National University of Technology

Artificial Intelligence Lab-CS3502

MSE Submitted to:

Sir Yosha Jawad

Submitted By:

Muhammad Abdullah

Reg. No: **F22607010**

Batch: **Fall AI 2022**

Question No .1:

```
#Import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('Data.csv')
print(df.head())
```

[7] ✓ 0.0s

```
   rank  finalWorth              category               personName   age  \
0     1      211000       Fashion & Retail  Bernard Arnault & family  74.0
1     2      180000             Automotive                Elon Musk  51.0
2     3      114000             Technology              Jeff Bezos  59.0
3     4      107000             Technology            Larry Ellison  78.0
4     5      106000  Finance & Investments            Warren Buffett  92.0

        country    city             source            industries  \
0        France   Paris               LVMH       Fashion & Retail
1  United States  Austin       Tesla, SpaceX            Automotive
2  United States  Medina             Amazon            Technology
3  United States   Lanai             Oracle            Technology
4  United States   Omaha  Berkshire Hathaway  Finance & Investments
```

```
# Checking for missing values in the entire DataFrame
missing_values = df.isnull().sum()

# Displaying missing values as a percentage of the total values
missing_percentage = (df.isnull().sum() / len(df)) * 100

# Displaying missing values and their percentage in a DataFrame
missing_info = pd.DataFrame({'Missing_Values': missing_values, 'Percentage': missing_percentage})
print(missing_info)
```

✓ 0.0s

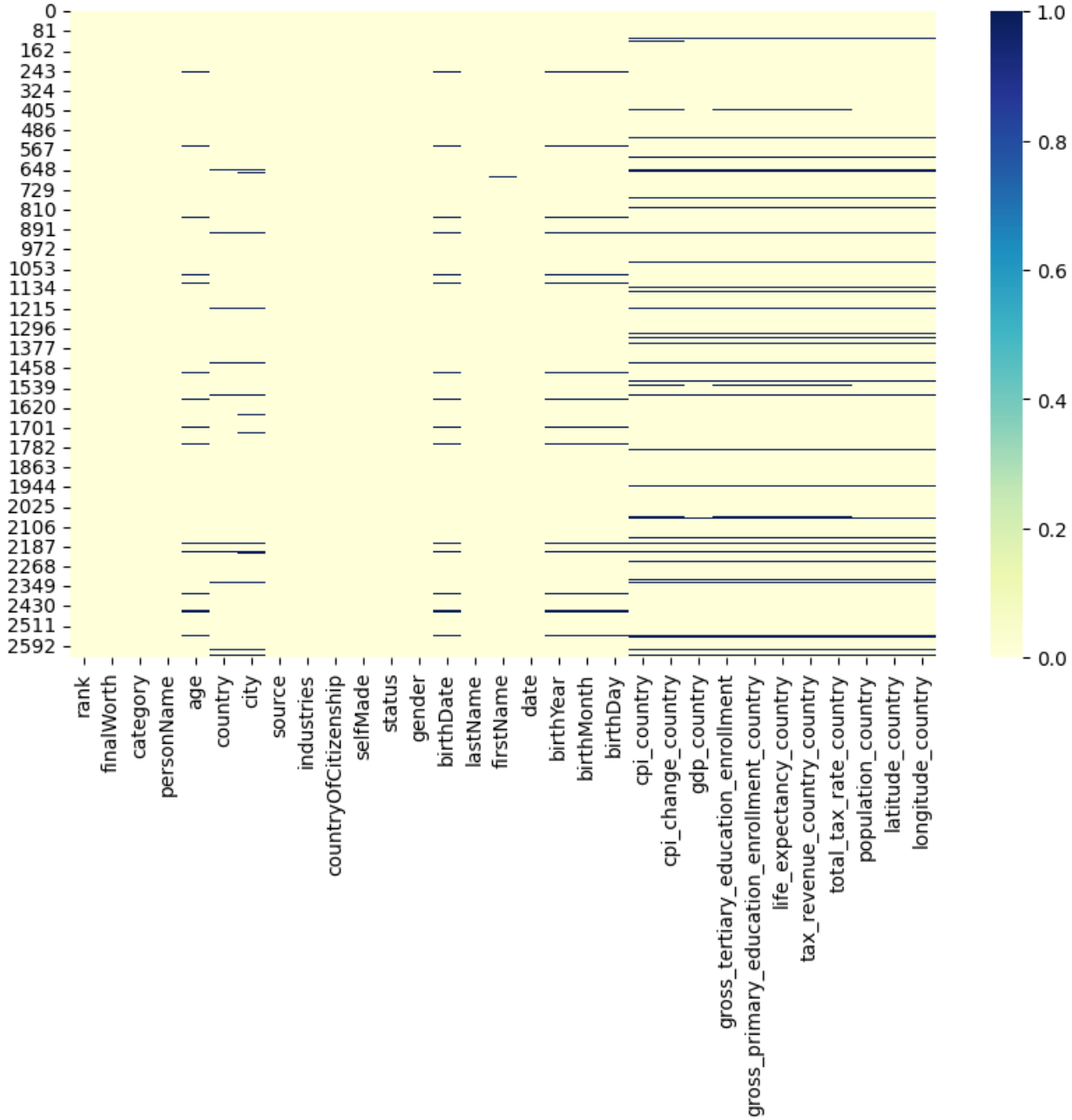
```
                      Missing_Values  Percentage
rank                               0    0.000000
finalWorth                         0    0.000000
category                           0    0.000000
personName                         0    0.000000
age                               65    2.462121
country                           38    1.439394
city                              72    2.727273
source                             0    0.000000
industries                         0    0.000000
countryOfCitizenship               0    0.000000
organization                    2315   87.689394
selfMade                           0    0.000000
status                             0    0.000000
```

