📊 Statistics Analysis Project

Author: Nikhilesh Bhattacharyya

📌 Introduction

This project looks at a dummy (synthetic) dataset to understand it better using simple statistics. We try to find patterns and connections between things like income, education, health, and habits. By using tools like averages, chances (probabilities), confidence ranges, and testing, we find useful information about people's age, jobs, studies, and lifestyle.

📁 Dataset Description

- **Source**: `synthetic_dataset.csv`
- **Total Rows**: 1000
- **Key Columns**:
  - `Gender`, `Age`, `Height`, `Weight`, `BMI`
  - `Income`, `EducationLevel`, `Occupation`, `SmokingStatus`
  - `Region`, `MaritalStatus`, `YearsOfExperience`, `JobSatisfaction`

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("synthetic_dataset.csv")
df
```

Out[2]:

	ID	Age	DateOfBirth	Gender	MaritalStatus	EducationLevel	Occupation	
0	1	47	1978-01-01	Male	Single	Bachelor's	Teacher	388
1	2	37	1988-01-01	Male	Single	High School	Other	1094
2	3	49	1976-01-01	Female	Single	Bachelor's	Teacher	520
3	4	62	1963-01-01	Female	Married	Master's	Teacher	376
4	5	36	1989-01-01	Male	Divorced	Bachelor's	Other	947
...	...	...	...	...	...	...	...	
995	996	35	1990-01-01	Male	Divorced	High School	Doctor	1740
996	997	66	1959-01-01	Male	Married	Bachelor's	Other	243
997	998	49	1976-01-01	Female	Married	PhD	Doctor	256
998	999	31	1994-01-01	Male	Widowed	Bachelor's	Teacher	909
999	1000	48	1977-01-01	Male	Single	High School	Other	207

1000 rows × 20 columns

Find the range, variance, and standard deviation of the Income column.

In [4]:
```
mean_income=df["Income"].mean()
range_income=df["Income"].max()-df["Income"].min()
variance_income=df["Income"].var(ddof=1)
std_income=df["Income"].std(ddof=1)
median_income=df["Income"].median()

mean_income
```

Out[4]: 47396.09051681785

In [5]:
```
range_income
```

Out[5]: 180000.0

In [6]:
```
variance_income
```

Out[6]: 1750914316.3087862

In [7]:
```
std_income
```

Out[7]: 41843.92806977837

In [8]:
```
median_income
```

Out[8]: 29689.30932693922

Mean Income: ₹47,396.09 ➤ On average, income is around ₹47,000. ➤ The average is affected by some very high income values (outliers). Median Income: ₹29,689.31 ➤ Half of the people earn less than ₹29,689, and half earn more. ➤ Median is more reliable when data is unevenly spread. Range: ₹1,80,000 ➤ Difference between highest and lowest income. ➤ Indicates a wide gap in income levels. Standard Deviation: ₹41,843.93 ➤ Tells how much incomes vary from the average. ➤ A high number means income values are widely spread. Skewness (Shape of Data): Right-skewed ➤ Since mean is greater than median, data is pulled toward higher values. ➤ Most people earn on the lower side, with a few earning much more

Frequency Distribution Table for the EducationLevel Column

In [10]:
```
df["EducationLevel"].value_counts()
```

Out[10]:
```
EducationLevel
Bachelor's     418
High School    305
Master's       189
PhD             88
Name: count, dtype: int64
```

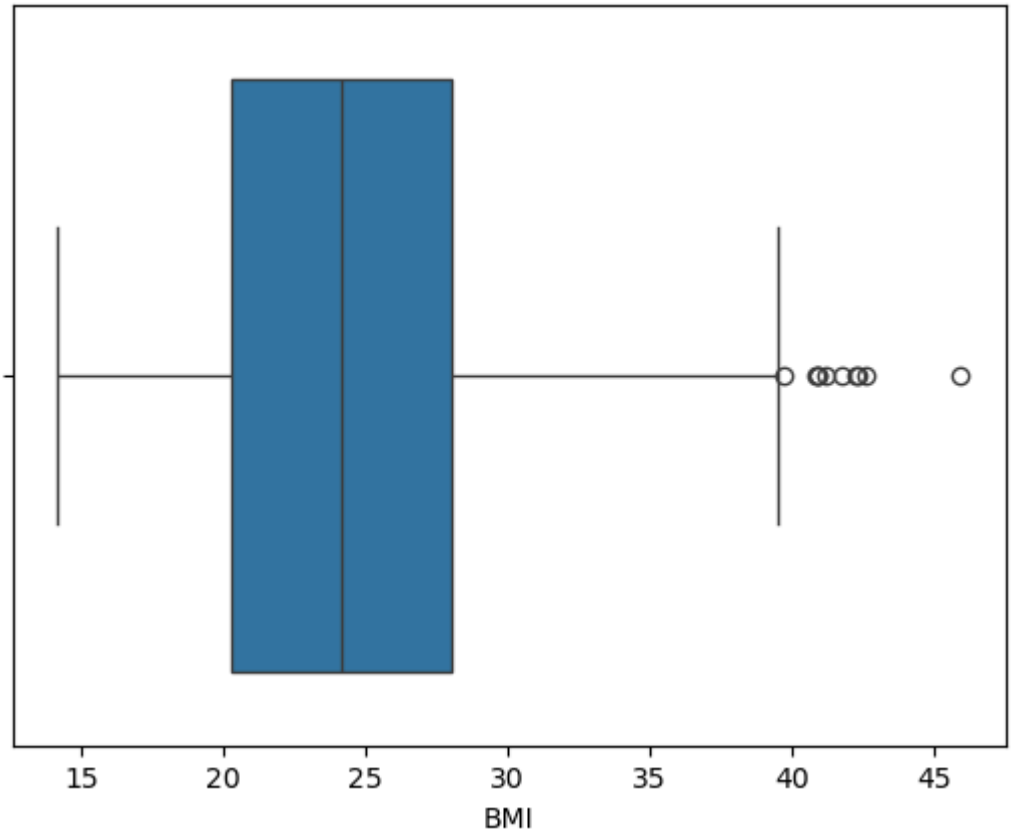
Interquartile Range (IQR) for BMI

In [12]:
```
Q1 =df["BMI"].quantile(0.25)
Q3= df["BMI"].quantile(0.75)
IQR_BMI=Q3-Q1
IQR_BMI
```

Out[12]:
```
7.708988520179258
```

In [13]:
```
sns.boxplot(data=df,x="BMI")
```

Out[13]:
```
<Axes: xlabel='BMI'>
```



Correlation between Years Of Experience and Income

In [15]:
```
corr=df['YearsOfExperience'].corr(df['Income'])
corr
```

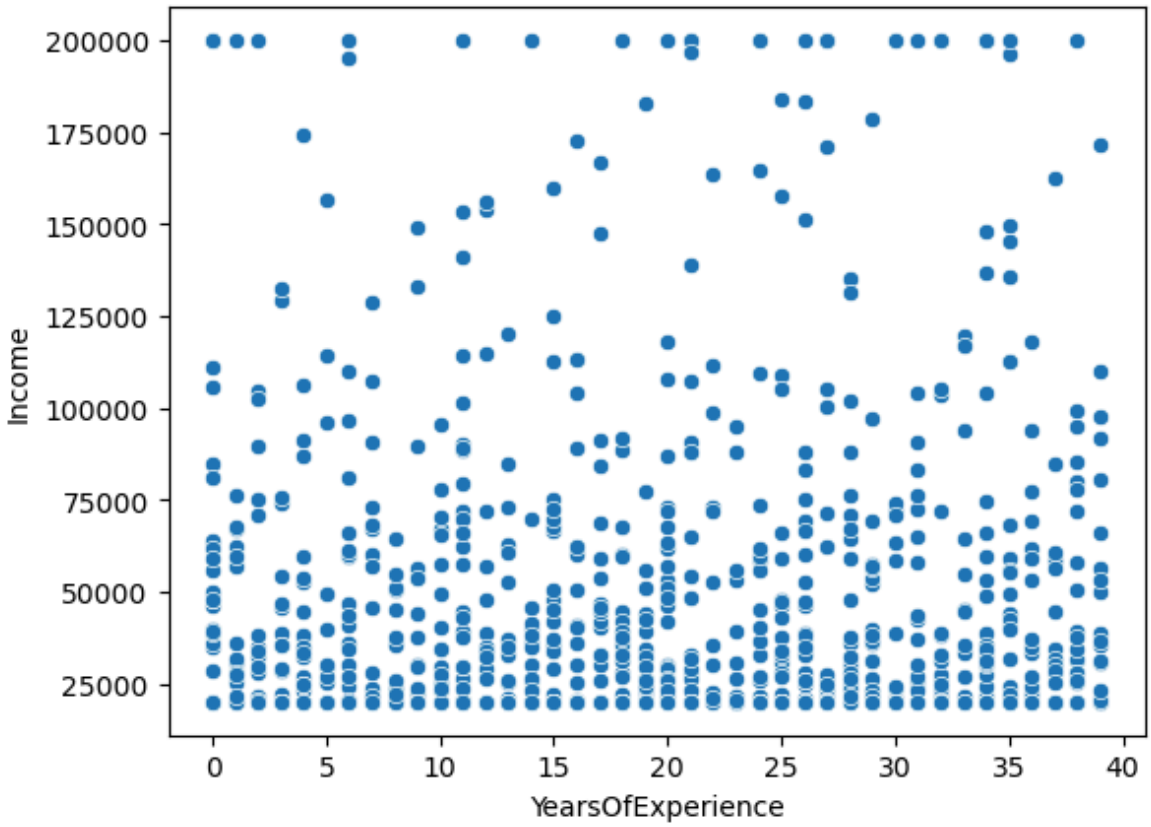
Out[15]:
```
0.035040049760047924
```

In [16]:
```
corr.round(2)
```

Out[16]:
```
0.04
```

In [17]:
```
sns.scatterplot(x="YearsOfExperience", y='Income', data=df)
```

Out[17]:
```
<Axes: xlabel='YearsOfExperience', ylabel='Income'>
```



almost no correlation between Years Of Experience and Income

some people are newely freshers earning more then a experience

---

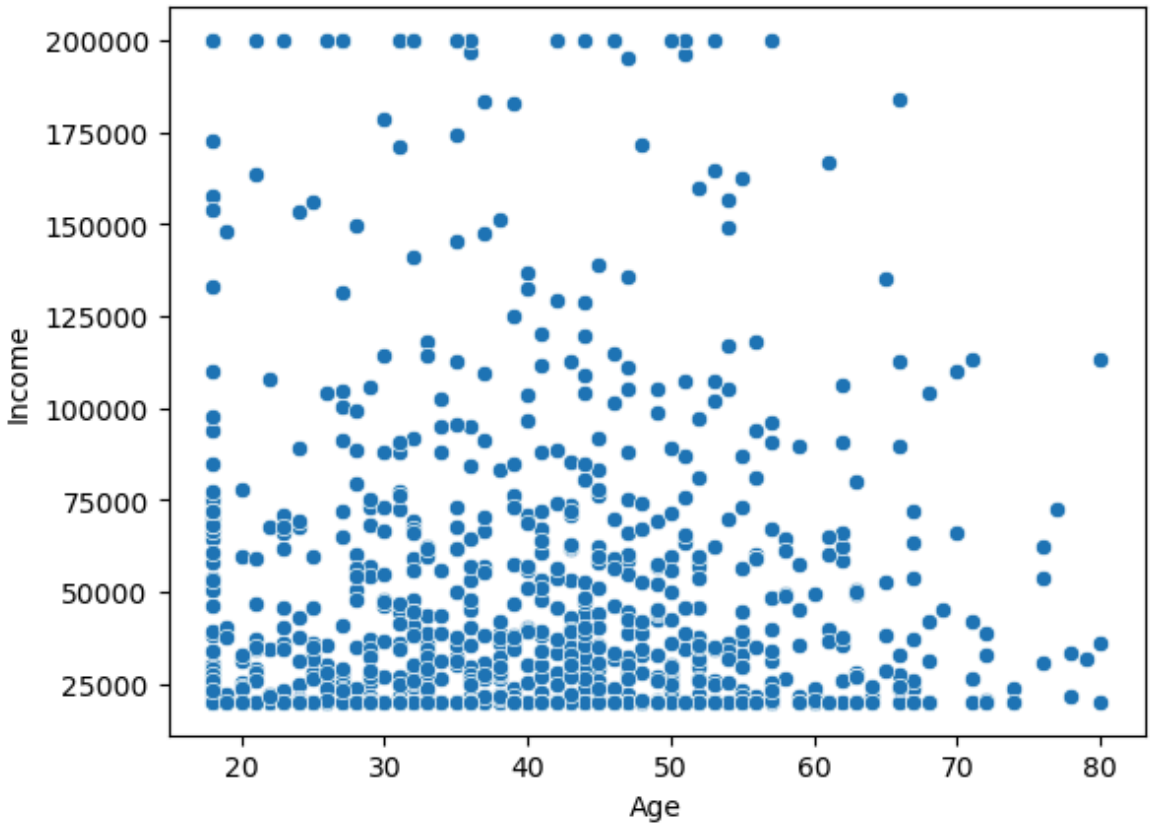
What is the correlation between age and income?

In [20]:
```
corr=df["Income"].corr(df["Age"])
corr
```

Out[20]: -0.041461037069182916

In [21]:
```
sns.scatterplot(x="Age",y="Income", data=df)
```

Out[21]: <Axes: xlabel='Age', ylabel='Income'>



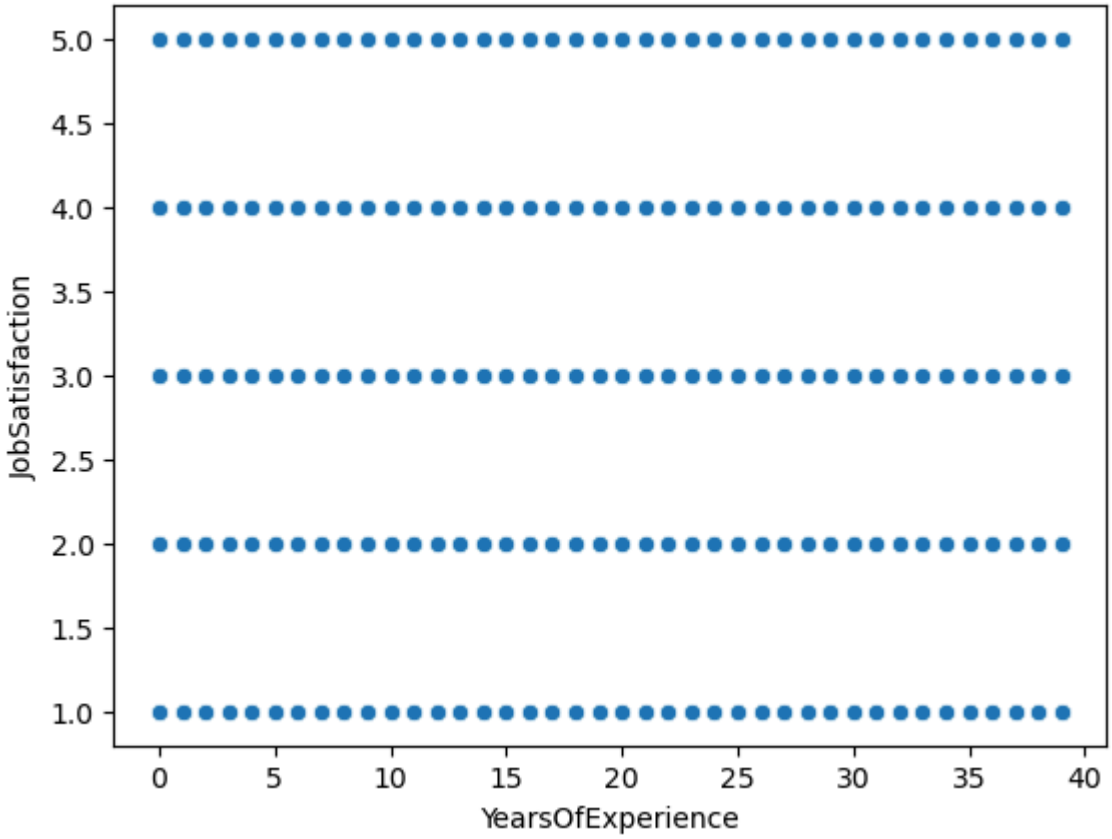
Is there a correlation between years of experience and job satisfaction?

In [23]:
```
df["YearsOfExperience"].corr(df["JobSatisfaction"])
```

Out[23]:
```
-0.015634486003166224
```

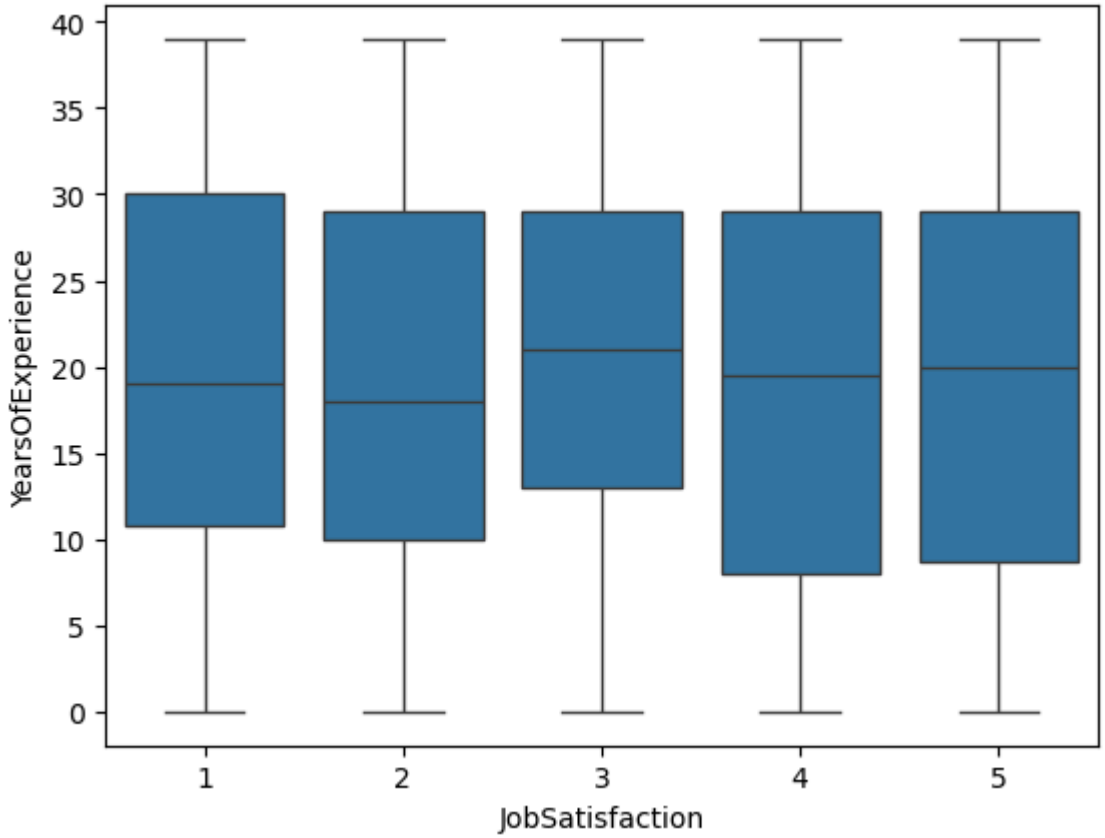
In [24]:
```
sns.scatterplot(x="YearsOfExperience" , y="JobSatisfaction" , data=df)
```

Out[24]:
```
<Axes: xlabel='YearsOfExperience', ylabel='JobSatisfaction'>
```



In [25]:
```
sns.boxplot(x="JobSatisfaction" , y="YearsOfExperience", data =df)
```

Out[25]: <Axes: xlabel='JobSatisfaction', ylabel='YearsOfExperience'>



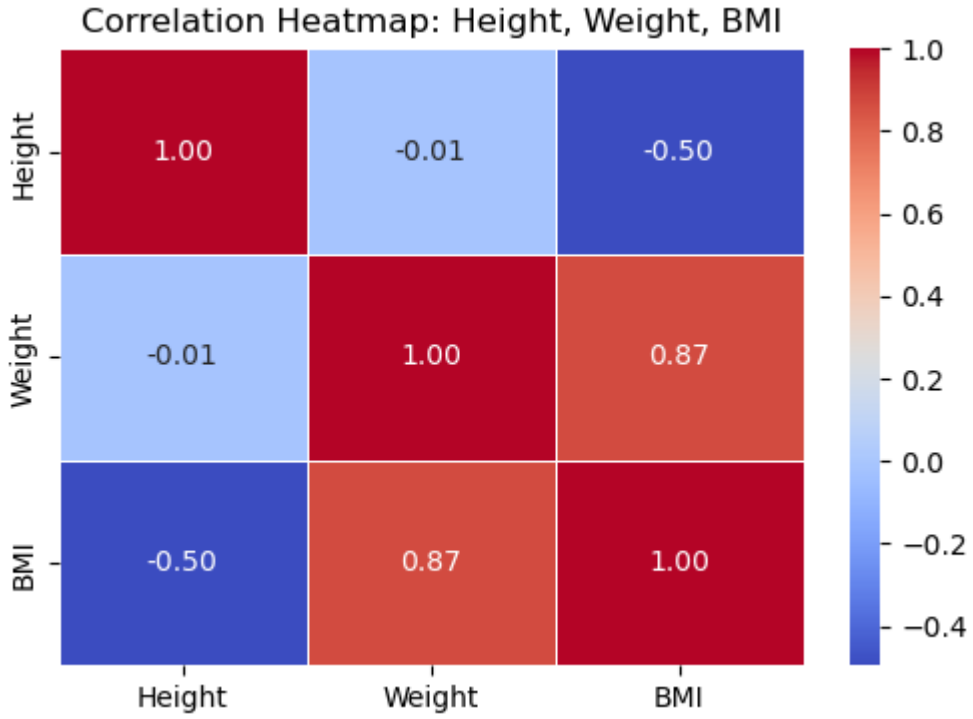
What is the correlation matrix for height, weight, and BMI?

In [28]:
```
corr_matrix = df[["Height", "Weight", "BMI"]].corr()
print("Correlation Matrix for Height, Weight, BMI:")
print(corr_matrix)
```

```
Correlation Matrix for Height, Weight, BMI:
          Height    Weight       BMI
Height  1.000000 -0.013521 -0.495645
Weight -0.013521  1.000000  0.868848
BMI    -0.495645  0.868848  1.000000
```

In [154…
```
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Calculate correlation matrix
corr_matrix = df[["Height", "Weight", "BMI"]].corr()

print("Correlation Matrix for Height, Weight, BMI:")
print(corr_matrix)

# Step 2: Plot the heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", linewidths=0.5, fmt=".2f")
plt.title("Correlation Heatmap: Height, Weight, BMI")
plt.show()
```

```
Correlation Matrix for Height, Weight, BMI:
          Height    Weight       BMI
Height  1.000000 -0.013521 -0.495645
Weight -0.013521  1.000000  0.868848
BMI    -0.495645  0.868848  1.000000
```



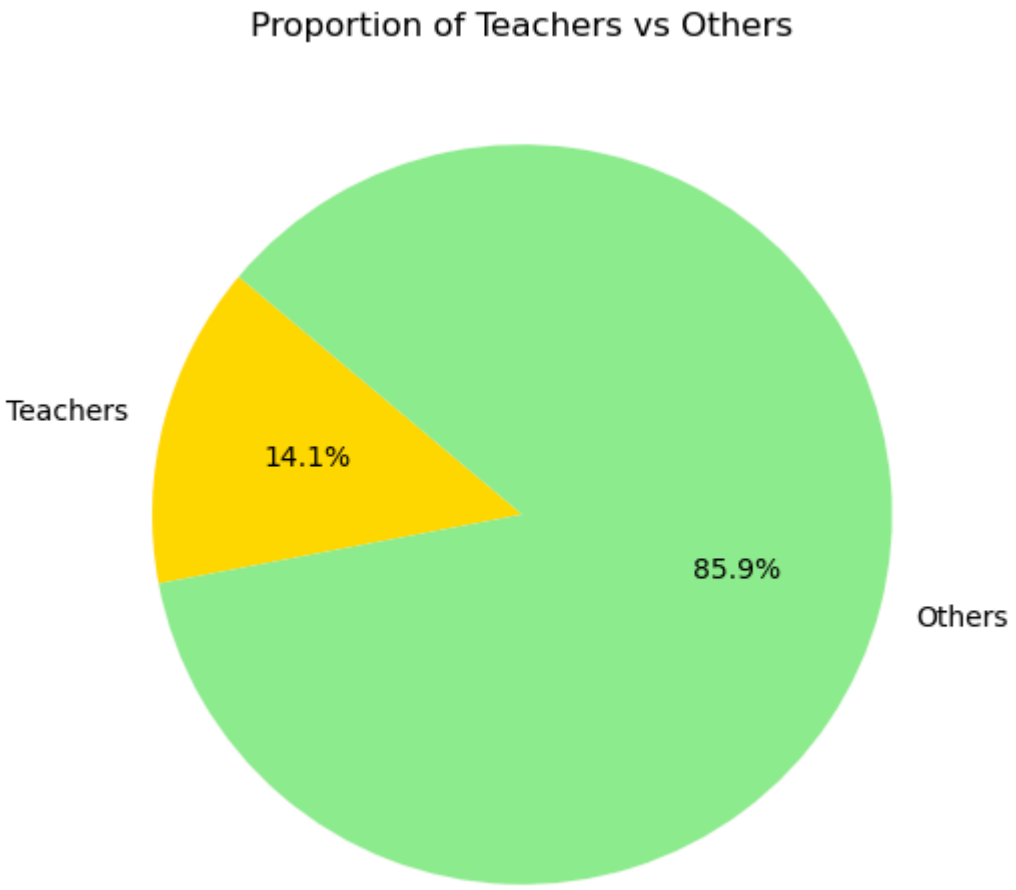
Probability of Being a Teacher

In [149…
```
total=len(df)
teacher=len(df[df["Occupation"]=="Teacher"])
probality=(teacher/total)*100
print("Probability of Being a Teacher:")
round(probality,2)
```

```
Probability of Being a Teacher:
```

Out[149… 14.1

In [151…
```
import matplotlib.pyplot as plt

non_teacher = total - teacher
labels = ['Teachers', 'Others']
counts = [teacher, non_teacher]
colors = ['#FFD700', '#90EE90']

plt.figure(figsize=(6,6))
plt.pie(counts, labels=labels, autopct='%1.1f%%', colors=colors, startangle=140)
plt.title('Proportion of Teachers vs Others')
plt.show()
```

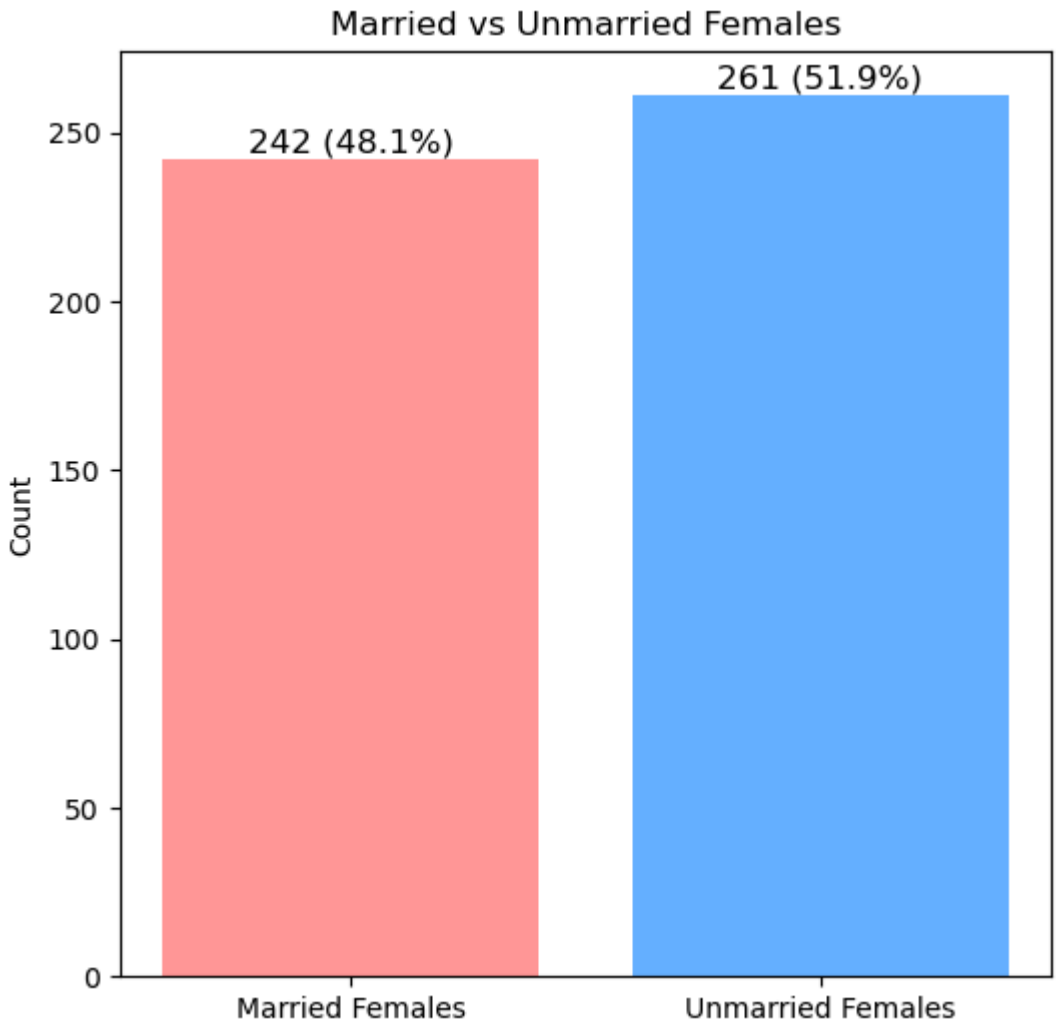


What is the probability that a female is married?

In [145…
```
f = len(df[(df["Gender"] == "Female") & (df["MaritalStatus"] == "Married")])
m = len(df[df["Gender"] == "Female"])
probality = f / m
print(f"probability that a female is married:  {round(probality *100,2)}%")
```

```
probability that a female is married:  48.11%
```

In [147…
```
import matplotlib.pyplot as plt

labels = ['Married Females', 'Unmarried Females']
values = [f, m - f]

colors = ['#ff9999','#66b3ff']

plt.figure(figsize=(6,6))
plt.bar(labels, values, color=colors)

# Add count and %
for i, v in enumerate(values):
    percent = v / m * 100
    plt.text(i, v + 2, f"{v} ({percent:.1f}%)", ha='center', fontsize=12)

plt.title("Married vs Unmarried Females")
plt.ylabel("Count")
plt.show()
```



What is the probability that someone has a bachelor's degree or is single?

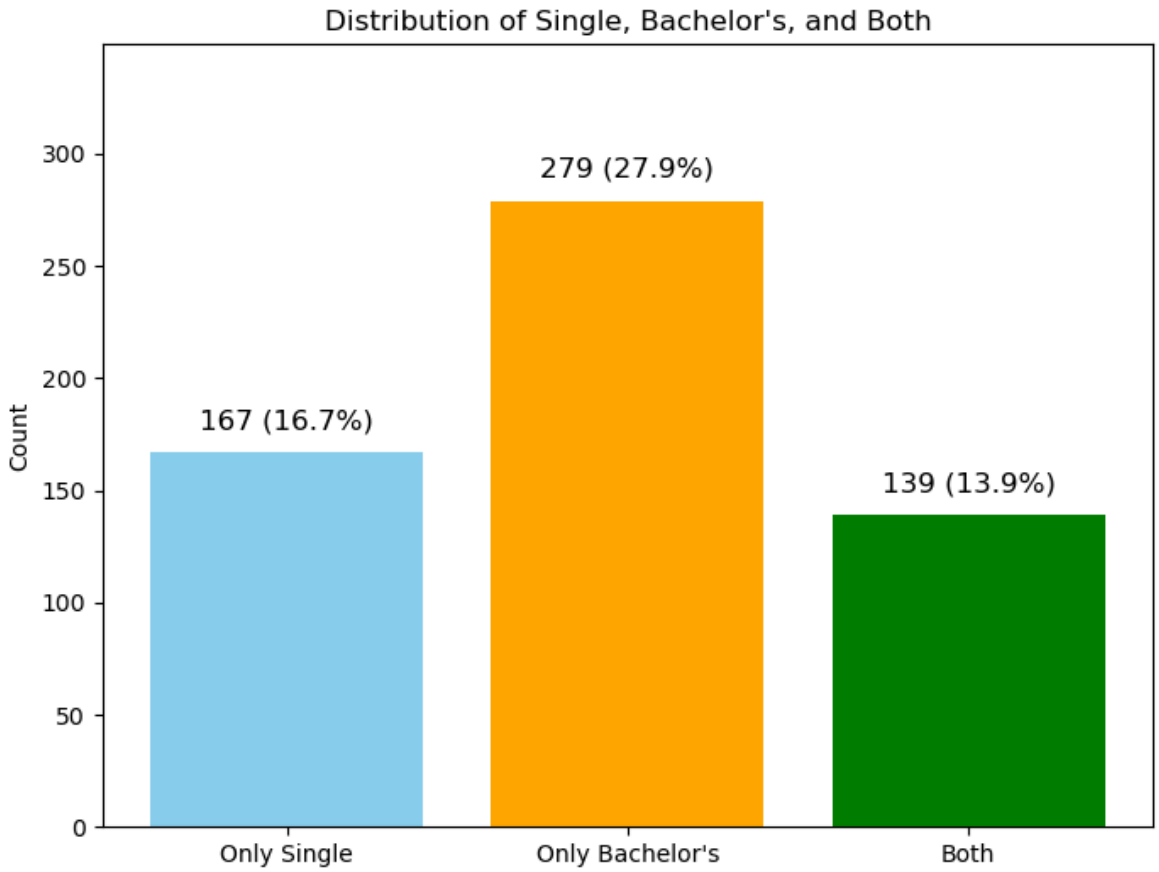
In [143…

```
n = len(df)

A = len(df[df["MaritalStatus"] == "Single"])
B = len(df[df["EducationLevel"] == "Bachelor's"])
A_and_B = len(df[(df["MaritalStatus"] == "Single") & (df["EducationLevel"] == "B

prob = (A + B - A_and_B) / n

print(f"P(Single or Bachelor's): {round(prob * 100, 2)}%")
```

```
P(Single or Bachelor's): 58.5%
```

In [141…

```
import matplotlib.pyplot as plt

n = len(df)

# Counts
A = len(df[df["MaritalStatus"] == "Single"])
B = len(df[df["EducationLevel"] == "Bachelor's"])
A_and_B = len(df[(df["MaritalStatus"] == "Single") & (df["EducationLevel"] == "B

only_A = A - A_and_B
only_B = B - A_and_B
both = A_and_B

labels = ['Only Single', "Only Bachelor's", 'Both']
counts = [only_A, only_B, both]

# Bar Plot
plt.figure(figsize=(8,6))
colors = ['skyblue', 'orange', 'green']
bars = plt.bar(labels, counts, color=colors)

# Set y-axis limit dynamically
plt.ylim(0, max(counts) * 1.25)

# Add count + percentage on top of bars
for bar, count in zip(bars, counts):
    percentage = (count / n) * 100
    label = f"{count} ({percentage:.1f}%)"
    plt.text(bar.get_x() + bar.get_width()/2,
             count + max(counts) * 0.03,
             label,
             ha='center',
             va='bottom',
             fontsize=12)

plt.title("Distribution of Single, Bachelor's, and Both")
plt.ylabel('Count')
plt.show()
```



Are EducationLevel and Occupation Independent?

In [36]:
```
# independent event

total_people = len(df)

p_bachelors = len(df[df["EducationLevel"] == "Bachelor's"]) / total_people
p_teacher = len(df[df["Occupation"] == "Teacher"]) / total_people
p_bachelors_and_teacher = len(df[(df["EducationLevel"] == "Bachelor's") & (df["C


if abs(p_bachelors_and_teacher - (p_bachelors * p_teacher)) < 0.01:
    print("EducationLevel and Occupation are **independent**.")
else:
    print("EducationLevel and Occupation are **dependent**.")
```

```
EducationLevel and Occupation are **independent**.
```

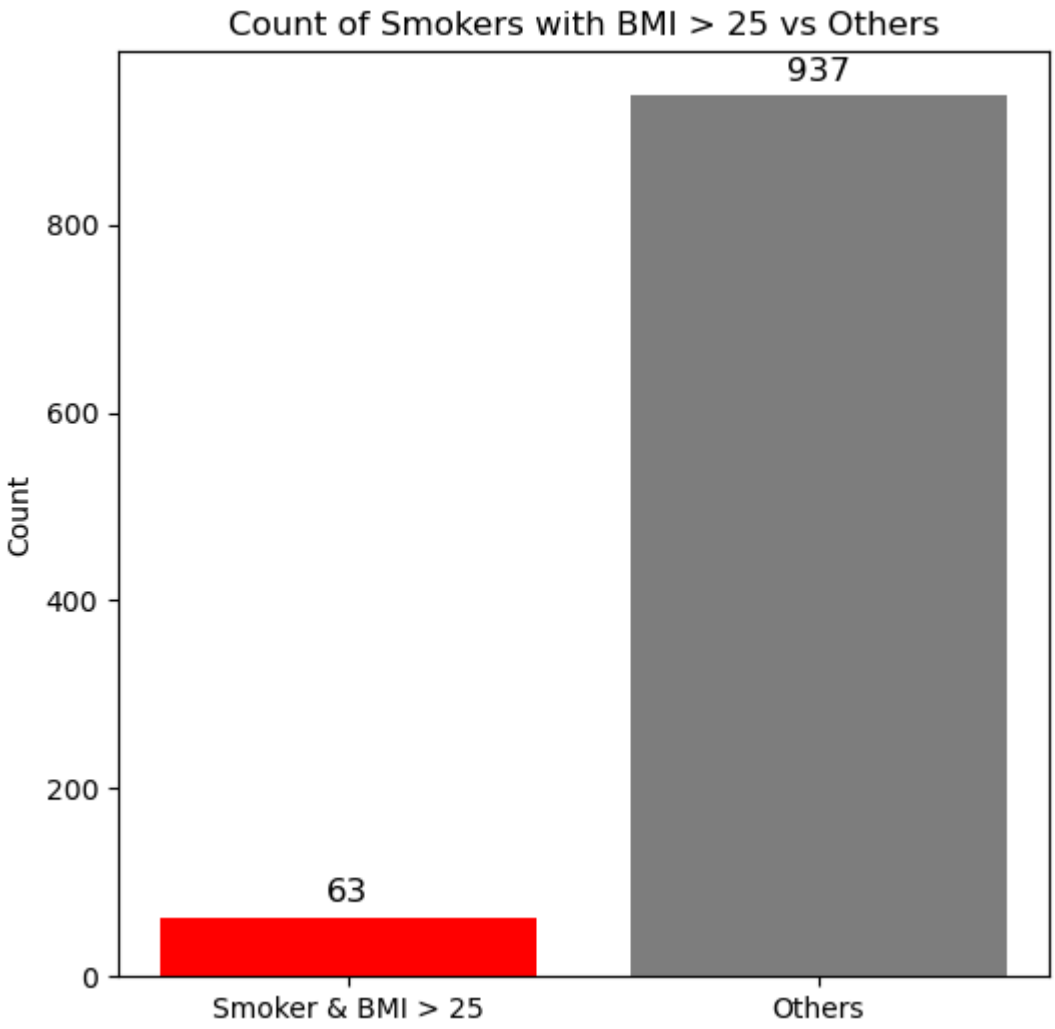
Probability of Smoker and BMI > 25

In [38]:
```
# let x be the person who smokes and above the bmi 25
T=len(df)
X=len(df[(df["SmokingStatus"]=="Yes") & (df["BMI"]>25)])
probality=X/T*100
print("Probability of Smoker and BMI > 25","    " , probality,"%")
```

```
Probability of Smoker and BMI > 25      6.3 %
```

In [117…

```
import matplotlib.pyplot as plt

T = len(df)
X = len(df[(df["SmokingStatus"] == "Yes") & (df["BMI"] > 25)])
others = T - X

labels = ['Smoker & BMI > 25', 'Others']
counts = [X, others]

plt.figure(figsize=(6,6))
colors = ['red', 'gray']
plt.bar(labels, counts, color=colors)

for i, count in enumerate(counts):
    plt.text(i, count + T*0.01, str(count), ha='center', va='bottom', fontsize=1

plt.title('Count of Smokers with BMI > 25 vs Others')
plt.ylabel('Count')
plt.show()
```



Are the events SmokingStatus = "Yes" and SmokingStatus = "No" disjoint?

What is the probability that an individual is both a smoker and a non-smoker?

In [40]:
```
print("yes its a disjoint and the probolity is zero","because  ","disjoint event
```

```
yes its a disjoint and the probolity is zero because   disjoint events cannot  be
happend at same time
```

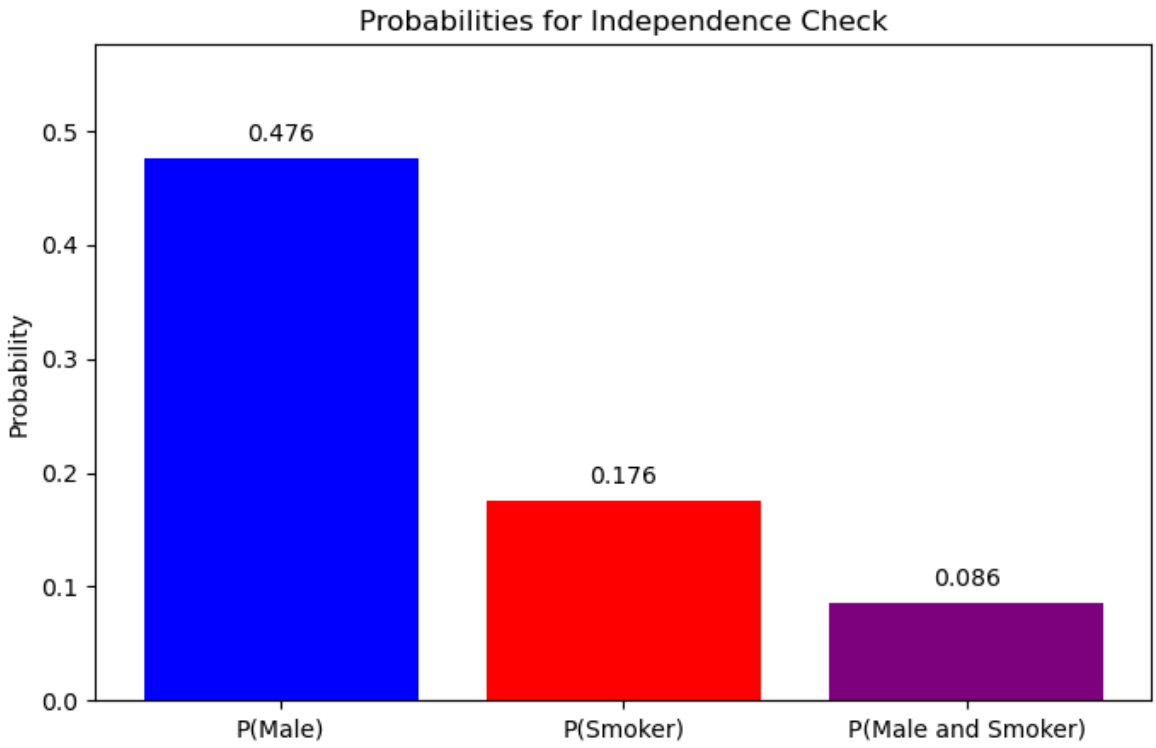
Are the events Gender = ”Male” and SmokingStatus = ”Yes” independent?

Calculate P(Male), P(Smoker), and P(Male and Smoker).

In [42]:
```
import pandas as pd

df = pd.read_csv('synthetic_dataset.csv')
total = len(df)

p_male = len(df[df['Gender'] == 'Male']) / total
p_smoker = len(df[df['SmokingStatus'] == 'Yes']) / total
p_male_and_smoker = len(df[(df['Gender'] == 'Male') & (df['SmokingStatus'] == 'Y

independence_check = abs(p_male_and_smoker - (p_male * p_smoker)) < 0.0001

print(f"P(Male): {p_male:.4f}, P(Smoker): {p_smoker:.4f}, P(Male and Smoker): {p
print(f"Are they independent? {independence_check}")
```

```
P(Male): 0.4760, P(Smoker): 0.1760, P(Male and Smoker): 0.0860
Are they independent? False
```

In [115…
```
import matplotlib.pyplot as plt

probs = [p_male, p_smoker, p_male_and_smoker]
labels = ['P(Male)', 'P(Smoker)', 'P(Male and Smoker)']

plt.figure(figsize=(8,5))
bars = plt.bar(labels, probs, color=['blue', 'red', 'purple'])

# Probability values upar show karna
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.01, f"{height:.3f}", ha

plt.ylim(0, max(probs) + 0.1)
plt.title('Probabilities for Independence Check')
plt.ylabel('Probability')
plt.show()
```

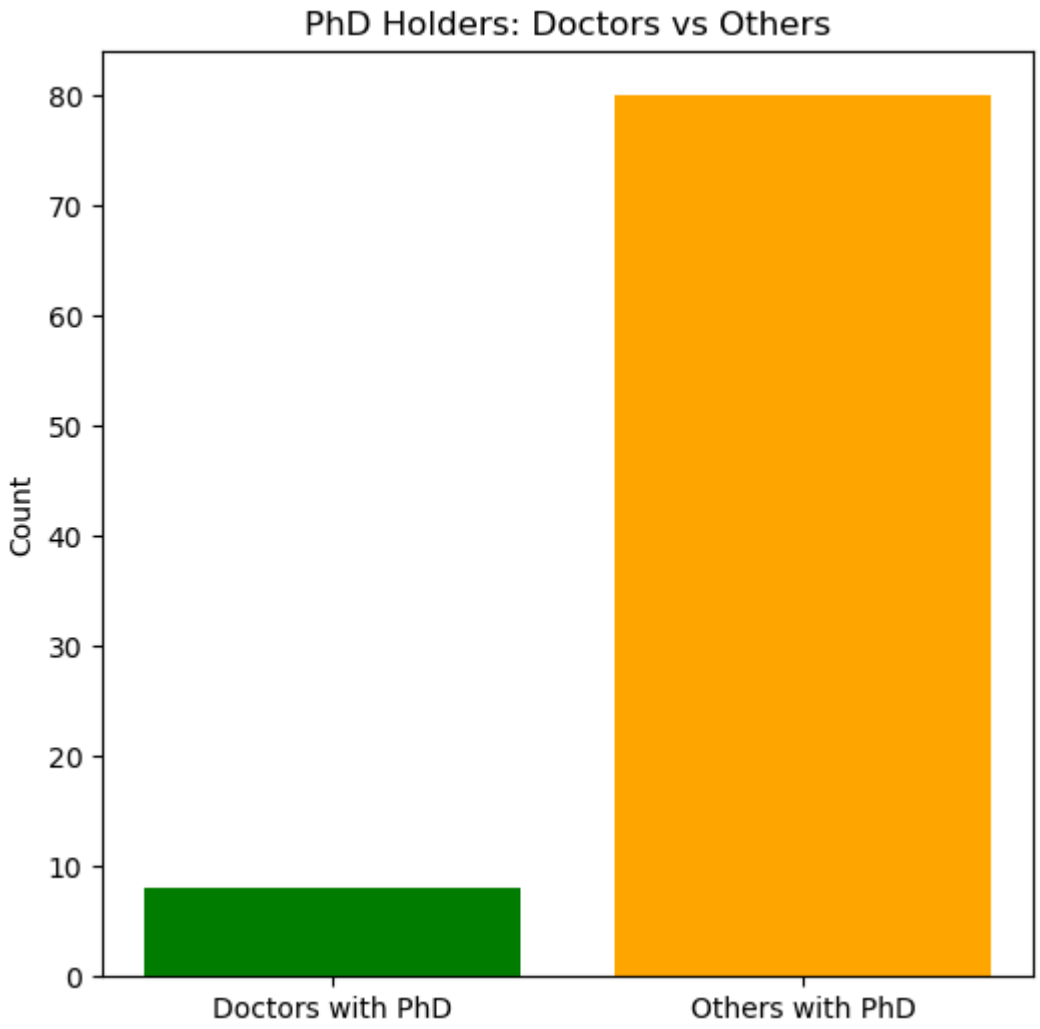


What is the probability someone is a doctor given they have a PhD?

In [44]:
```
phds = df[df['EducationLevel'] == 'PhD']  # Filter only PhD holders

doctors_with_phd = len(phds[phds['Occupation'] == 'Doctor'])  # From PhDs, count

prob_doctor_given_phd = doctors_with_phd / len(phds)  # Conditional probability

print(f"P(Doctor | PhD): {prob_doctor_given_phd*100}%")
```

```
P(Doctor | PhD): 9.090909090909092%
```

In [113…
```
import matplotlib.pyplot as plt

# Data for plot
total_phd = len(phds)
doctors_phd = len(phds[phds['Occupation'] == 'Doctor'])
non_doctors_phd = total_phd - doctors_phd

# Labels and counts
labels = ['Doctors with PhD', 'Others with PhD']
counts = [doctors_phd, non_doctors_phd]

# Plot
plt.figure(figsize=(6,6))
plt.bar(labels, counts, color=['green', 'orange'])
plt.title('PhD Holders: Doctors vs Others')
plt.ylabel('Count')
plt.show()
```



P(Doctor I PhD)= PhD Doctor AND PhD/PhD

What is the 95% confidence interval for the average BMI?

In [79]:
```
import numpy as np
from scipy.stats import t #t-distribution

n = len(df)
mean_bmi = df['BMI'].mean()
std_bmi = df['BMI'].std(ddof=1)
se_bmi = std_bmi / np.sqrt(n)
t_critical = t.ppf(0.975, df=n-1)

#Margin of Error
margin_of_error = t_critical * se_bmi

#Confidence Interval
ci_lower = mean_bmi - margin_of_error
ci_upper = mean_bmi + margin_of_error

print(f"95% Confidence Interval for BMI: ({ci_lower:.2f}, {ci_upper:.2f})")
```

```
95% Confidence Interval for BMI: (24.17, 24.86)
```

95% Confidence Interval for Proportion of Smokers

In [82]:
```
from statsmodels.stats.proportion import proportion_confint

# Count how many smokers
smokers = len(df[df['SmokingStatus'] == 'Yes'])

# Total people in the dataset
total_people = len(df)

# Calculate 95% confidence interval for proportion of smokers
ci_smokers = proportion_confint(count=smokers, nobs=total_people, alpha=0.05, me

print(f"95% CI for Proportion of Smokers: ({ci_smokers[0]:.3f}, {ci_smokers[1]:.
```

```
95% CI for Proportion of Smokers: (0.152, 0.200)
```

to numeric conversion

In [92]:
```
import pandas as pd
import numpy as np
from scipy.stats import t

# Step 1: Convert 'Yes'/'No' to 1/0
df['Smoking_numeric'] = df['SmokingStatus'].apply(lambda x: 1 if x == 'Yes' else

# Step 2: Drop missing values if any (optional, safety ke liye)
df = df.dropna(subset=['Smoking_numeric'])

# Step 3: Sample size
n = len(df)

# Step 4: Mean and Standard Deviation
mean_smoke = df['Smoking_numeric'].mean()
std_smoke = df['Smoking_numeric'].std(ddof=1)

# Step 5: Standard Error
se_smoke = std_smoke / np.sqrt(n)

# Step 6: t-critical value for 95%
t_critical = t.ppf(0.975, df=n-1)

# Step 7: Margin of Error
margin_of_error = t_critical * se_smoke

# Step 8: Confidence Interval
ci_lower = mean_smoke - margin_of_error
ci_upper = mean_smoke + margin_of_error

# Step 9: Print the result
print(f"95% Confidence Interval for Proportion of Smokers (t-distribution): ({ci
```

```
95% Confidence Interval for Proportion of Smokers (t-distribution): (0.1524, 0.19
96)
```

Two-Sample T-Test for Income by Gender

In [99]:
```
from scipy.stats import ttest_ind

# Step 1: Divide income by gender
male_income = df[df['Gender'] == 'Male']['Income']
female_income = df[df['Gender'] == 'Female']['Income']

# Step 2: Perform t-test (Welch's t-test, unequal variance)
t_stat_gender, p_value_gender = ttest_ind(male_income, female_income, equal_var=

# Step 3: Print results
print(f"T-Statistic for Income by Gender: {t_stat_gender:.4f}, P-Value: {p_value

# Step 4: Hypothesis test interpretation
if p_value_gender < 0.05:
    print("Reject H0: Income differs between Male and Female.")
else:
    print("Fail to reject H0: No significant difference in Income between gender
```

```
T-Statistic for Income by Gender: -0.4822, P-Value: 0.6298
Fail to reject H0: No significant difference in Income between genders.
```

In [ ]:

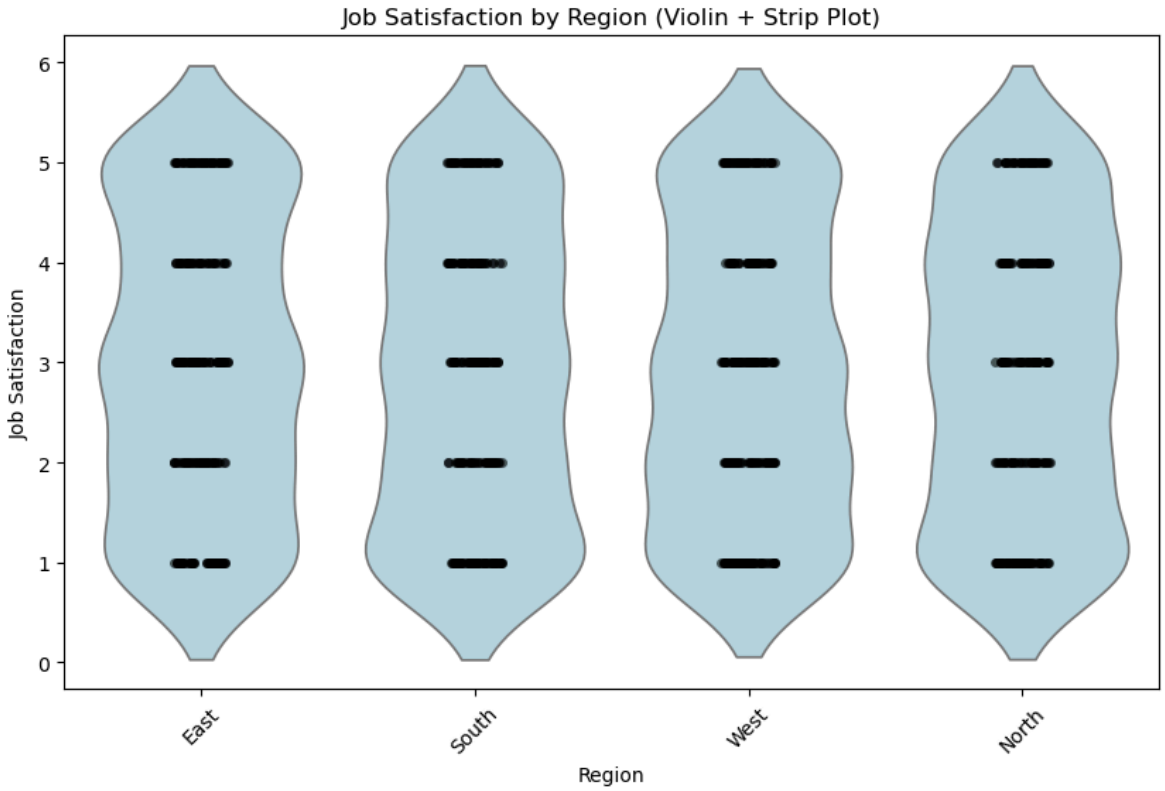
ANOVA for JobSatisfaction across Regions

In [109…
```
from scipy.stats import f_oneway

# Step 1: Get unique regions
regions = df['Region'].unique()

# Step 2: Create list of JobSatisfaction groups by region
groups = [df[df['Region'] == region]['JobSatisfaction'] for region in regions]

# Step 3: Perform one-way ANOVA
f_stat, p_value_anova = f_oneway(*groups)

# Step 4: Print results
print(f"F-Statistic for JobSatisfaction: {f_stat:.4f}, P-Value: {p_value_anova:.

# Step 5: Hypothesis test decision
if p_value_anova < 0.05:
    print("Reject H0: JobSatisfaction differs across Regions.")
else:
    print("Fail to reject H0: No significant difference in JobSatisfaction acros
```

```
F-Statistic for JobSatisfaction: 0.2680, P-Value: 0.8485
Fail to reject H0: No significant difference in JobSatisfaction across Regions.
```

In [111…
```
plt.figure(figsize=(10,6))
sns.violinplot(x='Region', y='JobSatisfaction', data=df, inner=None, color='ligh
sns.stripplot(x='Region', y='JobSatisfaction', data=df, jitter=True, color='blac
plt.title('Job Satisfaction by Region (Violin + Strip Plot)')
```

```
plt.xlabel('Region')
plt.ylabel('Job Satisfaction')
plt.xticks(rotation=45)
plt.show()
```



In [ ]: