BILLIONAIRES STATISTICS

Project Overview

1. Data Cleaning

In this initial phase, we ensure the dataset's accuracy and reliability through rigorous cleaning, focusing on standardizing column names, handling missing data, addressing duplicates, and managing outliers.

2. Univariate Feature Analysis

Delve into the nuances of individual features in this section, unveiling patterns in demographics, examining wealth distribution across countries/cities, and exploring correlations between economic indicators and the billionaire landscape.

3. Multivariate Feature Relationships Analysis

Gain a deeper understanding of feature relationships through multivariate analysis. Explore correlations between different features and uncover intricate patterns within the dataset.

4. Conclusion - Key Findings

Summarize key findings derived from the exploratory data analysis, highlighting significant patterns and insights.

1. Data Cleaning

This section covers the crucial step of Data Cleaning in the data analytics process.

To skip directly to to particular parts, use the following links:
1.1 Libraries / Reading Data
1.2 Discovering Data
1.3 Structuring Data
1.4 Handle Missing Data
1.5 Handle Duplicates

1.1 Libraries / Reading Data

In [1]:
```
import os
import warnings

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

# Set the style for Matplotlib plots
plt.style.use('ggplot')

# Suppress warnings for cleaner output
warnings.filterwarnings("ignore")
```

```
C:\Users\benne\AppData\Local\Temp\ipykernel_18372\3882677250.py:5: DeprecationWarnin
g:
Pyarrow will become a required dependency of pandas in the next major release of pan
das (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better inte
roperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

In [2]:
```
# Read the raw dataframe
df = pd.read_csv("data/raw_data.csv")
```

1.2 Discovering Data

In [3]:
```
df.head(10)
```

Out[3]:

	rank	finalWorth	category	personName	age	country	city	source	in
0	1	211000	Fashion & Retail	Bernard Arnault & family	74.0	France	Paris	LVMH	Fa
1	2	180000	Automotive	Elon Musk	51.0	United States	Austin	Tesla, SpaceX	Aut
2	3	114000	Technology	Jeff Bezos	59.0	United States	Medina	Amazon	Tec
3	4	107000	Technology	Larry Ellison	78.0	United States	Lanai	Oracle	Tec
4	5	106000	Finance & Investments	Warren Buffett	92.0	United States	Omaha	Berkshire Hathaway	Fi Inve
5	6	104000	Technology	Bill Gates	67.0	United States	Medina	Microsoft	Tec
6	7	94500	Media & Entertainment	Michael Bloomberg	81.0	United States	New York	Bloomberg LP	Entert
7	8	93000	Telecom	Carlos Slim Helu & family	83.0	Mexico	Mexico City	Telecom	
8	9	83400	Diversified	Mukesh Ambani	65.0	India	Mumbai	Diversified	Di
9	10	80700	Technology	Steve Ballmer	67.0	United States	Hunts Point	Microsoft	Tec

10 rows × 35 columns

In [4]:
```
df.shape
```

Out[4]:
```
(2640, 35)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2640 entries, 0 to 2639
Data columns (total 35 columns):
 #   Column                                      Non-Null Count  Dtype  
---  ------                                      --------------  -----  
 0   rank                                        2640 non-null   int64  
 1   finalWorth                                  2640 non-null   int64  
 2   category                                    2640 non-null   object 
 3   personName                                  2640 non-null   object 
 4   age                                         2575 non-null   float64
 5   country                                     2602 non-null   object 
 6   city                                        2568 non-null   object 
 7   source                                      2640 non-null   object 
 8   industries                                  2640 non-null   object 
 9   countryOfCitizenship                        2640 non-null   object 
 10  organization                                325 non-null    object 
 11  selfMade                                    2640 non-null   bool   
 12  status                                      2640 non-null   object 
 13  gender                                      2640 non-null   object 
 14  birthDate                                   2564 non-null   object 
 15  lastName                                    2640 non-null   object 
 16  firstName                                   2637 non-null   object 
 17  title                                       339 non-null    object 
 18  date                                        2640 non-null   object 
 19  state                                       753 non-null    object 
 20  residenceStateRegion                        747 non-null    object 
 21  birthYear                                   2564 non-null   float64
 22  birthMonth                                  2564 non-null   float64
 23  birthDay                                    2564 non-null   float64
 24  cpi_country                                 2456 non-null   float64
 25  cpi_change_country                          2456 non-null   float64
 26  gdp_country                                 2476 non-null   object 
 27  gross_tertiary_education_enrollment         2458 non-null   float64
 28  gross_primary_education_enrollment_country  2459 non-null   float64
 29  life_expectancy_country                     2458 non-null   float64
 30  tax_revenue_country_country                 2457 non-null   float64
 31  total_tax_rate_country                      2458 non-null   float64
 32  population_country                          2476 non-null   float64
 33  latitude_country                            2476 non-null   float64
 34  longitude_country                           2476 non-null   float64
dtypes: bool(1), float64(14), int64(2), object(18)
memory usage: 704.0+ KB
```

In [6]:

```
df.describe()
```

Out[6]:

	rank	finalWorth	age	birthYear	birthMonth	birthDay	cp
count	2640.000000	2640.000000	2575.000000	2564.000000	2564.000000	2564.000000	24
mean	1289.159091	4623.787879	65.140194	1957.183307	5.740250	12.099844	1
std	739.693726	9834.240939	13.258098	13.282516	3.710085	9.918876	
min	1.000000	1000.000000	18.000000	1921.000000	1.000000	1.000000	
25%	659.000000	1500.000000	56.000000	1948.000000	2.000000	1.000000	1
50%	1312.000000	2300.000000	65.000000	1957.000000	6.000000	11.000000	1
75%	1905.000000	4200.000000	75.000000	1966.000000	9.000000	21.000000	1
max	2540.000000	211000.000000	101.000000	2004.000000	12.000000	31.000000	2

1.3 Structuring Data

In [7]:
```
# Keep column identifiers consistent
df = df.rename(
    columns={
        'finalWorth': 'final_worth_usd',
        'personName': 'person_name', 'countryOfCitizenship': 'country_of_citizenshi
        'birthDate': 'birth_date',
        'lastName': 'last_name',
        'firstName': 'first_name',
        'residenceStateRegion': 'residence_state_region', 'birthYear': 'birth_year'
        'birthMonth': 'birth_month',
        'birthDay': 'birth_day',
        'gdp_country': 'gdp_country_usd',
        'gross_tertiary_education_enrollment': 'gross_tertiary_education_enrollment
        , 'tax_revenue_country_country': 'tax_revenue_country_usd'
    }
)
```

In [8]:
```
# Check datatypes for date-columns
print(type(df['birth_date'][0]))
print(type(df['date'][0]))
```

```
<class 'str'>
<class 'str'>
```

In [9]:
```
# Convert str datatype to datetime object and validate the updated datatype
df['birth_date'] = pd.to_datetime(df['birth_date'])
df['date'] = pd.to_datetime(df['date'])
print(type(df['birth_date'][0]))
print(type(df['date'][0]))
```

```
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

In [10]:
```
# Dict for replacing bool values in the 'wealth_source' column for better clarity
replace_dict = {
```

```
    True: 'Self-Made',
    False: 'Inherited/Unearned'
}
df['wealth_source'] = df['wealth_source'].replace(replace_dict)
```

In [11]:
```
# Convert 'gdp_country_in_dollars' from str to numeric after removing '$' and comma
df['gdp_country_usd'] = pd.to_numeric(
    df['gdp_country_usd'].str.strip('$ ').
    str.replace(',', '')
)
```

In [12]:
```
# Select only relevant features for the specific data analysis project
filtered_df = df[[
    'rank', 'final_worth_usd', 'person_name', 'organization', 'title', 'residence_s
    'age', 'country', 'country_of_citizenship', 'city', 'industries', 'wealth_sourc
    'life_expectancy_country', 'gross_tertiary_education_enrollment_country',
    'gross_primary_education_enrollment_country', 'gdp_country_usd', 'total_tax_rat
]]
```

Convert and Format 'final_worth_usd': Multiply the 'final_worth_usd' column by 1,000,000 to adapt the unit and create a new column, 'final_worth_usd_formatted,' which represents the values in billions for better readability.

In [13]:
```
# Before adapting the unit
print(filtered_df['final_worth_usd'][6])
```

```
94500
```

In [14]:
```
filtered_df['final_worth_usd'] = filtered_df['final_worth_usd'].mul(1000000)
```

In [15]:
```
def readable_numbers(x):
    """ takes a large number and formats it into K,M,B, to make it more readable"""
    x_abs = abs(x)
    if x_abs >= 1e9:
        s = '{:1.1f}B'.format(x * 1e-9)
    elif (x_abs < 1e9) & (x_abs >= 1e6):
        s = '{:1.1f}M'.format(x * 1e-6)
    else:
        s = '{:1.1f}K'.format(x * 1e-3)
    return s
```

In [16]:
```
filtered_df['final_worth_usd']
```

Out[16]:
```
0       211000000000
1       180000000000
2       114000000000
3       107000000000
4       106000000000
            ...
2635      1000000000
2636      1000000000
2637      1000000000
2638      1000000000
2639      1000000000
Name: final_worth_usd, Length: 2640, dtype: int64
```

In [17]:
```
filtered_df['final_worth_usd_formatted'] = filtered_df['final_worth_usd'].apply(rea
```

In [18]:
```
filtered_df[['final_worth_usd', 'final_worth_usd_formatted']].sort_values(by='final
```

Out[18]:

	final_worth_usd	final_worth_usd_formatted
2639	1000000000	1.0B
2565	1000000000	1.0B
2566	1000000000	1.0B
2567	1000000000	1.0B
2568	1000000000	1.0B
...	...	...
4	106000000000	106.0B
3	107000000000	107.0B
2	114000000000	114.0B
1	180000000000	180.0B
0	211000000000	211.0B

2640 rows × 2 columns

In [19]:
```
# After converting + formatting
print(filtered_df['final_worth_usd'][6])
print(filtered_df['final_worth_usd_formatted'][6])
```

```
94500000000
94.5B
```

1.4 Handle missing data

Checking for Zero Values

Before proceeding with handling missing data, it's crucial to check for zero values, as they could potentially indicate missing or undefined data.

In [20]:
```python
# Check for zero values in the dataframe
(df == 0).sum()
```

Out[20]:
```
rank                                          0
final_worth_usd                               0
category                                      0
person_name                                   0
age                                           0
country                                       0
city                                          0
source                                        0
industries                                    0
country_of_citizenship                        0
organization                                  0
wealth_source                                 0
status                                        0
gender                                        0
birth_date                                    0
last_name                                     0
first_name                                    0
title                                         0
date                                          0
state                                         0
residence_state_region                        0
birth_year                                    0
birth_month                                   0
birth_day                                     0
cpi_country                                   0
cpi_change_country                            0
gdp_country_usd                               0
gross_tertiary_education_enrollment_country   0
gross_primary_education_enrollment_country    0
life_expectancy_country                       0
tax_revenue_country_usd                       0
total_tax_rate_country                        0
population_country                            0
latitude_country                              0
longitude_country                             0
dtype: int64
```

If there are numeric columns with a substantial number of zeros, it's essential to investigate whether these zeros represent genuine data or if they are indicative of missing information. In the latter case, it's advisable to replace these zeros with NaN values to ensure consistent handling of missing data in subsequent steps. For example:

```python
df = df.replace(0, np.nan)
```

This step ensures a thorough examination of zero values and provides a standardized approach for addressing potential missing data. Once this is done, you can proceed with the subsequent steps of handling missing values in your dataset.

In [21]:
```
# Function to get an overview of the missing data per column
def get_missing_data_rate(input_df: pd.DataFrame):
    total = input_df.isna().sum().sort_values(ascending=False)
    percent = (input_df.isna().sum() / input_df.isna().count()).sort_values(ascendi
    missing_data = pd.concat([total, percent], axis=1, keys=['Missing Data Total',
    return missing_data
```

In [22]:
```
get_missing_data_rate(df)
```

Out[22]:

	Missing Data Total	Missing Data Percent
organization	2315	0.876894
title	2301	0.871591
residence_state_region	1893	0.717045
state	1887	0.714773
cpi_change_country	184	0.069697
cpi_country	184	0.069697
tax_revenue_country_usd	183	0.069318
total_tax_rate_country	182	0.068939
life_expectancy_country	182	0.068939
gross_tertiary_education_enrollment_country	182	0.068939
gross_primary_education_enrollment_country	181	0.068561
latitude_country	164	0.062121
population_country	164	0.062121
gdp_country_usd	164	0.062121
longitude_country	164	0.062121
birth_date	76	0.028788
birth_year	76	0.028788
birth_month	76	0.028788
birth_day	76	0.028788
city	72	0.027273
age	65	0.024621
country	38	0.014394
first_name	3	0.001136
last_name	0	0.000000
source	0	0.000000
category	0	0.000000
person_name	0	0.000000
wealth_source	0	0.000000
industries	0	0.000000
country_of_citizenship	0	0.000000

	Missing Data Total	Missing Data Percent
status	0	0.000000
gender	0	0.000000
date	0	0.000000
final_worth_usd	0	0.000000
rank	0	0.000000

Drop columns when more than 15 % of the data is missing

In [23]:
```
filtered_df = filtered_df.drop(columns=['title', 'organization', 'residence_state_r
```

In [24]:
```
get_missing_data_rate(filtered_df)
```

Out[24]:

	Missing Data Total	Missing Data Percent
life_expectancy_country	182	0.068939
total_tax_rate_country	182	0.068939
gross_tertiary_education_enrollment_country	182	0.068939
gross_primary_education_enrollment_country	181	0.068561
population_country	164	0.062121
gdp_country_usd	164	0.062121
city	72	0.027273
age	65	0.024621
country	38	0.014394
rank	0	0.000000
wealth_source	0	0.000000
gender	0	0.000000
final_worth_usd	0	0.000000
industries	0	0.000000
country_of_citizenship	0	0.000000
person_name	0	0.000000
final_worth_usd_formatted	0	0.000000

Handling Remaining Missing Data

When dealing with remaining missing data in your dataset, several options are available:

1. **Drop Rows:**

   - You can choose to drop the remaining rows with missing data. This is a straightforward approach but might result in a loss of information.

2. **Fill NaN Entries for Categorical Columns:**

   - For categorical columns, consider filling NaN entries with the mode (the most frequent value) to retain information without significant data loss.

3. **Fill NaN Entries for Numerical Columns:**

   - For numerical columns, filling NaN entries with the mean or median of the specific column is an option. Use the mean if the data is not skewed and does not contain outliers. If skewness or outliers are present, the median is a more robust fill option.

4. **Consideration for Outliers:**

   - It's important to be mindful of outliers in the data. If your dataset contains skewed data or outliers, favor using the median as the NaN-fill option to avoid the influence of extreme values.

5. **Avoid Biased Data:**

   - To ensure the integrity of your analysis and avoid biased results, consider dropping the remaining rows with missing data. This helps maintain the overall quality and fairness of your dataset.

Choose the appropriate strategy based on the nature and characteristics of your data, taking into account the potential impact on the analysis and results.

In [25]:

```
# Check rows with missing data
filtered_df[filtered_df.isna().any(axis=1)]
```

Out[25]:

	rank	final_worth_usd	person_name	age	country	country_of_citizenship	c
32	33	38000000000	Li Ka-shing	94.0	Hong Kong	Hong Kong	Na
46	47	29500000000	Lee Shau Kee	95.0	Hong Kong	Hong Kong	Hong Ko
85	86	18900000000	Eyal Ofer	72.0	Monaco	Israel	Mor Ca
107	108	15800000000	Karl Albrecht Jr. & family	NaN	Germany	Germany	Na
108	108	15800000000	Beate Heister	NaN	NaN	Germany	Na
...	...	...	...	...	...	...	
2609	2540	1000000000	Réal Plourde	NaN	Canada	Canada	Westmou
2610	2540	1000000000	Vera Rechulski Santo Domingo	74.0	Bermuda	Brazil	Na
2626	2540	1000000000	Masaru Wasami	77.0	NaN	Japan	Na
2629	2540	1000000000	Toto Wolff	51.0	Monaco	Austria	Na
2630	2540	1000000000	Franziska Wuerbser	35.0	NaN	Germany	Na

256 rows × 17 columns

In [26]:

```
filtered_df.isna().sum().max()
```

Out[26]: 182

In [27]:

```
# Drop remaining NaN-Enry rows
filtered_df = filtered_df.dropna(axis=0)
```

In [28]:

```
# Validate that there's no missing data left
filtered_df.isna().sum().max()
```

Out[28]: 0

1.5 Handle Duplicates

Duplicate entries in a dataset can introduce inconsistencies and skew analysis results. You begin by identifying and detecting duplicate rows in your dataset.

In [29]:
```
# Check for exact duplicated rows in the df
filtered_df.loc[filtered_df.duplicated()]
```

Out[29]:

rank	final_worth_usd	person_name	age	country	country_of_citizenship	city	industrie

In [30]:
```
# Check for feature-specific duplicated rows based on 'person_name' and 'country'
filtered_df.loc[filtered_df.duplicated(subset=['person_name', 'country'])]
```

Out[30]:

	rank	final_worth_usd	person_name	age	country	country_of_citizenship	city
2112	2020	1400000000	Wang Yanqing & family	76.0	China	China	Weihai
2317	2259	1200000000	Li Li	59.0	China	China	Shenzhen

In [31]:
```
# Investigate feature-specific duplicates for 'person_name'
filtered_df.query('person_name == "Li Li" or person_name.str.startswith("Wang Yanqi
```

Out[31]:

	rank	final_worth_usd	person_name	age	country	country_of_citizenship	city
785	766	3700000000	Wang Yanqing & family	56.0	China	China	Wuxi
1045	1027	2900000000	Li Li	57.0	China	China	Changsha
2112	2020	1400000000	Wang Yanqing & family	76.0	China	China	Weihai
2317	2259	1200000000	Li Li	59.0	China	China	Shenzhen

After identifying duplicate rows, the next steps involve deciding whether to drop or maintain them, considering the potential consequences of data loss. In this case, it's observed that for 'Wang Yanqing & family' and 'Li Li,' there are entries with slight differences in gender and age, suggesting they might not belong to the same person.

To demonstrate how to drop potentially feature-specific duplicated rows, the following code can be used:

In [32]:
```
# Drop potentially feature-specific duplicated rows
filtered_df_no_duplicates = filtered_df.loc[~filtered_df.duplicated(subset=['person
    .reset_index(drop=True).copy()
```

In [33]:
```
# Verify that dropping potentially duplicated rows worked properly
filtered_df_no_duplicates.query('person_name == "Li Li" or person_name.str.startswi
```

Out[33]:

	rank	final_worth_usd	person_name	age	country	country_of_citizenship	city
731	766	3700000000	Wang Yanqing & family	56.0	China	China	Wuxi
971	1027	2900000000	Li Li	57.0	China	China	Changsha

1.6 Handle outliers

Handling Outliers in Final Worth Distribution

Outliers, or extreme values, in the final worth distribution of billionaire data can introduce biases and distort the overall analysis. It is essential to address these outliers to ensure the analysis accurately reflects the underlying patterns in the data. One widely employed method is the Interquartile Range (IQR) approach, involving the following steps:

Calculate Percentiles: Compute the 25th (Q1) and 75th (Q3) percentiles of the final worth values in the dataset.

Calculate IQR: Determine the Interquartile Range (IQR) by subtracting Q1 from Q3.

Define Upper and Lower Thresholds: Establish upper and lower thresholds for identifying outliers, typically considering values outside 1.5 times the IQR as potential outliers.

Filter Outliers Using Boolean Masks: Use boolean masks to filter the dataframe, retaining only rows where the final worth is within the defined upper and lower limits.

Analyze Results: Examine the impact of outlier handling on central tendency measures, such as mean and median, to understand the distribution's revised characteristics.

In [34]:

```
print(f"Final Worth Mean before handling outliers: {readable_numbers(filtered_df['f
print(f"Final Worth Median before handling outliers: {readable_numbers(filtered_df[
```

```
Final Worth Mean before handling outliers: 4.8B
Final Worth Median before handling outliers: 2.4B
```

In [35]:

```
# List to store all generated plots
all_plots = []
```

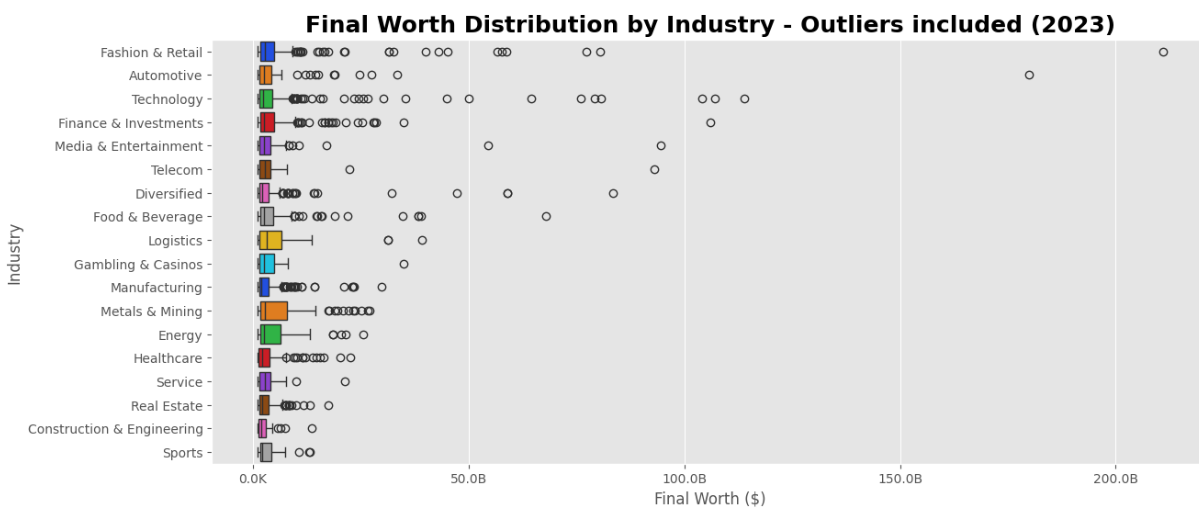
In [36]:

```
# Define a function for boxplot creation for better reusability
def create_box_plot(input_df, input_title, show_fliers):
    plt.figure(figsize=(14, 6))
    palette = sns.color_palette('bright')
    boxplot = sns.boxplot(
        data=input_df,
        x='final_worth_usd',
        y='industries',
        showfliers=show_fliers,
```

```
        palette=palette
    )
    boxplot.set(
        xlabel='Final Worth ($)',
        ylabel='Industry'
    )
    # Format the x-axis tick labels
    boxplot.set_xticklabels([readable_numbers(x) for x in boxplot.get_xticks()])
    plt.title(label=input_title, fontsize=18, fontweight='bold')
    plt.show()
    return boxplot
```

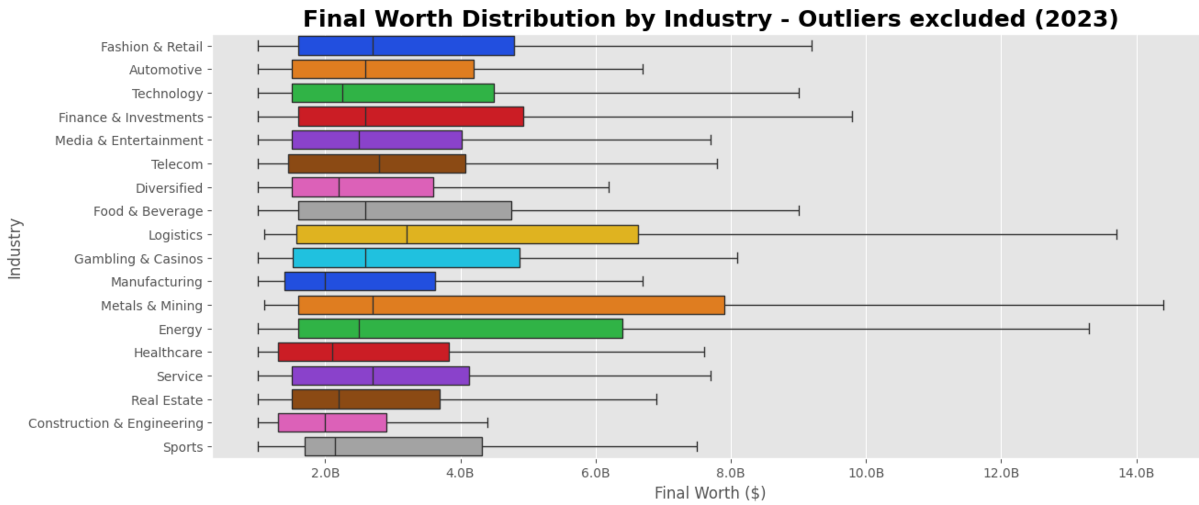
Usually you can check for outliers through a boxplot related to the specific numerical (x) and a dependent categorical (y) feature; But this type of dataset is very susceptible to outliers because of the wide range of billionaires' wealth

In [37]:
```
# Outliers included
boxplot_title_outliers = "Final Worth Distribution by Industry - Outliers included
file_name_1_6_1 = "Final_Worth_Distribution_by_Industry_Outliers_included_M.png"
boxplot_outliers_included = create_box_plot(filtered_df, boxplot_title_outliers, sh
all_plots.append((boxplot_outliers_included, file_name_1_6_1))
```



To get a better view of the final worth distribution we exclude the outliers

In [38]:
```
boxplot_title_no_outliers = "Final Worth Distribution by Industry - Outliers exclud
file_name_1_6_2 = "Final_Worth_Distribution_by_Industry_Outliers_excluded_M.png"
boxplot_outliers_excluded = create_box_plot(filtered_df, boxplot_title_no_outliers,
all_plots.append((boxplot_outliers_excluded, file_name_1_6_2))
```



Here starts the IQR approach

In [39]:
```
# Calculate 25th percentile of annual strikes
percentile25 = filtered_df['final_worth_usd'].quantile(0.25)

# Calculate 75th percentile of annual strikes
percentile75 = filtered_df['final_worth_usd'].quantile(0.75)

# Calculate interquartile range
iqr = percentile75 - percentile25

# Calculate upper and lower thresholds for outliers
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr

print('Upper limit is: ', readable_numbers(upper_limit))
print('Lower limit is: ', readable_numbers(lower_limit))
```

```
Upper limit is:  8.5B
Lower limit is:  -2.7B
```

Boolean masks were used to filter the dataframe so it only contained rows where the number of strikes was less than the lower limit / more than the upper limit

In [40]:
```
print(len(filtered_df[filtered_df['final_worth_usd'] < lower_limit]))
print(len(filtered_df[filtered_df['final_worth_usd'] > upper_limit]))
```

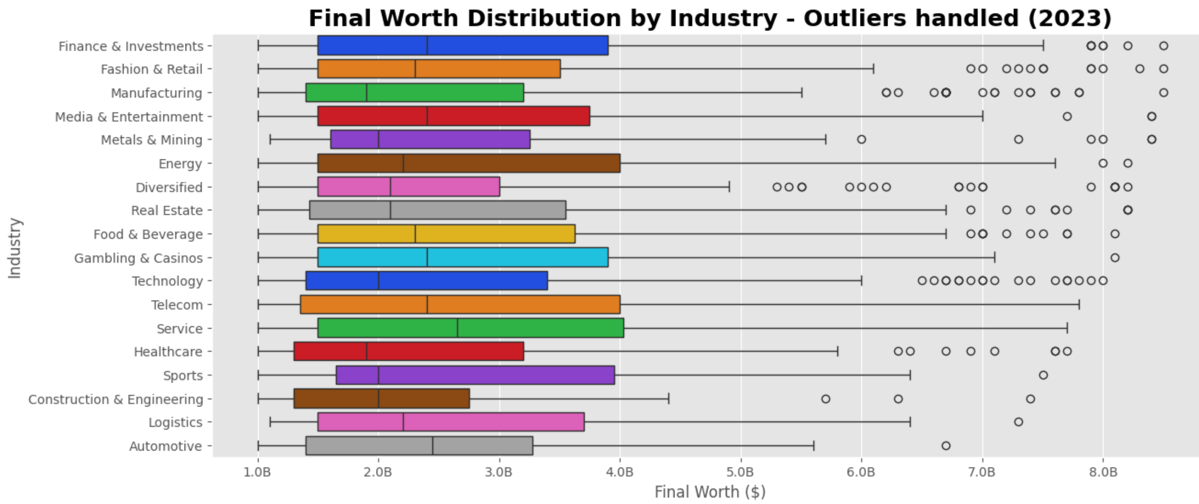
```
0
227
```

In [41]:
```
mask = (filtered_df['final_worth_usd'] >= lower_limit) & (filtered_df['final_worth_
                                                          upper_limit)
outliers_handled = filtered_df[mask].copy()
print(f"Final Worth Mean after handling outliers: {readable_numbers(outliers_handle
print(f"Final Worth Median after handling outliers: {readable_numbers(outliers_hand
```

```
Final Worth Mean after handling outliers: 2.8B
Final Worth Median after handling outliers: 2.2B
```

As shown, the data is now less widely dispersed and there are way less outliers than before

In [42]:
```
boxplot_title_outliers_handled = "Final Worth Distribution by Industry - Outliers h
file_name_1_6_3 = "Final_Worth_Distribution_by_Industry_Outliers_handled_M.png"
boxplot_outliers_handled = create_box_plot(outliers_handled, boxplot_title_outliers
all_plots.append((boxplot_outliers_handled, file_name_1_6_3))
```



1.7 Save The Cleaned Dataframe

Not to be forgotten

While handling outliers using the IQR method is a common practice in exploratory data analysis (EDA), it is important to be aware of its potential drawbacks. Removing outliers can lead to data loss and bias, which will affect the overall representativeness of the analysis. The decision on how to deal with outliers (delete, reassign or leave) should be made carefully, taking into account the specific characteristics of the dataset and the intended use, especially in the context of EDA without subsequent development of predictive machine learning models. Whether outliers are retained, reassigned or removed depends on the type and size of the dataset and the objectives of the analysis. It is crucial to balance the benefits of outlier correction with the potential drawbacks to ensure a thoughtful and context-aware approach.

For future EDA, you should use the original filtered df (without outlier handling), because outliers aren't systematic data errors here that could influence our analysis in a bad way; they are a valuable part of the overall df.

In [43]:
```
filtered_df.to_csv("data/cleaned_data.csv", index=False)
```

In [44]:
```
outliers_handled.to_csv("data/cleaned_data_outliers_handled.csv", index=False)
```

2. Univariate Feature Analysis

Explore the data analytics process of Univariate Feature Analysis in this section.

To skip directly to to particular parts, use the following links:

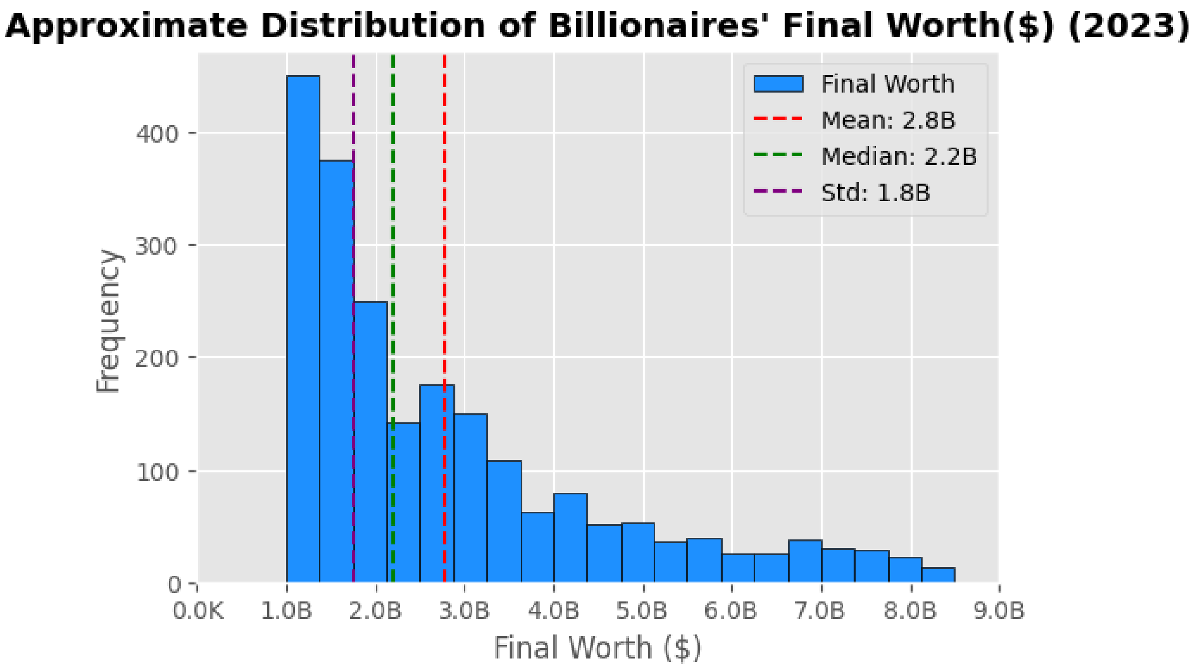
2.1 Feature: Final Worth | Hist-Chart
2.2 Feature: Country | Bar-Plot
2.3 Feature: City | Bar-Plot
2.4 Feature: Industries | BarH-Plot
2.5 Feature: Age | Hist-Chart
2.6 Feature: Gender | Pie-Chart
2.7 Feature: Wealth-Source | Pie-Chart

In [45]:
```
df = pd.read_csv("data/cleaned_data.csv")
df_outliers_handled = pd.read_csv("data/cleaned_data_outliers_handled.csv")
```

2.1 Feature: Final Worth | Hist-Chart

In [46]:
```
# Plot histogram
hist_chart_2_1 = df_outliers_handled['final_worth_usd'].plot(kind='hist',
                                                             bins=20,
                                                             figsize=(6, 4),
                                                             color='dodgerblue',
                                                             edgecolor='black',
                                                             label='Final Worth')

# Set title and labels
plt.title("Approximate Distribution of Billionaires' Final Worth($) (2023)",
          fontsize=14,
          fontweight="bold")
plt.xlabel('Final Worth ($)')

# Set ticks and labels
ticks = hist_chart_2_1.get_xticks()
hist_chart_2_1.set_xticks(ticks)
hist_chart_2_1.set_xticklabels([readable_numbers(x) for x in ticks])

# Calculate statistics
mean_final_worth, median_final_worth, std_final_worth = np.mean(df_outliers_handled
    df_outliers_handled['final_worth_usd']), np.std(df_outliers_handled['final_wort

# Add vertical lines for statistics
for stat, color, label in zip(
        [mean_final_worth, median_final_worth, std_final_worth],
        ['r', 'g', 'purple'],
        ['Mean', 'Median', 'Std']):
    hist_chart_2_1.axvline(stat, color=color, linestyle='dashed', label=f'{label}:

# Display legend
hist_chart_2_1.legend()

file_name_2_1 = "Approximate_Distribution_of_Billionaires_Final_Worth_Worldwide_U.p
```

```
all_plots.append((hist_chart_2_1, file_name_2_1))
plt.show()
```



2.2 Feature: Country | Bar-Plot

In [47]:
```
len(df['country'].unique())
```

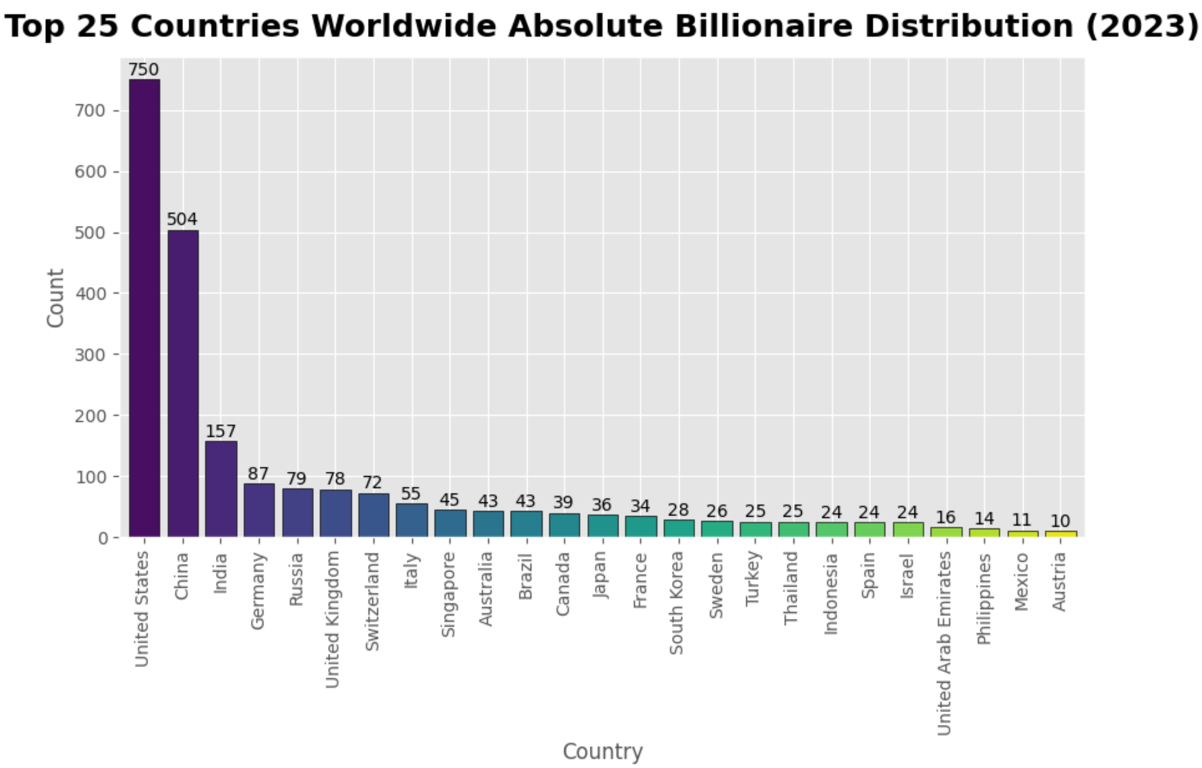
Out[47]:
```
65
```

In [48]:
```
country_counts = df['country'].value_counts()
country_counts
```

Out[48]:
```
country
United States    750
China            504
India            157
Germany           87
Russia            79
                ...
Portugal           1
Georgia            1
Colombia           1
Uzbekistan         1
Armenia            1
Name: count, Length: 65, dtype: int64
```

In [49]:
```
country_counts_top_25 = country_counts.head(25).sort_values(ascending=False)
# Use a seaborn color palette
colors = sns.color_palette('viridis', len(country_counts_top_25))

bar_plot_2_2 = country_counts_top_25.plot(kind='bar',
                                          figsize=(10, 5),
```

```
                                                   color=colors,
                                                   edgecolor='black',
                                                   width=0.8)

plt.title('Top 25 Countries Worldwide Absolute Billionaire Distribution (2023)',
          fontsize=18,
          fontweight='bold',
          y=1.02)
bar_plot_2_2.set_ylabel('Count')
bar_plot_2_2.set_xlabel('Country')
for i, count in enumerate(country_counts_top_25):
    plt.text(i, count + 1, str(count), ha='center', va='bottom')

file_name_2_2 = "Top_25_Countries_Worldwide_Absolute_Billionaire_Distribution_U.png
all_plots.append((bar_plot_2_2, file_name_2_2))
plt.show()
```

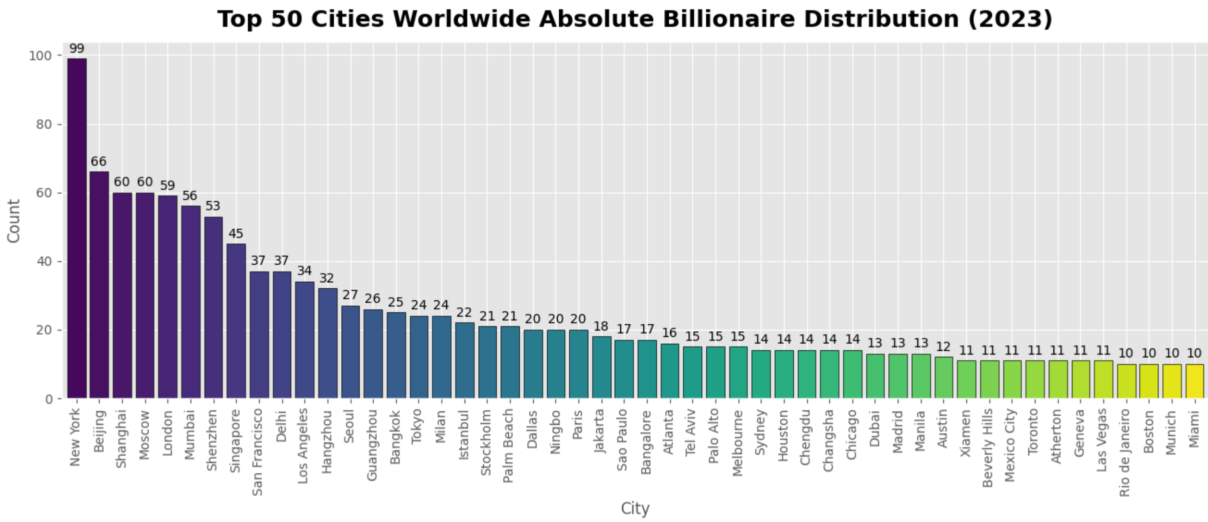


2.3 Feature: City | Bar-Plot

In [51]:
```
city_counts = df['city'].value_counts()
city_counts
```

Out[51]:
```
city
New York       99
Beijing        66
Shanghai       60
Moscow         60
London         59
               ..
Brownsville     1
Montpellier     1
Santa Clara     1
Stuttgart       1
Makati          1
Name: count, Length: 711, dtype: int64
```

In [52]:
```python
city_counts_top_50 = city_counts.head(50)
# Use a seaborn color palette
colors = sns.color_palette('viridis', len(city_counts_top_50))

# Create the bar plot
bar_plot_2_3 = city_counts_top_50.plot(kind='bar',
                                       figsize=(16, 5),
                                       color=colors,
                                       edgecolor='black',
                                       width=0.8)

# Add title and labels
plt.title("Top 50 Cities Worldwide Absolute Billionaire Distribution (2023)",
          fontsize=18,
          fontweight='bold',
          y=1.02)
bar_plot_2_3.set_ylabel('Count')
bar_plot_2_3.set_xlabel('City')

# Add bar labels with annotations
for i, count in enumerate(city_counts_top_50):
    bar_plot_2_3.text(i, count + 1, str(count), ha='center', va='bottom', fontsize=

file_name_2_3 = "Top_50_Cities_Worldwide_Absolute_Billionaire_Distribution_U.png"
all_plots.append((bar_plot_2_3, file_name_2_3))
plt.show()
```



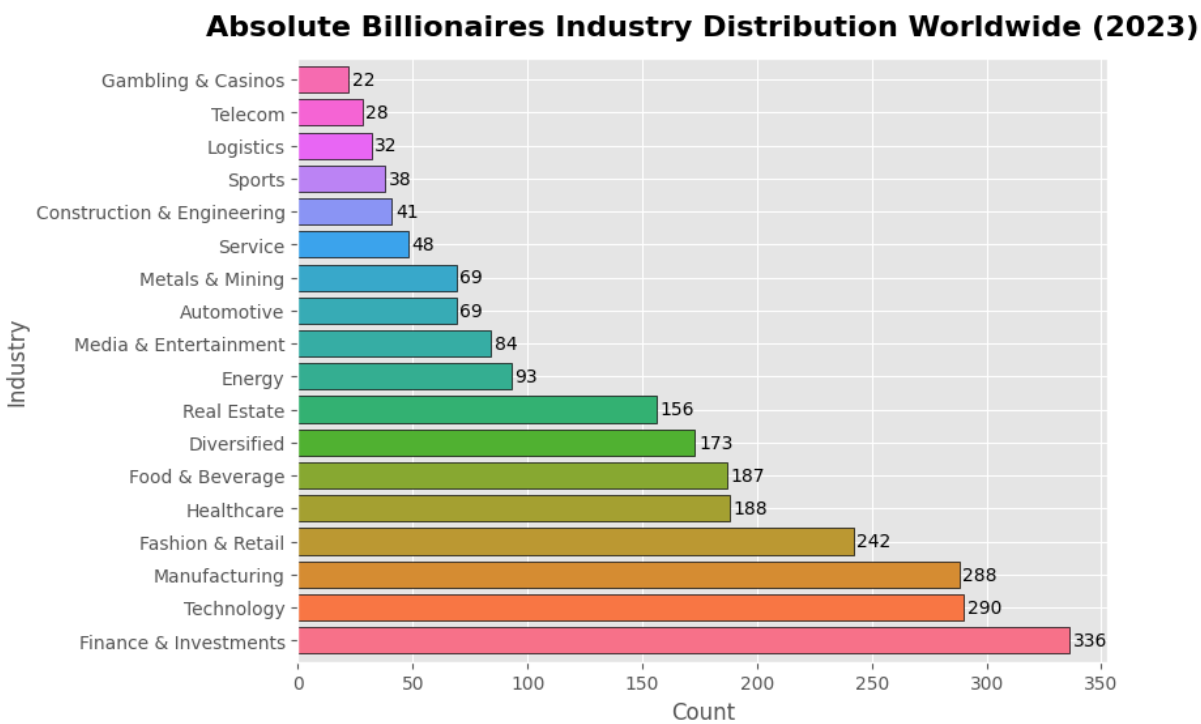
2.4 Feature: Industries | BarH-Plot

In [53]:
```
industry_counts = df['industries'] \
    .value_counts().sort_values(ascending=False)
industry_counts
```

Out[53]:
```
industries
Finance & Investments        336
Technology                   290
Manufacturing                288
Fashion & Retail             242
Healthcare                   188
Food & Beverage              187
Diversified                  173
Real Estate                  156
Energy                        93
Media & Entertainment         84
Automotive                    69
Metals & Mining               69
Service                       48
Construction & Engineering    41
Sports                        38
Logistics                     32
Telecom                       28
Gambling & Casinos            22
Name: count, dtype: int64
```

In [54]:
```
# Use a seaborn color palette
colors = sns.color_palette('husl', len(industry_counts))

# Create the horizontal bar plot
barh_plot_2_4 = industry_counts.plot(kind='barh', figsize=(8, 6), color=colors, edg

plt.title('Absolute Billionaires Industry Distribution Worldwide (2023)',
          fontsize=16,
          fontweight='bold',
          y=1.02)
```

```python
barh_plot_2_4.set_xlabel('Count')
barh_plot_2_4.set_ylabel('Industry')

for i, count in enumerate(industry_counts):
    plt.text(count + 1.5, i, str(count), ha='left', va='center')

file_name_2_4 = "Absolute_Billionaires_Industry_Distribution_Worldwide_U.png"
all_plots.append((barh_plot_2_4, file_name_2_4))
plt.show()
```

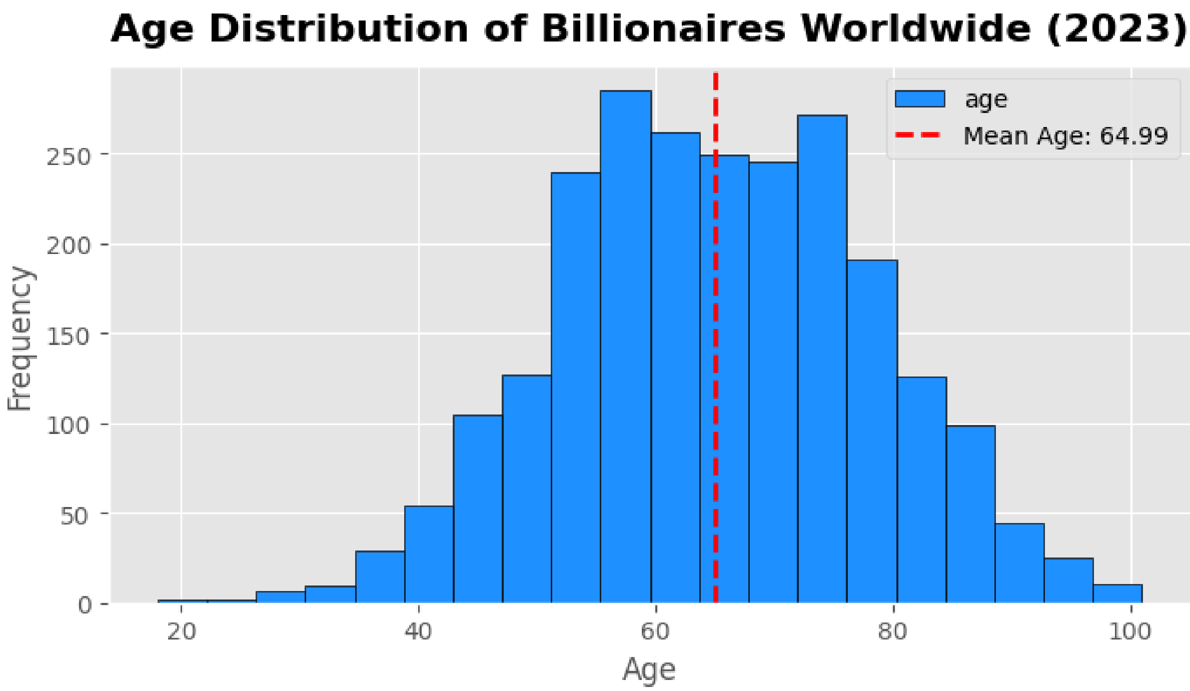


2.5 Feature: Age | Hist-Chart

In [55]:

```python
# Create the histogram plot
hist_chart_2_5 = df['age'].plot(kind='hist',
                                bins=20,
                                figsize=(8, 4),
                                color='dodgerblue',
                                edgecolor='black')

# Customize the plot
hist_chart_2_5.set_title('Age Distribution of Billionaires Worldwide (2023)',
                         fontsize=16,
                         fontweight='bold',
                         y=1.02)
hist_chart_2_5.set_xlabel('Age')

# Add vertical line for mean age
mean_age = df['age'].mean()
hist_chart_2_5.axvline(mean_age,
                       color='red',
                       linestyle='dashed',
```

```
                       linewidth=2,
                       label=f'Mean Age: {mean_age:.2f}')
hist_chart_2_5.legend()

file_name_2_5 = "Age_Distribution_of_Billionaires_Worldwide_U.png"
all_plots.append((hist_chart_2_5, file_name_2_5))
plt.show()
```



2.6 Feature: Gender | Pie-Chart

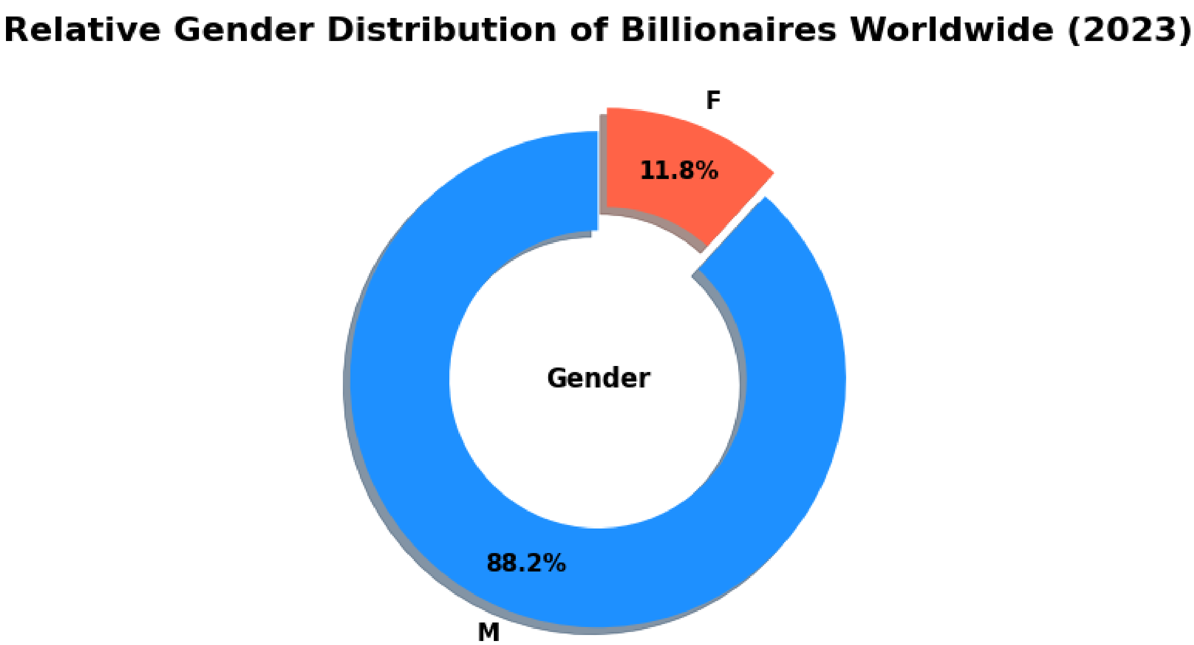
In [56]:
```
def create_pie_chart(data_val_counts, col_palette, input_title, feature_name):
    ax = data_val_counts.plot(kind='pie',
                              autopct='%1.1f%%',
                              colors=col_palette,
                              wedgeprops=dict(width=0.4),
                              textprops=dict(fontsize=11, fontweight='bold'),
                              startangle=90,
                              shadow=True,
                              explode=(0, 0.1),
                              pctdistance=0.8
                              )

    plt.text(0, 0, feature_name, ha='center', va='center', fontsize=12, fontweight=

    # Customize the plot
    plt.title(input_title,
              fontweight='bold',
              fontsize=16,
              y=1.05)
    ax.set_ylabel('')

    # Equal aspect ratio ensures that pie is drawn as a circle.
```

```
    ax.axis('equal')

    # Save and show the plot
    plt.show()
    return ax
```

In [57]:
```
gender_counts = df['gender'].value_counts()
gender_counts
```

Out[57]:
```
gender
M    2103
F     281
Name: count, dtype: int64
```

In [58]:
```
colors = ['dodgerblue', 'tomato']
gender_title = "Relative Gender Distribution of Billionaires Worldwide (2023)"
file_name_5 = "Relative_Gender_Distribution_of_Billionaires_Worldwide.png"
pie_chart_gender = create_pie_chart(gender_counts, colors, gender_title, feature_na
file_name_2_6 = "Relative_Gender_Distribution_of_Billionaires_Worldwide_U.png"
all_plots.append((pie_chart_gender, file_name_2_6))
```



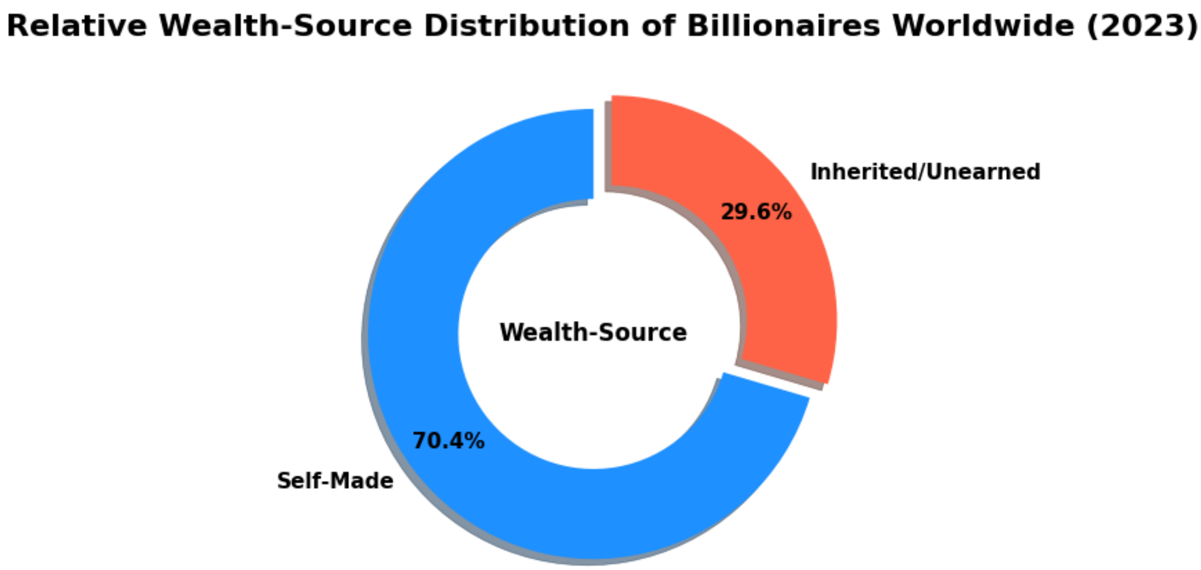
2.7 Feature: Wealth-Source | Pie-Chart

In [59]:
```
wealth_source_counts = df['wealth_source'].value_counts()
wealth_source_counts
```

Out[59]:
```
wealth_source
Self-Made              1679
Inherited/Unearned      705
Name: count, dtype: int64
```

In [60]:
```
wealth_source_title = "Relative Wealth-Source Distribution of Billionaires Worldwid
pie_chart_wealth_source = create_pie_chart(wealth_source_counts, colors, wealth_sou
```

```
                                        feature_name="Wealth-Source")
file_name_2_7 = "Relative_Wealth_Source_Distribution_of_Billionaires_Worldwide_U.pn
all_plots.append((pie_chart_wealth_source, file_name_2_7))
```



3. Multivariate Feature Relationships Analysis

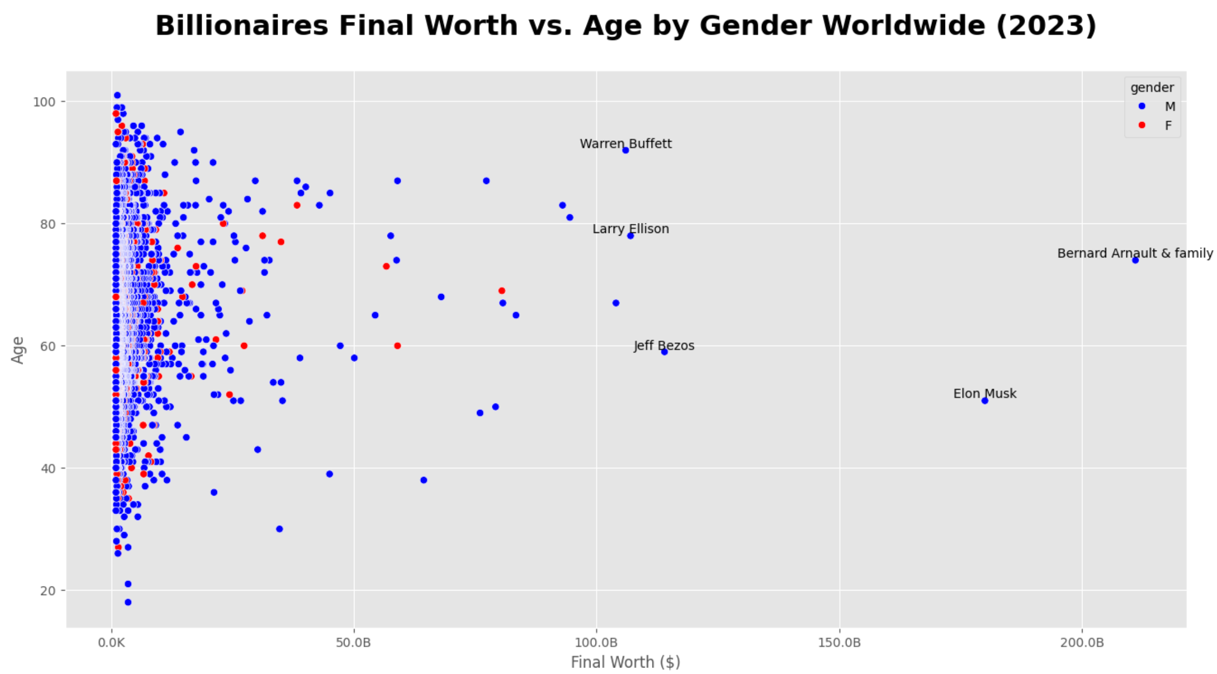
In this section we will go through the data analytics process of Multivariate Analysis to get a better understanding of the Feature-Relationships

To skip directly to to particular parts, use the following links:
3.1 **Final Worth vs. Age by Gender | Scatterplot**
3.2 **Billionaires Count by Top Countries Worldwide and Wealth Source | Histogram**
3.3 **Billionaires per Industry, Age and Gender | Violin Plot**
3.4 **International Billionaire Average Final Worth by Global Region and Industry | Facet Grid**
3.5 **Worldwide Billionaire Correlations between Economic and Demographic Indicators | Heatmap**

3.1 Final Worth vs. Age by Gender | Scatterplot

In [61]:
```
# Create the Scatterplot
plt.figure(figsize=(16, 8))
scatter_3_1 = sns.scatterplot(x='final_worth_usd',
                              y='age',
                              hue='gender',
                              palette={'M': 'blue', 'F': 'red'},
                              data=df)
plt.title('Billionaires Final Worth vs. Age by Gender Worldwide (2023)',
```

```
          fontweight='bold',
          fontsize=22,
          y=1.05)
scatter_3_1.set(xlabel='Final Worth ($)', ylabel='Age')

scatter_3_1.set_xticklabels([readable_numbers(x) for x in scatter_3_1.get_xticks()]

# Add text annotations for top 5 persons
top_5_billionaires = df.nlargest(5, 'final_worth_usd')
for _, person in top_5_billionaires.iterrows():
    plt.text(person['final_worth_usd'], person['age'], person['person_name'], fonts
             va='bottom')

file_name_3_1 = "Billionaires_Final_Worth_vs_Age_by_Gender_Worldwide_M.png"
all_plots.append((scatter_3_1, file_name_3_1))
plt.show()
```



3.2 Billionaires Count by Top Countries Worldwide and Wealth Source | Histogram

In [62]:
```
# Create a list of top 20 countries with most billionaires
top_countries = df['country'].value_counts().head(20).index.tolist()
top_countries
```

Out[62]:
```
['United States',
 'China',
 'India',
 'Germany',
 'Russia',
 'United Kingdom',
 'Switzerland',
 'Italy',
 'Singapore',
 'Australia',
 'Brazil',
 'Canada',
 'Japan',
 'France',
 'South Korea',
 'Sweden',
 'Turkey',
 'Thailand',
 'Indonesia',
 'Spain']
```

In [63]:
```
# Filter the df based on the top 20 countries criteria
df_top_countries = df[df['country'].isin(top_countries)][['country', 'wealth_source
df_top_countries
```

Out[63]:

	country	wealth_source
0	France	Inherited/Unearned
1	United States	Self-Made
2	United States	Self-Made
3	United States	Self-Made
4	United States	Self-Made
...	...	...
2378	China	Self-Made
2379	China	Self-Made
2380	United States	Inherited/Unearned
2381	China	Self-Made
2382	China	Self-Made

2174 rows × 2 columns

In [64]:
```
# Create a pivot table with 'country' as the index and 'wealth_source' as the colum
# Use 'size' as the aggregation function to count the occurrences of each combinati
# Fill NaN entries with 0 to represent countries where a certain wealth source cate
df_pivot_wealth_source = pd.pivot_table(df_top_countries,
                                        index="country",
                                        columns="wealth_source",
```

```
                                        aggfunc='size').fillna(0).reset_index()
# Create a new column 'total' by summing up the counts of 'Inherited/Unearned' and
df_pivot_wealth_source['total'] = (
        df_pivot_wealth_source['Inherited/Unearned'] + df_pivot_wealth_source['Self
).astype(int)
df_pivot_wealth_source.sort_values(by='total', ascending=False)
```

Out[64]:

wealth_source	country	Inherited/Unearned	Self-Made	total
19	United States	213.0	537.0	750
3	China	15.0	489.0	504
6	India	90.0	67.0	157
5	Germany	57.0	30.0	87
10	Russia	0.0	79.0	79
18	United Kingdom	22.0	56.0	78
15	Switzerland	30.0	42.0	72
8	Italy	31.0	24.0	55
11	Singapore	17.0	28.0	45
1	Brazil	26.0	17.0	43
0	Australia	14.0	29.0	43
2	Canada	11.0	28.0	39
9	Japan	8.0	28.0	36
4	France	19.0	15.0	34
12	South Korea	15.0	13.0	28
14	Sweden	13.0	13.0	26
16	Thailand	11.0	14.0	25
17	Turkey	11.0	14.0	25
7	Indonesia	10.0	14.0	24
13	Spain	14.0	10.0	24

In [65]:

```
# Merge the original DataFrame 'df_top_countries' with the pivot table 'df_pivot_we
# on the 'country' column, keeping only the columns 'country' and 'total' from the
# Sort the resulting DataFrame by the 'total' column in descending order
df_wealth_source_merged = pd.merge(df_top_countries, df_pivot_wealth_source[['count
                                   on='country').sort_values(by='total', ascending=
df_wealth_source_merged
```

Out[65]:

	country	wealth_source	total
1087	United States	Self-Made	750
448	United States	Self-Made	750
647	United States	Self-Made	750
648	United States	Inherited/Unearned	750
1522	United States	Self-Made	750
...	...	...	...
987	Spain	Inherited/Unearned	24
566	Indonesia	Self-Made	24
1948	Indonesia	Inherited/Unearned	24
555	Indonesia	Self-Made	24
49	Indonesia	Self-Made	24

2174 rows × 3 columns

In [66]:

```
df_top_country_counts = df_wealth_source_merged[['country', 'total']].drop_duplicat
    drop=True)
df_top_country_counts
```

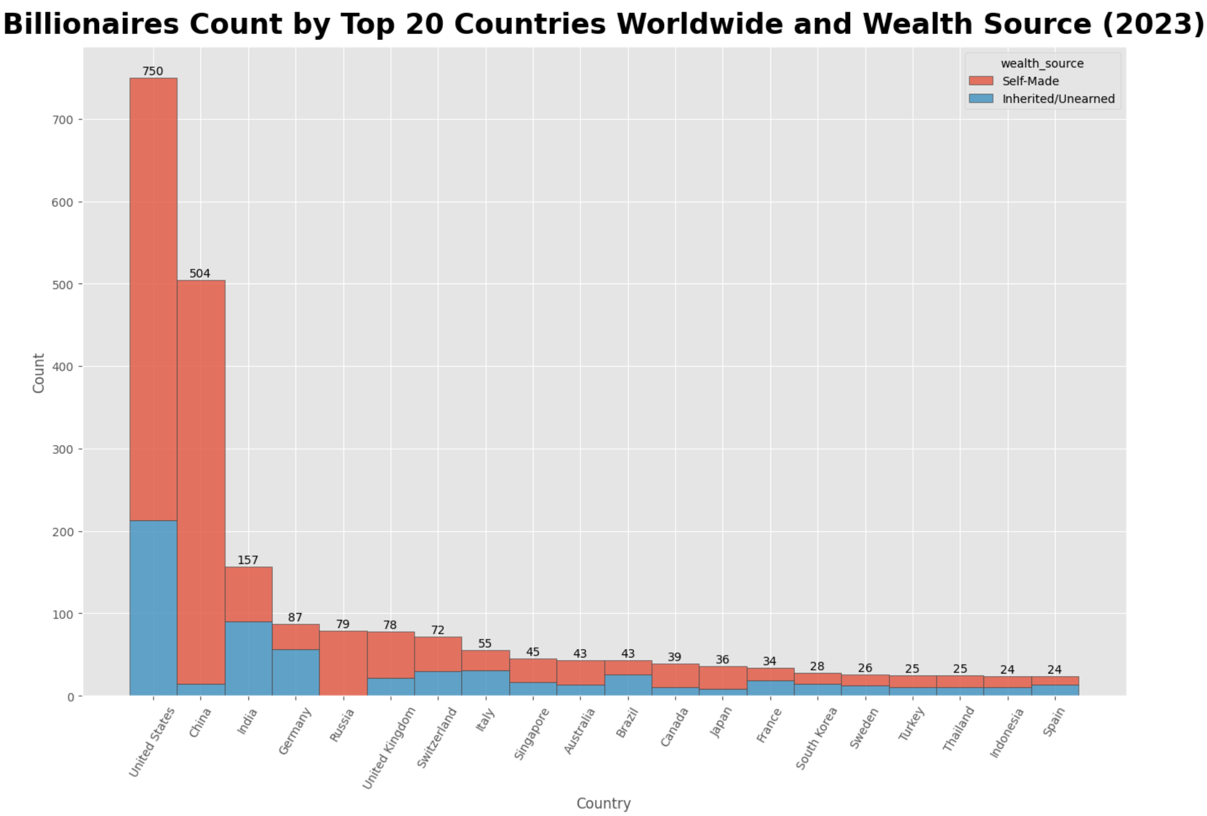
Out[66]:

	country	total
0	United States	750
1	China	504
2	India	157
3	Germany	87
4	Russia	79
5	United Kingdom	78
6	Switzerland	72
7	Italy	55
8	Singapore	45
9	Australia	43
10	Brazil	43
11	Canada	39
12	Japan	36
13	France	34
14	South Korea	28
15	Sweden	26
16	Turkey	25
17	Thailand	25
18	Indonesia	24
19	Spain	24

In [67]:

```
# Create the histplot
plt.figure(figsize=(16, 10))
hist_plot_3_2 = sns.histplot(
    data=df_wealth_source_merged,
    x="country",
    hue="wealth_source",
    edgecolor=".3",
    multiple='stack',
    linewidth=.5,
    stat='count'
)
plt.xticks(rotation=60)
hist_plot_3_2.set_title("Billionaires Count by Top 20 Countries Worldwide and Wealt
                        fontsize=24,
                        fontweight='bold',
                        y=1.01)
hist_plot_3_2.set_ylabel("Count")
```

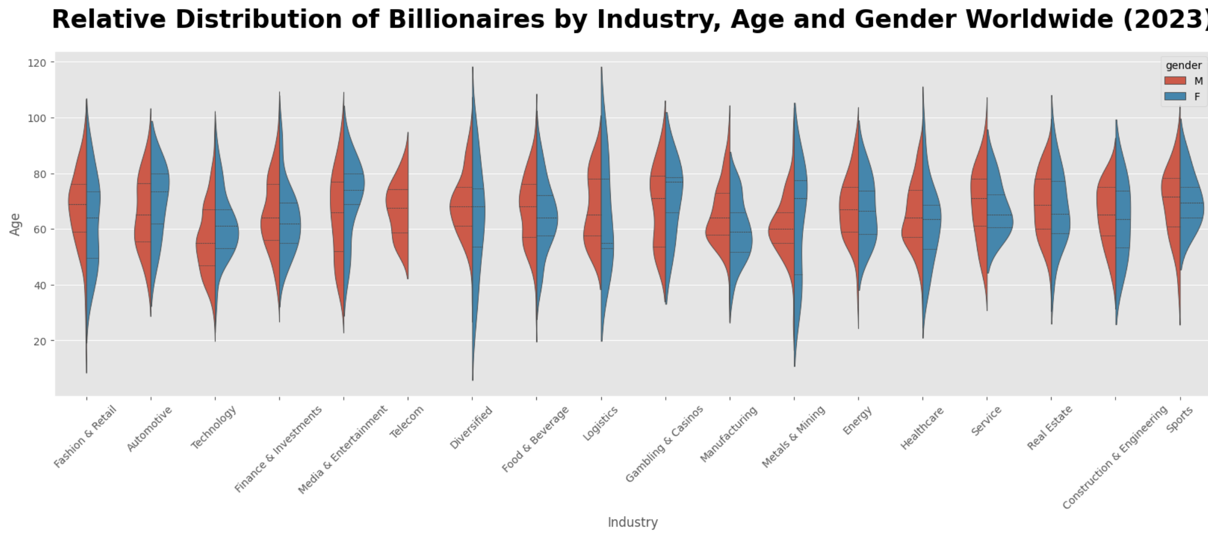
```
hist_plot_3_2.set_xlabel("Country")

# Add labels for total count
for index, row in df_top_country_counts.iterrows():
    country = row['country']
    total_count = row['total']
    plt.text(country, total_count + 1, str(total_count), ha='center', va='bottom')

file_name_3_2 = "Billionaires_Count_by_Top_20_Countries_Worldwide_and_Wealth_Source
all_plots.append((hist_plot_3_2, file_name_3_2))
plt.show()
```



3.3 Billionaires per Industry, Age and Gender | Violin Plot

In [68]:
```
# Create the Violinplot
plt.figure(figsize=(20, 6))
violin_plot_3_3 = sns.violinplot(data=df,
                                 x="industries",
                                 y="age",
                                 hue='gender',
                                 inner="quart",
                                 split=True)
violin_plot_3_3.set_title("Relative Distribution of Billionaires by Industry, Age a
                          y=1.05,
                          fontsize=24,
                          fontweight="bold")
violin_plot_3_3.set_ylabel("Age")
```

```
violin_plot_3_3.set_xlabel("Industry")
plt.xticks(rotation=45)

file_name_3_3 = "Relative_Distribution_of_Billionaires_by_Industry_Age_and_Gender_W
all_plots.append((violin_plot_3_3, file_name_3_3))
plt.show()
```



3.4 International Billionaire Average Final Worth by Global Region and Industry | Facet Grid

In [69]:
```
# Dictionary mapping countries to their respective global regions
country_to_global_region_dict = {
    'France': 'West Europe',
    'United States': 'North America',
    'Mexico': 'North America',
    'India': 'Asia',
    'Spain': 'West Europe',
    'China': 'Asia',
    'Canada': 'North America',
    'Germany': 'West Europe',
    'Switzerland': 'West Europe',
    'Belgium': 'West Europe',
    'Hong Kong': 'Asia',
    'Austria': 'West Europe',
    'Japan': 'Asia',
    'United Kingdom': 'West Europe',
    'Australia': 'Oceania',
    'Indonesia': 'Asia',
    'United Arab Emirates': 'Asia',
    'Russia': 'East Europe',
    'Chile': 'South America',
    'Monaco': 'West Europe',
    'Czech Republic': 'East Europe',
    'Sweden': 'West Europe',
    'Thailand': 'Asia',
    'Uzbekistan': 'Asia',
    'Singapore': 'Asia',
```

```
    'Nigeria': 'Africa',
    'Israel': 'Asia',
    'Italy': 'West Europe',
    'South Africa': 'Africa',
    'Brazil': 'South America',
    'Malaysia': 'Asia',
    'South Korea': 'Asia',
    'New Zealand': 'Oceania',
    'Philippines': 'Asia',
    'Taiwan': 'Asia',
    'Norway': 'West Europe',
    'Egypt': 'Africa',
    'Denmark': 'West Europe',
    'Eswatini (Swaziland)': 'Africa',
    'Colombia': 'South America',
    'Netherlands': 'West Europe',
    'Poland': 'East Europe',
    'Bahamas': 'North America',
    'Ukraine': 'East Europe',
    'Cayman Islands': 'North America',
    'Greece': 'West Europe',
    'Turkey': 'Asia',
    'Argentina': 'South America',
    'Georgia': 'East Europe',
    'Portugal': 'West Europe',
    'Kazakhstan': 'Asia',
    'Algeria': 'Africa',
    'Vietnam': 'Asia',
    'Latvia': 'East Europe',
    'Finland': 'West Europe',
    'Bermuda': 'North America',
    'Luxembourg': 'West Europe',
    'British Virgin Islands': 'North America',
    'Cambodia': 'Asia',
    'Lebanon': 'Asia',
    'Oman': 'Asia',
    'Ireland': 'West Europe',
    'Cyprus': 'Asia',
    'Guernsey': 'West Europe',
    'Liechtenstein': 'West Europe',
    'Turks and Caicos Islands': 'North America',
    'Romania': 'East Europe',
    'Qatar': 'Asia',
    'Uruguay': 'South America',
    'Nepal': 'Asia',
    'Slovakia': 'East Europe',
    'Morocco': 'Africa',
    'Hungary': 'East Europe',
    'Tanzania': 'Africa',
    'Bahrain': 'Asia',
    'Peru': 'South America',
    'Andorra': 'West Europe',
    'Armenia': 'East Europe',
    'NaN': 'Unknown'
}
```

In [70]:
```
# Create a new column 'global_region' by mapping the 'country' column using the pro
df['global_region'] = df['country'].replace(country_to_global_region_dict)
# Verify that the replacement worked for every country
df.query('country == global_region')
```

Out[70]:

rank	final_worth_usd	person_name	age	country	country_of_citizenship	city	industrie

In [71]:
```
# Group the DataFrame by 'industries' and 'global_region', calculate the mean of 'f
avg_final_worth_per_global_region = df.groupby(['industries', 'global_region'])['fi
avg_final_worth_per_global_region
```

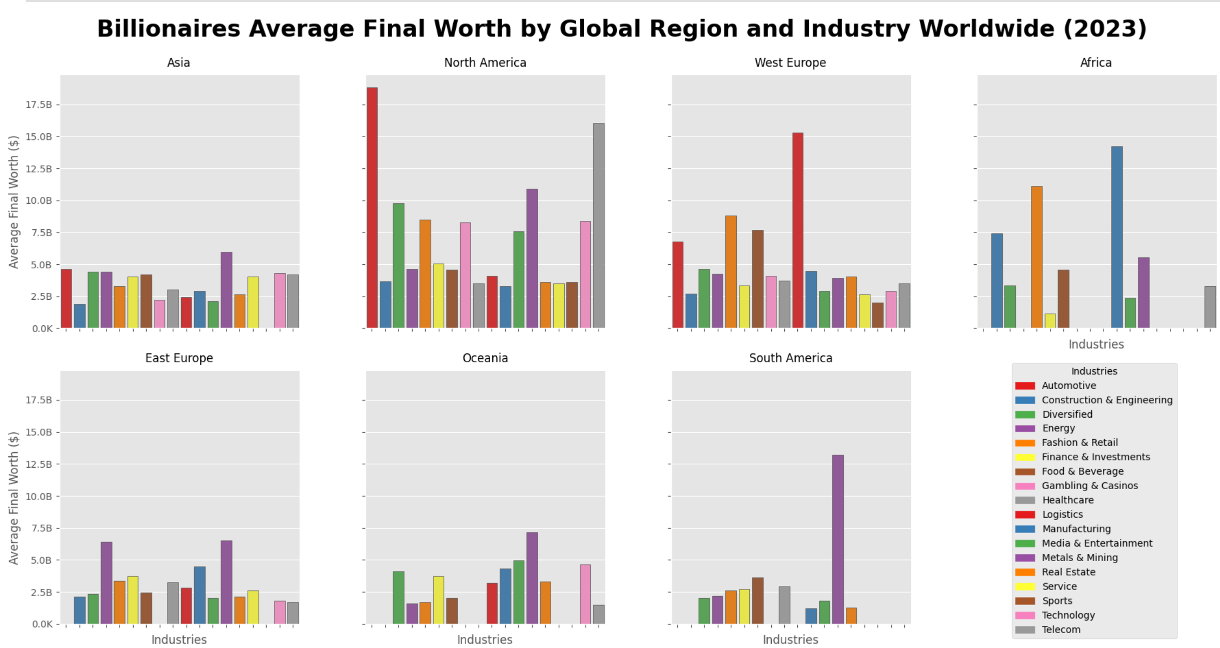
Out[71]:

	industries	global_region	final_worth_usd
0	Automotive	Asia	4.629545e+09
1	Automotive	North America	1.882500e+10
2	Automotive	West Europe	6.792308e+09
3	Construction & Engineering	Africa	7.400000e+09
4	Construction & Engineering	Asia	1.876923e+09
...	...	...	...
94	Telecom	Asia	4.200000e+09
95	Telecom	East Europe	1.700000e+09
96	Telecom	North America	1.602857e+10
97	Telecom	Oceania	1.500000e+09
98	Telecom	West Europe	3.500000e+09

99 rows × 3 columns

In [72]:
```
# Create a FacetGrid for each global region with a bar plot for average final worth
facet_grid_3_4 = sns.FacetGrid(data=avg_final_worth_per_global_region,
                               col='global_region',
                               col_wrap=4,
                               height=4.5,
                               hue='industries',
                               palette='Set1'
                               )


# Define a function to set individual subplot titles
def set_title(col_name):
    plt.gca().set_title(col_name)


# Map a bar plot for each industry by global region
facet_grid_3_4.map(sns.barplot,
                   'industries', 'final_worth_usd',
```

```
                    edgecolor=".3",
                    order=avg_final_worth_per_global_region['industries'].unique()
                    ).set_titles("{col_name}", verticalalignment='bottom')

facet_grid_3_4.set_xticklabels(rotation=45)
facet_grid_3_4.set_axis_labels('Industries', 'Average Final Worth ($)')

# Adjust y-axis tick labels
plt.gca().set_yticklabels([readable_numbers(y) for y in plt.gca().get_yticks()])

# Create custom legend handles
handles = [mpatches.Patch(color=color,
                         label=label) for color, label in
           zip(sns.color_palette('Set1', n_colors=18), avg_final_worth_per_global_r

# Add legend and adjust layout
plt.legend(handles=handles, title='Industries', bbox_to_anchor=(1.4, 1.05), loc='up

# Set overall title
title = plt.suptitle("Billionaires Average Final Worth by Global Region and Industr
                     y=1.05,
                     fontweight='bold',
                     fontsize=24,
                     )

# Set individual subplot titles
facet_grid_3_4.set_titles(col_template="{col_name}", row_template="{row_name}", ver

file_name_3_4 = "Billionaires_Average_Final_Worth_by_Global_Region_and_Industry_Wor
all_plots.append((facet_grid_3_4, file_name_3_4))
plt.show()
```



3.5 Worldwide Billionaire Correlations between Economic and Demographic Indicators | Heatmap

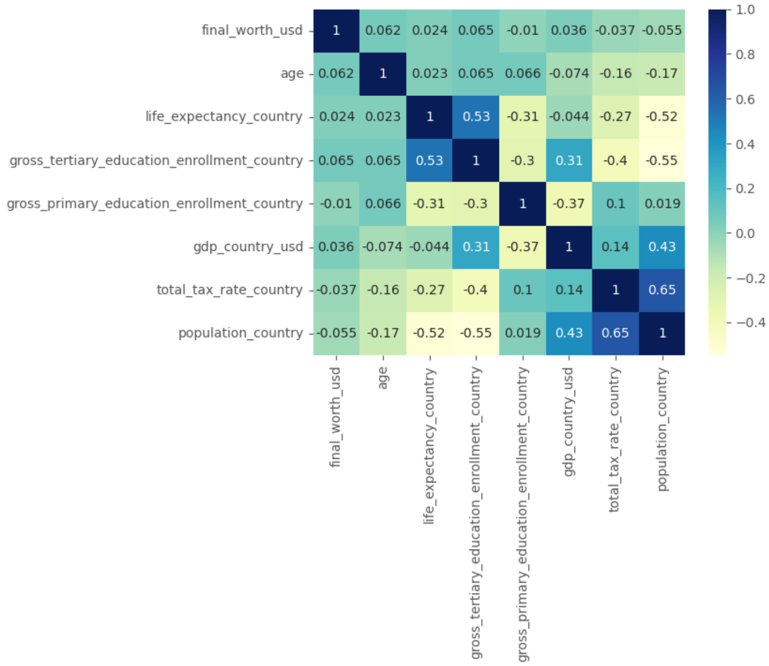
In [73]:
```
# Select numerical columns excluding the 'rank' column and calculate the correlatio
corr_df = df.select_dtypes(exclude='object').drop('rank', axis=1).corr()
corr_df
```

Out[73]:

	final_worth_usd	age	life_expectancy_cou
final_worth_usd	1.000000	0.061611	0.02
age	0.061611	1.000000	0.02
life_expectancy_country	0.024380	0.023309	1.00
gross_tertiary_education_enrollment_country	0.065300	0.064680	0.52
gross_primary_education_enrollment_country	-0.010040	0.066280	-0.31
gdp_country_usd	0.035914	-0.074021	-0.04
total_tax_rate_country	-0.037038	-0.157074	-0.26
population_country	-0.054723	-0.174421	-0.52

In [74]:
```
# Create a heatmap of the correlation matrix
heatmap_3_5 = sns.heatmap(data=corr_df,
                          annot=True,
                          cmap='YlGnBu')
heatmap_3_5.set_title("Billionaires Correlations between Economic and Demographic I
                      y=1.05,
                      fontweight='bold',
                      fontsize=18)
file_name_3_5 = "Billionaires_Correlations_between_Economic_and_Demographic_Indicat
all_plots.append((heatmap_3_5, file_name_3_5))
plt.show()
```

Billionaires Correlations between Economic and Demographic Indicators Worldwide (2023)



```
In [75]: # Iterate through all_plots, where each element is a tuple containing a figure (fig
for fig, file_name in all_plots:
    # Define the base directory for media files
    media_dir = "media/"

    # Check if the file_name ends with "U" (indicating univariate analysis) or not
    if file_name.strip(" .png").endswith("U"):
        media_dir += "univariate_analysis"
    else:
        media_dir += "multivariate_analysis"

    # Create the directory if it doesn't exist
    os.makedirs(media_dir, exist_ok=True)

    # Save the figure to the appropriate directory with a tight bounding box
    fig.figure.savefig(f"{media_dir}/{file_name}", bbox_inches='tight')
```

4. Conclusion - Key Findings

- **Wealth Distribution:** The dataset includes 2,384 billionaires with a mean net worth of 4.77 billion (USD), ranging from 1 billion to 211 billion (USD). The top 5 industries dominating the billionaire landscape are Finance & Investments, Technology, Manufacturing, Fashion & Retail, and Healthcare, collectively representing approximately 56.38% of the total billionaires in the dataset

- **Age Distribution:** The age distribution of billionaires ranges from 18 to 101 years, with a mean age of 64.99 years. The majority of billionaires fall between the ages of 56 and 74.

- **Gender Disparity:** Male billionaires significantly outnumber female billionaires, constituting approximately 88.21% of the total billionaires in the dataset.

- **Self-Made Success:** The majority of billionaires are self-made, representing approximately 70.42% of the total billionaires in the dataset.

- **Top 5 Countries and Wealth Source Distribution:**

  - **United States:** 213 Inherited/Unearned billionaires, 537 Self-Made billionaires, totaling 750 billionaires.
  - **China:** 15 Inherited/Unearned billionaires, 489 Self-Made billionaires, totaling 504 billionaires.
  - **India:** 90 Inherited/Unearned billionaires, 67 Self-Made billionaires, totaling 157 billionaires.
  - **Germany:** 57 Inherited/Unearned billionaires, 30 Self-Made billionaires, totaling 87 billionaires.
  - **Russia:** 0 Inherited/Unearned billionaires, 79 Self-Made billionaires, totaling 79 billionaires.
- **Correlation Analysis:** Examining numerical correlations reveals some interesting insights:

  - **Net Worth:**

    - *Positive Correlation with Tertiary Education Enrollment and Age:* Higher net worth tends to correlate with regions where there is a higher percentage of the population enrolled in tertiary education. Additionally, there is a slight positive correlation with age, suggesting that older individuals may accumulate more wealth.
    - *Negative Correlation with Country Population and Total Tax Rate:* The negative correlation with country population indicates that billionaires might be more concentrated in less populous countries. The negative correlation with the total tax rate suggests that billionaires tend to accumulate more wealth in countries with lower total tax rates.
  - **Age:**

    - *Negative Correlation with Country Population and Total Tax Rate:* The negative correlation with country population suggests that older billionaires may prefer residing in less populated areas. The negative correlation with the total tax rate implies that older billionaires might choose countries with lower tax burdens.
  - **Life Expectancy in a Country:**

    - *Positive Correlation with Gross Tertiary Education Enrollment:* Countries with higher life expectancies tend to have a higher percentage of the population enrolled in tertiary education. This could indicate a positive correlation between educational opportunities and overall well-being.

    - *Negative Correlation with Country Population and Total Tax Rate:* Higher life expectancy correlates with lower country population and lower total tax rates, suggesting a potential preference for less crowded and fiscally favorable environments.
- **Gross Tertiary Education Enrollment:**
    - *Positive Correlation with GDP:* Countries with a higher percentage of the population enrolled in tertiary education tend to have higher Gross Domestic Product (GDP). This could signify a positive relationship between educational investment and economic prosperity.
    - *Negative Correlation with Country Population and Total Tax Rate:* The negative correlation with country population indicates that regions with more tertiary education enrollment might have a lower population density. The negative correlation with the total tax rate suggests that these regions may have lower tax burdens, potentially attracting individuals seeking financial opportunities.