city	72	2.727273
source	0	0.000000
industries	0	0.000000
countryOfCitizenship	0	0.000000
organization	2315	87.689394
selfMade	0	0.000000
status	0	0.000000
gender	0	0.000000
birthDate	76	2.878788
lastName	0	0.000000
firstName	3	0.113636
title	2301	87.159091
date	0	0.000000
state	1887	71.477273
residenceStateRegion	1893	71.704545
birthYear	76	2.878788
birthMonth	76	2.878788
birthDay	76	2.878788
...		
total_tax_rate_country	182	6.893939
population_country	164	6.212121
latitude_country	164	6.212121
longitude_country	164	6.212121

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output sett

```
# Visualizing missing values using
missing_values = df.isnull().sum()
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar={'label': 'Missing Data'}, cmap='YlGnBu')
plt.show()
```

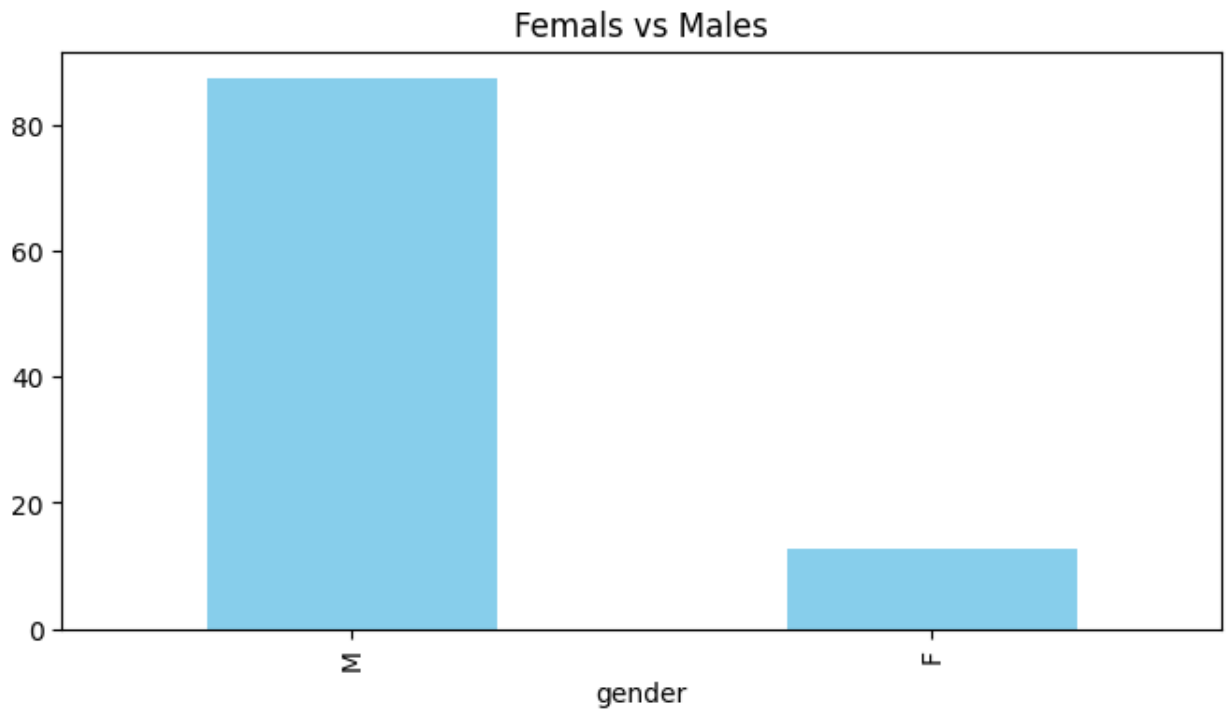
[21] ✓ 0.7s


0
81
162
243
324
405
486
567
648
729
810
891
972
1053
1134
1215
1296
1377
1458
1539
1620
1701
1782
1863
1944
2025
2106
2187
2268
2349
2430
2511
2592
1.0
0.8
0.6
0.4
0.2
0.0
rank
finalWorth
category
personName
age
country
city
source
industries
countryOfCitizenship
selfMade
status
gender
birthDate
lastName
firstName
date
birthYear
birthMonth
birthDay
cpi_country
cpi_change_country
gdp_country
gross_tertiary_education_enrollment
gross_primary_education_enrollment_country
life_expectancy_country
tax_revenue_country_country
total_tax_rate_country
population_country
latitude_country
longitude_country

```python
# Calculating the percentage of males and females
gender_counts = df['gender'].value_counts(normalize=True)*100

# Display the result
print("Percentage % of Males and Females in the Dataset:")
print(gender_counts.round(2))

#visualising the genders in the dataset
plt.figure(figsize=(10, 6))
gender_counts.plot(kind='bar', color='skyblue')
plt.title('Femals vs Males')
plt.show()
```

2] ✓ 0.1s

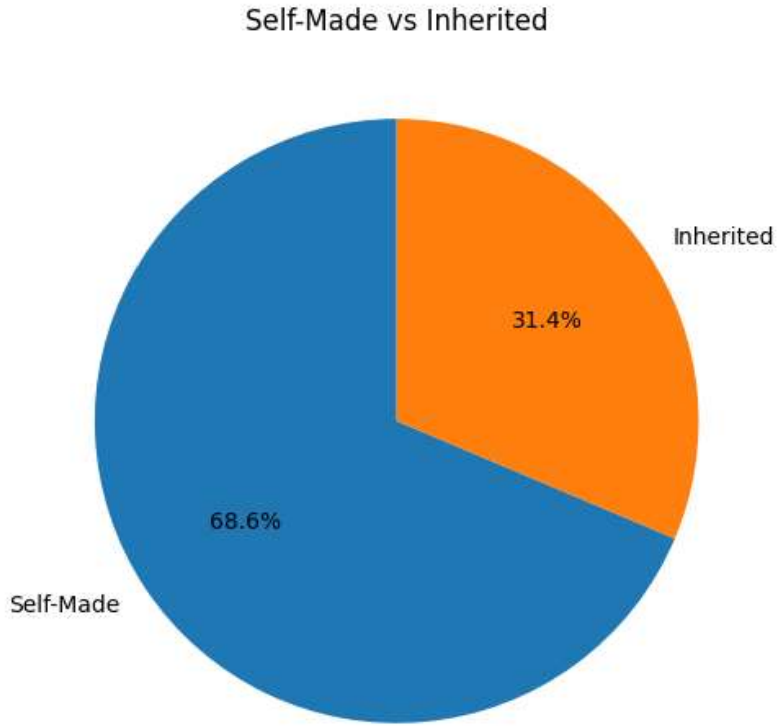
```
Percentage % of Males and Females in the Dataset:
gender
M    87.23
F    12.77
```



```python
# Calculating the percentage count of self-made and inherited billionaires
self_made_counts = df['selfMade'].value_counts(normalize=True) * 100

# Creating a DataFrame from the percentages
self_made_percentage_table = pd.DataFrame({
    'Status': ['Self-Made', 'Inherited'],
    'Percentage %': self_made_counts.values
})

# Displaying the result
print("Percentage Count of Self-Made and Inherited Billionaires:")
print(self_made_percentage_table.round(2))


#now  visualing the self made and inherited billionaires
plt.figure(figsize=(10, 6))
plt.pie(self_made_counts, labels=['Self-Made', 'Inherited'], autopct='%1.1f%%', startangle=90)
plt.title('Self-Made vs Inherited')
plt.show()
```

[17] ✓ 0.2s

```
Percentage Count of Self-Made and Inherited Billionaires:
      Status  Percentage %
0  Self-Made         68.64
1  Inherited         31.36
```

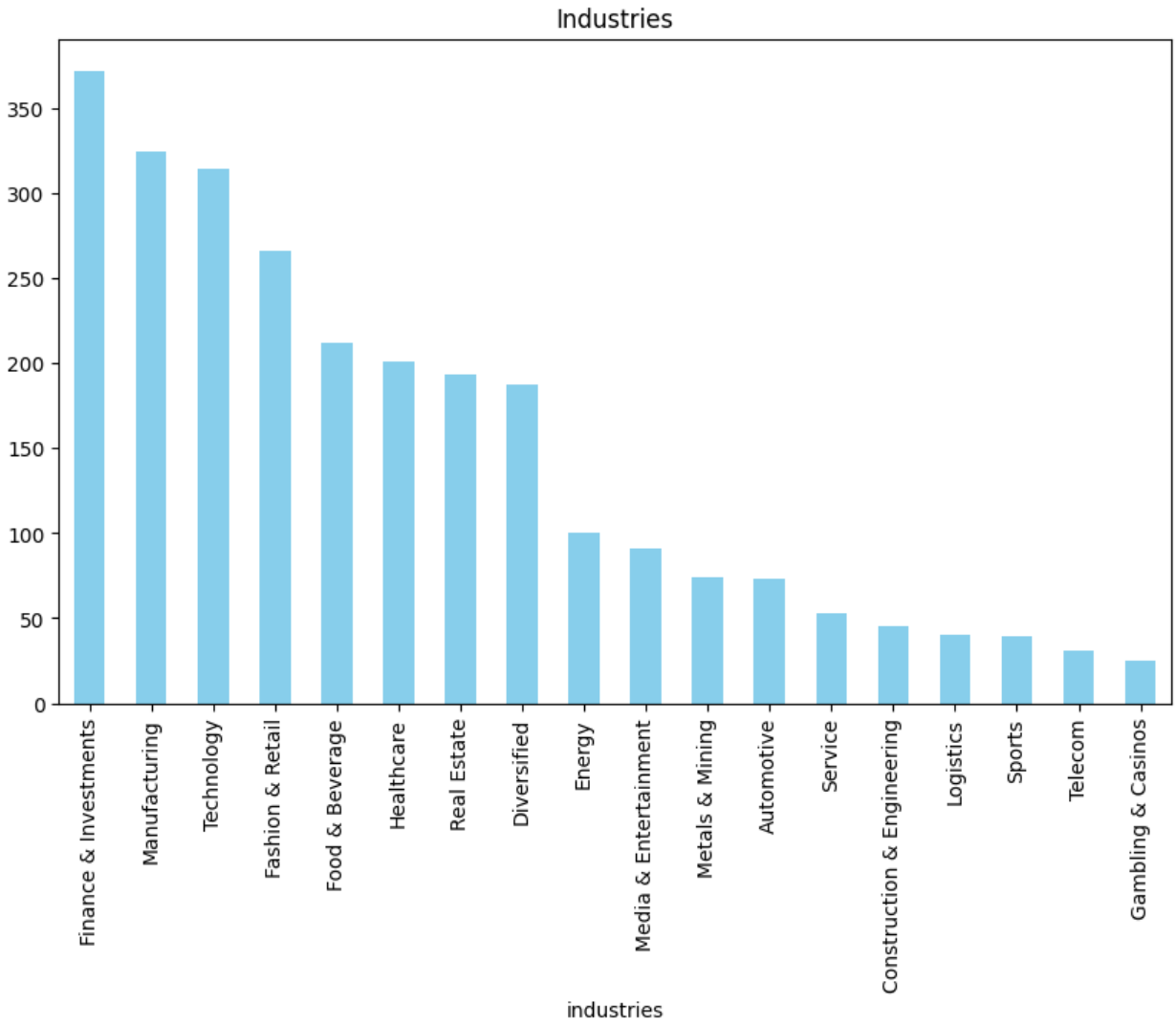


```python
#function to find statistics for numerical columns

def categorical_descriptive_stats(column):

    print(f" Statistics for '{column.name}':")

    # Frequency Counts
    frequency_counts = column.value_counts()
    print(f"Frequency Counts:\n{frequency_counts}\n")

    # Mode
    mode_value = column.mode().iloc[0]
    print(f"Mode: {mode_value}\n")

    # Unique Values
    unique_values = column.unique()
    print(f"Unique Values:\n{unique_values}\n")
```

```python
# Checking the industry with most Billionaires
categorical_descriptive_stats(df['industries'])

#visualing the data
plt.figure(figsize=(10, 6))
df['industries'].value_counts().plot(kind='bar', color='skyblue')
plt.title('Industries with most Billionaires')
plt.show()
```

[29] ✓ 0.3s

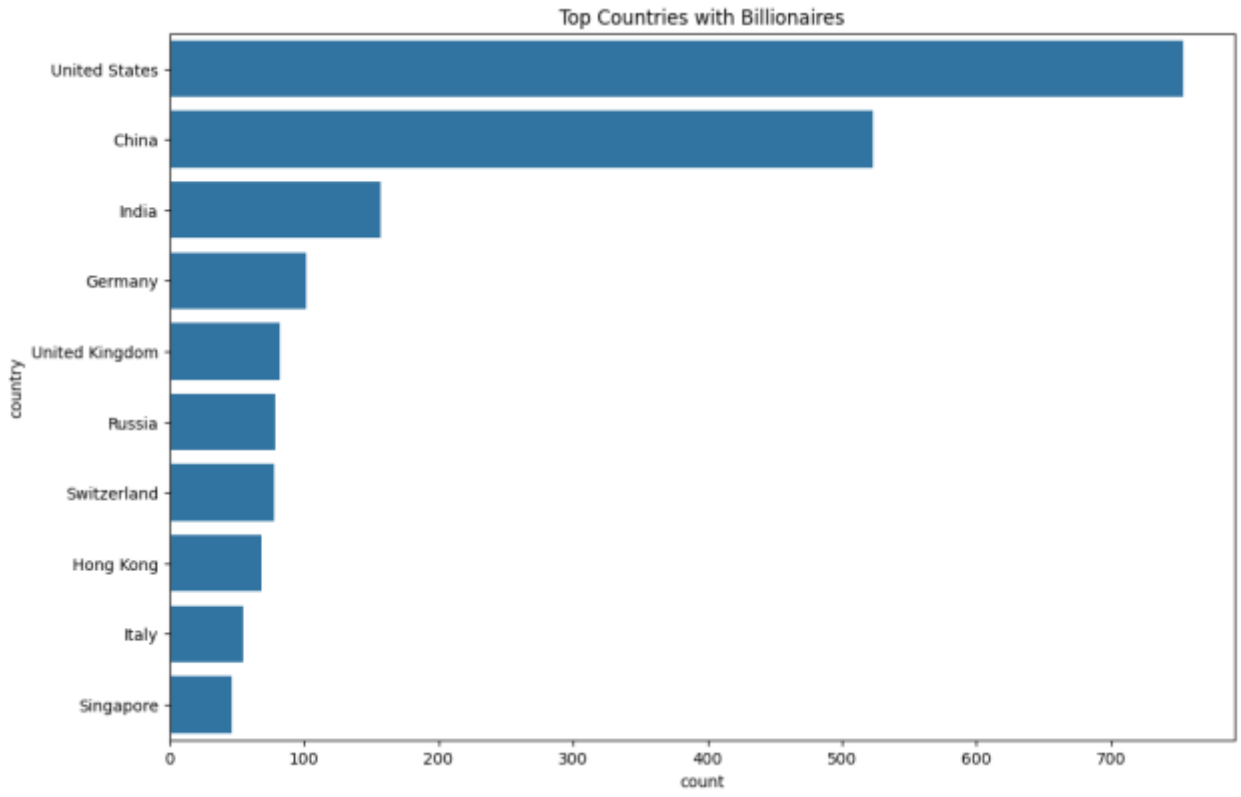
```
 Statistics for 'industries':
Frequency Counts:
industries
Finance & Investments         372
Manufacturing                 324
Technology                    314
Fashion & Retail              266
Food & Beverage               212
Healthcare                    201
Real Estate                   193
Diversified                   187
Energy                        100
Media & Entertainment          91
Metals & Mining                74
Automotive                     73
Service                        53
```




```
# countries with most billionaires
categorical_descriptive_stats(df['country'])
plt.figure(figsize=(12, 8))
df['country'].value_counts().plot(kind='bar', color='blue')
plt.title('Countries with Billionaires')
plt.show()
```

[48] ✓ 1.0s

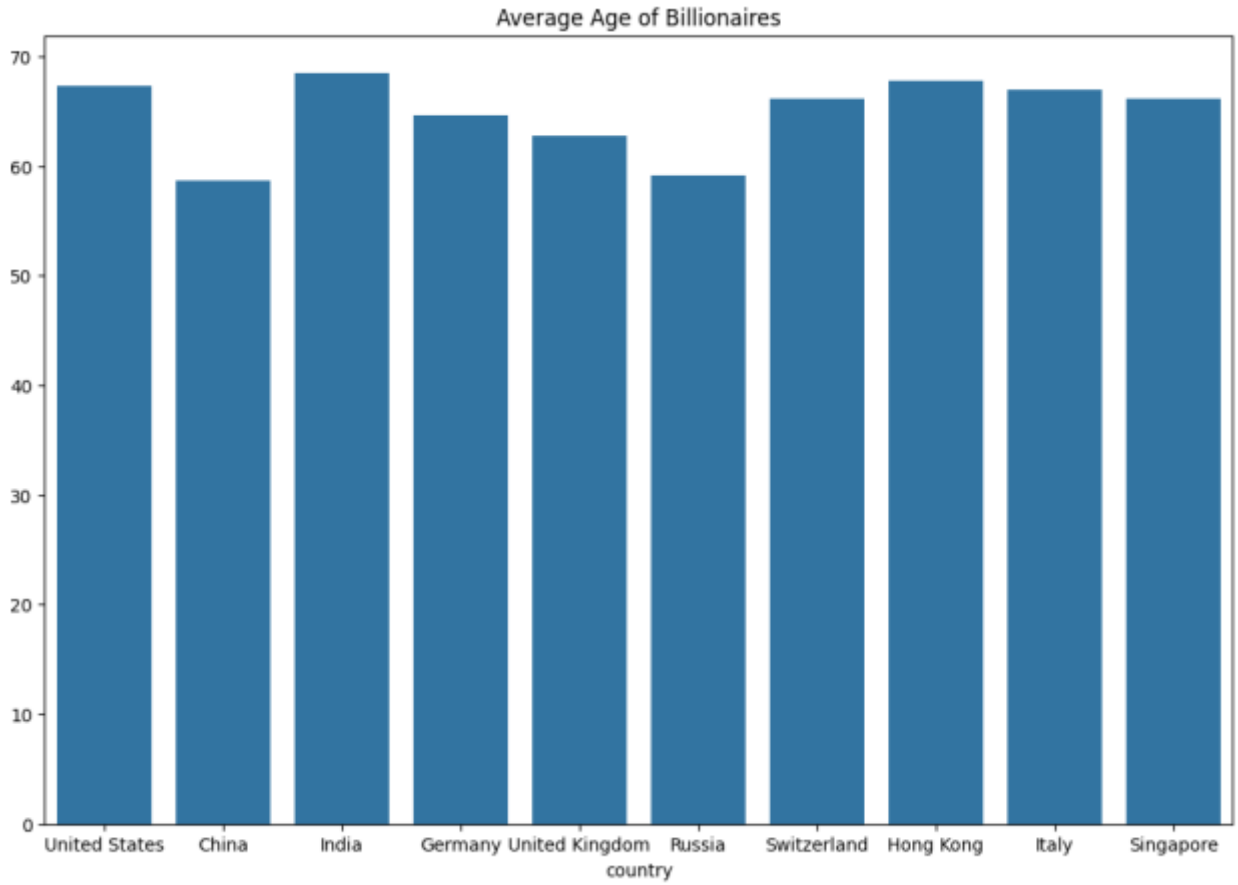
```
Statistics for 'country':
Frequency Counts:
country
United States          754
China                  523
India                  157
Germany                102
United Kingdom          82
                       ...
```




```python
#checking the avarage  of the billionaires in each country
top_countries_billionaires = df['country'].value_counts().head(10).index

# Calculate the average age of billionaires for the top countries
average_age = df.groupby('country')['age'].mean().loc[top_countries_billionaires]
print('Average Age:')
print(average_age)

#visualing the average age of billionaires for the top countries
plt.figure(figsize=(12, 8))
sns.barplot(x=average_age.index, y=average_age.values)
plt.title('Average Age of Billionaires in top countries')
plt.show()
```

[59] ✓ 0.1s

```
Average Age:
country
United States     67.306366
China             58.696429
India             68.458599
Germany           64.647727
United Kingdom    62.700000
Russia            59.101266
```




```
top_countries_billionaires = df['country'].value_counts().head(10).index


# Filter the DataFrame to include only billionaires from the top countries
filtered_df = df[df['country'].isin(top_countries_billionaires)]

# Calculate the average tax rate for the selected countries
average_tax_rate = filtered_df['total_tax_rate_country'].mean()
print(f'Average Tax Rate: {average_tax_rate:.2f}%')
```

[60] ✓ 0.0s

```
Average Tax Rate: 44.76%
```

```
# Group the data by country and calculate the average tax rate
average_tax_rate_country = df.groupby('country')['total_tax_rate_country'].mean()

# Find the country with the highest tax rate
country_with_highest_tax_rate = average_tax_rate_country.idxmax()
# Print the result along with the percentage value
print(f"The country with the highest tax rate is: {country_with_highest_tax_rate} ({average_tax_rate_country
    [country_with_highest_tax_rate]:.2f}%)")
```

[79] ✓ 0.0s

```
The country with the highest tax rate is: Argentina (106.30%)
```

```
industry_gender_counts = df.groupby(['industries', 'gender']).size().unstack()
industry_gender_counts['Most Involved'] = industry_gender_counts.idxmax(axis=1)
print(industry_gender_counts['Most Involved'])
```

[65] ✓ 0.0s

```
industries
Automotive                  M
Construction & Engineering  M
Diversified                 M
Energy                      M
Fashion & Retail            M
Finance & Investments       M
Food & Beverage             M
Gambling & Casinos          M
Healthcare                  M
Logistics                   M
Manufacturing               M
Media & Entertainment       M
Metals & Mining             M
Real Estate                 M
Service                     M
Sports                      M
Technology                  M
Telecom                     M
```

```
#checking the industry with highest success rate with Female

# Grouping the data by industry and gender
industry_gender_counts = df.groupby(['industries', 'gender']).size().unstack()

# Geting the count of women in each industry
women_counts = industry_gender_counts['F']

# Finding the industry with the highest count of women
most_involved_industry = women_counts.idxmax()

print(f"The industry with the most involved women is: {most_involved_industry}")
```

[66] ✓ 0.0s

```
The industry with the most involved women is: Food & Beverage
```

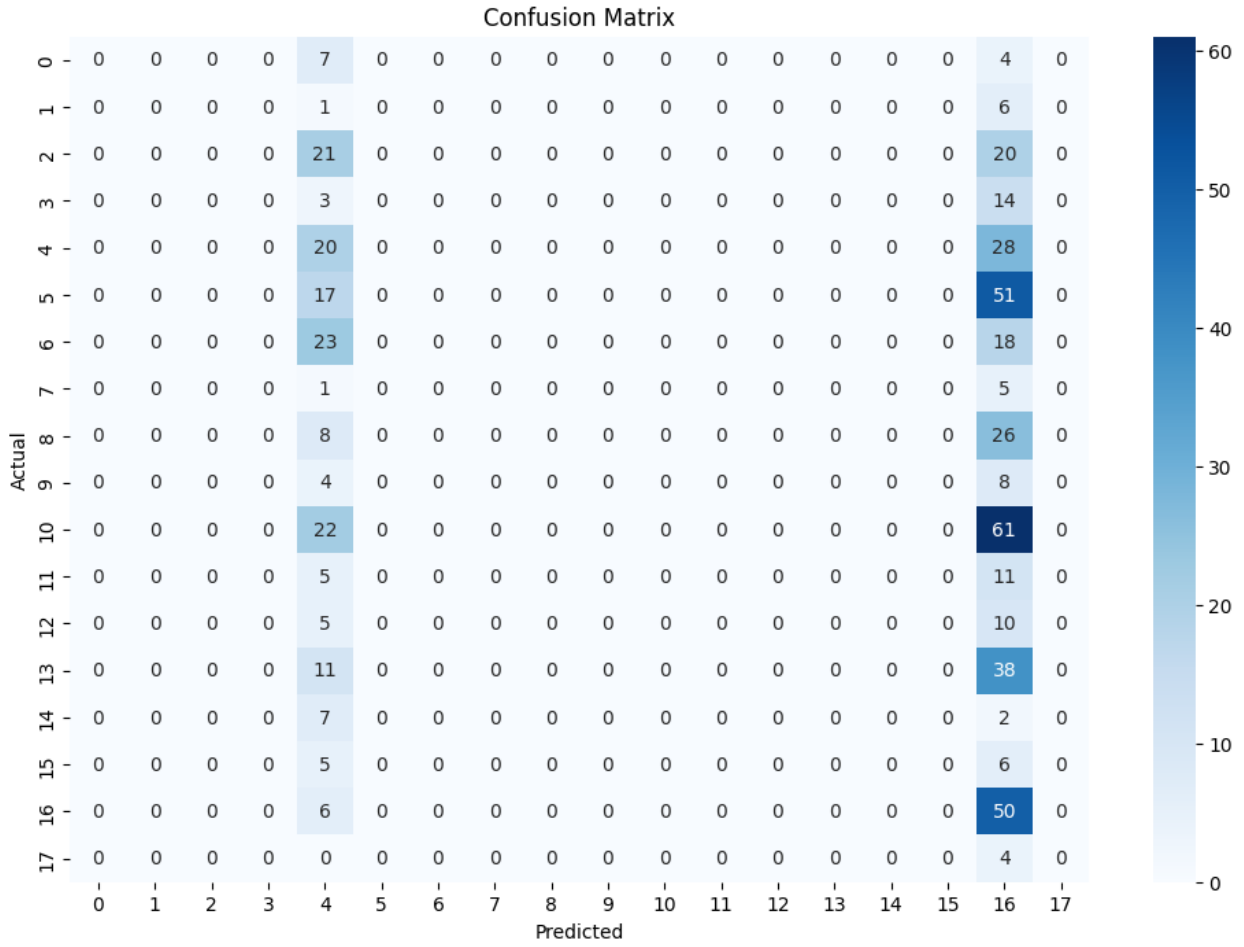
```
#checking the most involved industry with self made billionaires
# Grouping the data by industry and selfMade status
industry_selfmade_counts = df.groupby(['industries', 'selfMade']).size().unstack()

# Geting the count of self-made billionaires in each industry
selfmade_counts = industry_selfmade_counts[True]

most_involved_industry = selfmade_counts.idxmax()

print(f"The industry with the most involved self-made billionaires is: {most_involved_industry}")
```

[72] ✓ 0.0s

```
The industry with the most involved self-made billionaires is: Technology
```

```
features = ['selfMade']
target = 'industries'

X = df[features]
y = df[target]

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Choosing a classification algorithm and train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Making predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

#now printing the confusion matrix
confusion_matrix(y_test, y_pred)

#ploting the confusion matrix
plt.figure(figsize=(12, 8))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy: 0.13257575757575757
Precision: 0.025602157499914274
Recall: 0.13257575757575757
F1-score: 0.042365704173153974
```


Confusion Matrix
Actual
Predicted
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 4 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 6 0
0 0 0 0 21 0 0 0 0 0 0 0 0 0 0 0 20 0
0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 14 0
0 0 0 0 20 0 0 0 0 0 0 0 0 0 0 0 28 0
0 0 0 0 17 0 0 0 0 0 0 0 0 0 0 0 51 0
0 0 0 0 23 0 0 0 0 0 0 0 0 0 0 0 18 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 5 0
0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 26 0
0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 8 0
0 0 0 0 22 0 0 0 0 0 0 0 0 0 0 0 61 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 11 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 10 0
0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 38 0
0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 2 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 6 0
0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 50 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0
60
50
40
30
20
10
0

Question No .2:

```python
from collections import deque

def bfs(graph, root):
    visited = set()
    queue = deque([root])

    while queue:
        vertex = queue.popleft()
        print(vertex, end=" ")

        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)

graph = {
    'A': ['B', 'D'],
    'B': ['U', 'L'],
    'D': ['L'],
    'U': [],
    'L': ['A'],
    'H': ['G'],
}

bfs(graph, 'B')
```

Output: