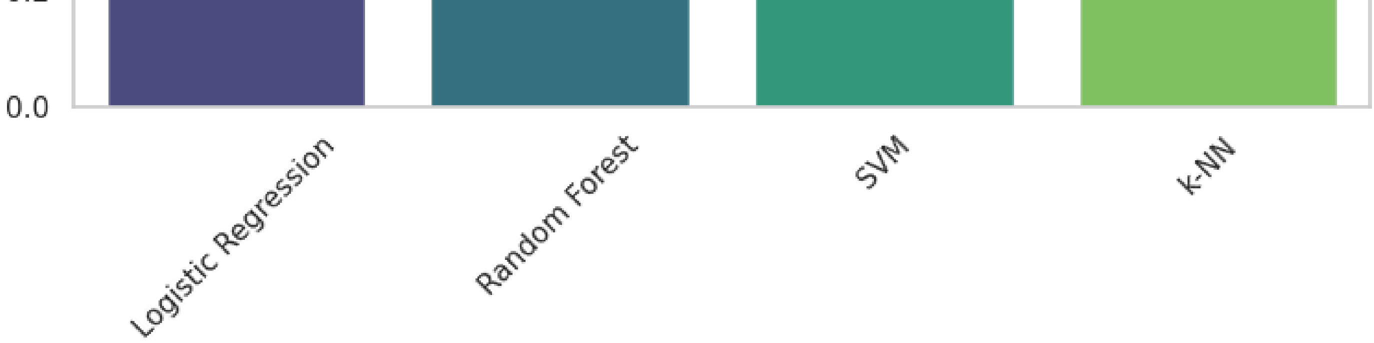
```
SVM
Accuracy: 0.98
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.97      0.98        61
           1       0.97      0.98      0.98        66
           2       0.99      1.00      0.99        73

    accuracy                           0.98       200
   macro avg       0.99      0.98      0.98       200
weighted avg       0.99      0.98      0.98       200


k-NN
Accuracy: 0.99
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.97      0.98        61
           1       0.97      1.00      0.99        66
           2       1.00      1.00      1.00        73

    accuracy                           0.99       200
   macro avg       0.99      0.99      0.99       200
weighted avg       0.99      0.99      0.99       200

CPU times: user 2.76 s, sys: 8.33 ms, total: 2.77 s
Wall time: 2.43 s
```

In [46]:

```
# Plotting model accuracy comparison
accuracies = [results[m]["accuracy"] for m in models]
plt.figure(figsize=(8, 5))
sns.barplot(x=list(models.keys()), y=accuracies, palette="viridis")
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

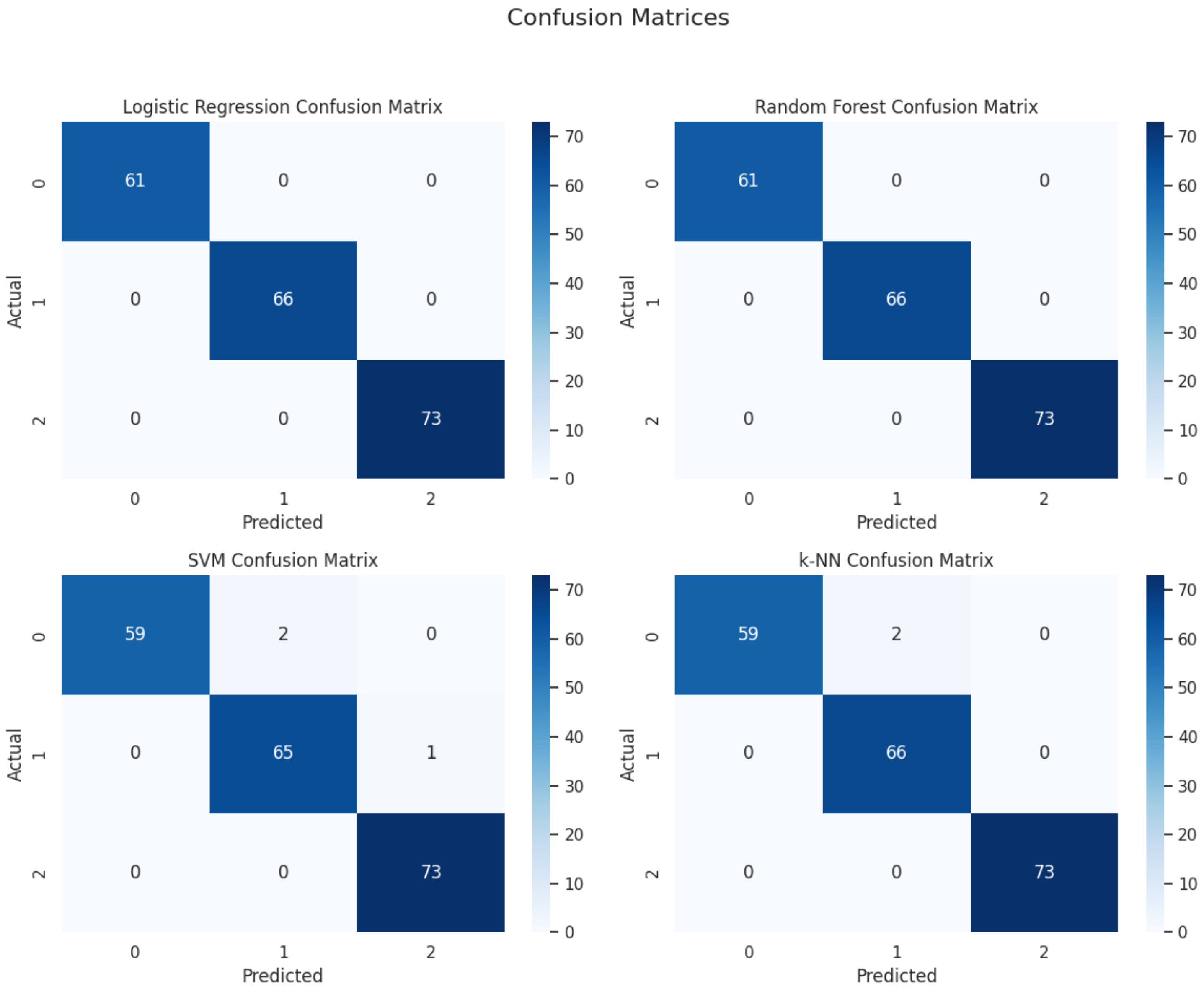
```
<ipython-input-46-5f2253f25700>:4: FutureWarning:

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.barplot(x=list(models.keys()), y=accuracies, palette="viridis")
```





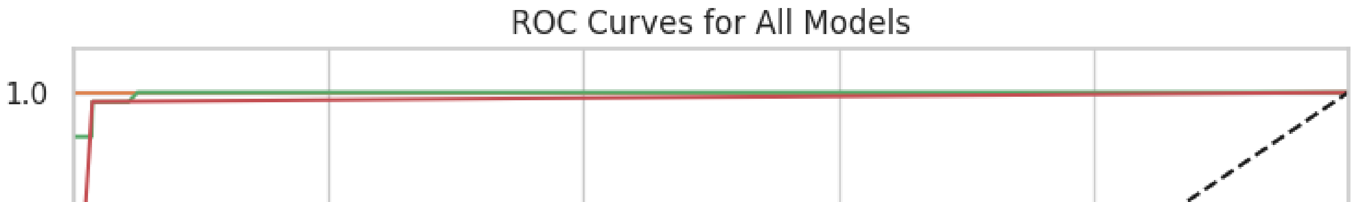
In [47]:

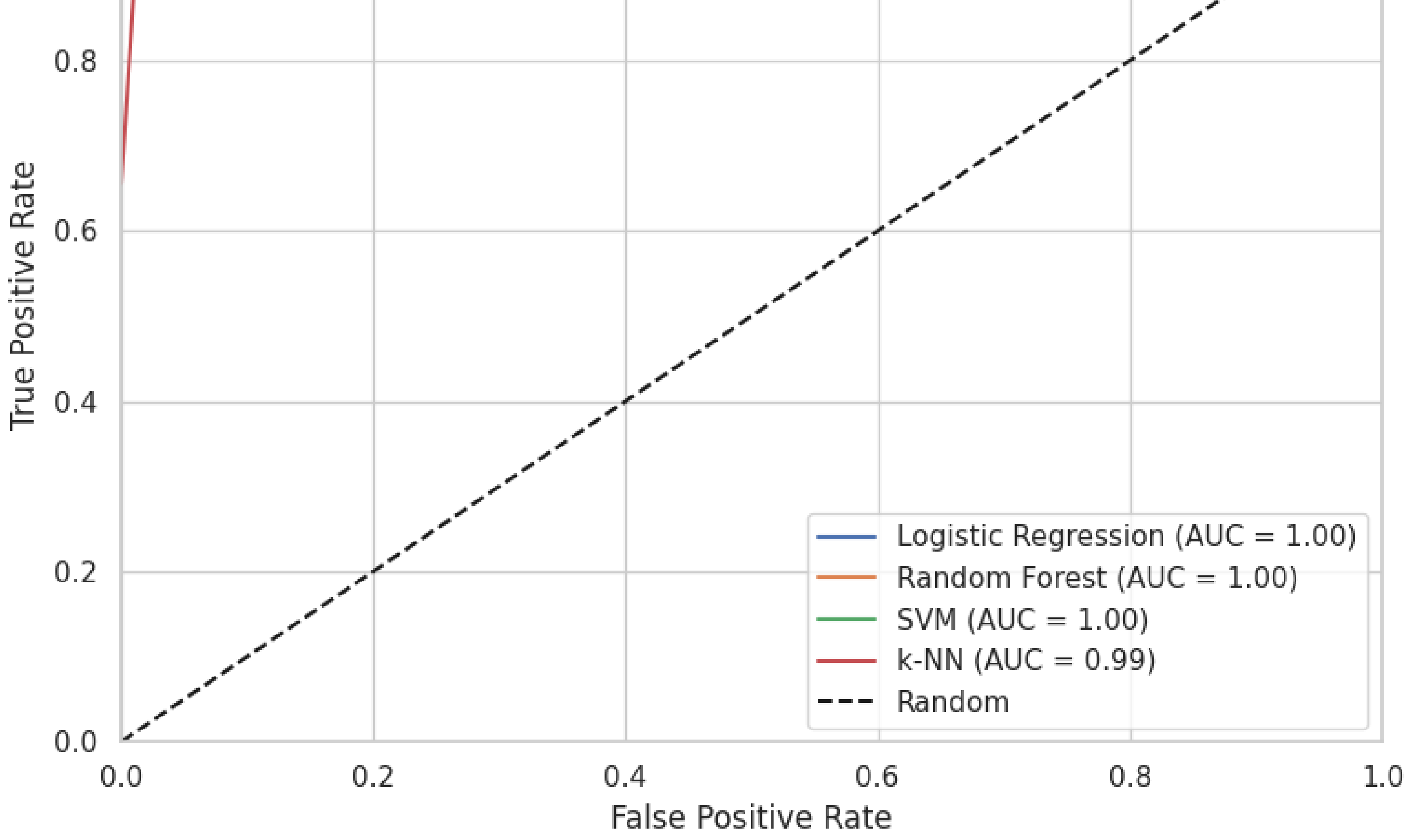
```
# Plot confusion matrices for each model
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()

for idx, name in enumerate(models):
    sns.heatmap(results[name]["conf_matrix"], annot=True, fmt="d", cmap="Blues", ax=
axes[idx])
    axes[idx].set_title(f"{name} Confusion Matrix")
    axes[idx].set_xlabel("Predicted")
    axes[idx].set_ylabel("Actual")

plt.suptitle("Confusion Matrices", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



In [48]:

```python
from sklearn.preprocessing import label_binarize

# Binarize the output for multiclass ROC
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_bin.shape[1]

# Prepare plot
plt.figure(figsize=(8, 6))

# Plot ROC for each model
for name, model in models.items():
    if hasattr(model, "predict_proba"):
        y_score = model.predict_proba(X_test)
    elif hasattr(model, "decision_function"):
        y_score = model.decision_function(X_test)
        # If 1D, reshape to 2D
        if y_score.ndim == 1:
            y_score = y_score.reshape(-1, 1)
    else:
        print(f"{name} does not support probability/decision score; skipping ROC.")
        continue

    if n_classes == 1:
        print(f"{name}: Only one class present; skipping.")
        continue

    # Compute ROC curve and AUC for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Average macro AUC
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
    mean_tpr /= n_classes

    roc_auc["macro"] = auc(all_fpr, mean_tpr)

    # Plot macro-average ROC
    plt.plot(all_fpr, mean_tpr, label=f"{name} (AUC = {roc_auc['macro']:.2f})")

# Plot settings
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for All Models')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```





Overall results show that different types of training models of the second dataset have between 0.98-1.00 accuracy, highest being 1.00 usign Logistic Regression and Random Forest as a the base model. The result was unexpected and the accuracy level of 1.00 indicates an error most of the times. Since the dataset used can be considered small, a certain column could have had an effect on the output. This could be checked using correlation matrix.

RESULTS

This concludes the comparison between the datasets. As a result, the second dataset achieved far better solutions from my expectations. The main reason is most likely the lack of data in the dataset. If both datasets had more data to process on, the results would differ as well. Nevertheless, the outputs show that the second dataset is better for lung cancer detection.

```
In [48]:
```