PHASE 3 PROJECT

SUBMISSION

- **Student name: LESLEY KAMAMO**
- **Student pace: PART TIME**
- **Scheduled project review date/time: 23RD OCTOBER 2023**
- **Instructor name: SAMUEL KARU**

Overview

Financial freedom is what everyone in the employment industry aims to achieve. I tend to think the base for this is in the education system. More than the need to acquire knowledge, we go through the education system so as to give back to the community through work/employment. For the skilss learnt, we aim that one point in time, it will yield to financial freedom in whatever industry and phase we choose to place ourselves in.

Soar High Research Agency is an non-governmental organization that deals with social development. They are looking for an industry to invest that will yield return even in the future. Based on their source question, we are required to provide insights on the wealth distribution status and industries they are to investment in.

For this project, we are going to be solving a classification problem using the Billionaire Statistis dataset chosen.

Business Problem

Objectives

The objectives of this project based on the dataset chosen is to find out:

1. **The Wealth Status of Billionaires (whether self-made or not)**
2. **What industry/sources is more inclined to produce billionaires?**
3. **Demographic analysis of billionaires (age, gender, country)**
4. **Provide classification to the wealth status of billionaires based on the features.**
5. **Provide insights into which industry are likely to produce billionaires in future (logistic regression)**

Data Understanding

For this project, we will use the data under the file path `data/...`. This folder contains the information on billionaires. The dataset represents historical data on billionaires for recent past years, hence this data will be modified for the purpose of the analysis.

Data Description

The data is contained in a CSV file:

1. `Billionaires Statistics Dataset.csv`**: each record represents rank,finalWorth,category,personName,age,country,source, selfMade,status,gender,birthDate,title,residenceStateRegion,birthYear, tax_revenue_country_country,total_tax_rate_country,population_country among other fields.**

The columns for the dataset represented for this analysis will be:

`rank`**: The ranking of the billionaire in terms of wealth.**

`finalWorth`**: The final net worth of the billionaire in U.S. dollars.**

`category`: **The category or industry in which the billionaire's business operates.**

`age`: **The age of the billionaire.**

`country`: **The country in which the billionaire resides.**

`source`: **The source of the billionaire's wealth.**

`industries`: **The industries associated with the billionaire's business interests.**

`countryOfCitizenship`: **The country of citizenship of the billionaire.**

`organization`: **The name of the organization or company associated with the billionaire.**

`selfMade`: **Indicates whether the billionaire is self-made (True/False).**

`status`: **"D" represents self-made billionaires (Founders/Entrepreneurs) and "U" indicates inherited or unearned wealth.**

`gender`: **The gender of the billionaire.**

`title`: **The title or honorific of the billionaire.**

`state`: **The state in which the billionaire resides.**

`population_country`: **Population of the billionaire's country.**

The data from this file as seen above represents various column findings. Such kind of findings will be available for the analyisis and insights into what to recommend to the Businees Stakeholders.

Data Preparation

This step includes importing all the standard packages to be used for the purpose of analysis

In the cell below, we will:

- **Import and alias** `pandas` **as** `pd`
- **Import and alias** `numpy` **as** `np`
- **Import and alias** `seaborn` **as** `sns`
- **Import and alias** `matplotlib.pyplot` **as** `plt`
- **Set Matplotlib visualizations to display inline in the notebook**

In [1]:

```
# import all

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

%matplotlib inline
```

Load the Dataset

Open the csv file as a Dataframe

In [2]:

```
# load the dataset as `billionaire_df`
billionaire_df = pd.read_csv("data/Billionaires Statistics Dataset.csv", index_col=0)

billionaire_df
```

Out[2]:

rank	finalWorth	category	personName	age	country	city	source	industries	countryOfCitizenship
1	211000	Fashion & Retail	Bernard Arnault & family	74.0	France	Paris	LVMH	Fashion & Retail	France
2	180000	Automotive	Elon Musk	51.0	United States	Austin	Tesla, SpaceX	Automotive	United States
3	114000	Technology	Jeff Bezos	59.0	United States	Medina	Amazon	Technology	United States
4	107000	Technology	Larry Ellison	78.0	United States	Lanai	Oracle	Technology	United States
5	106000	Finance & Investments	Warren Buffett	92.0	United States	Omaha	Berkshire Hathaway	Finance & Investments	United States
...	...	...	...	...	...	...	...	...	...
2540	1000	Healthcare	Yu Rong	51.0	China	Shanghai	Health clinics	Healthcare	China
2540	1000	Food & Beverage	Richard Yuengling, Jr.	80.0	United States	Pottsville	Beer	Food & Beverage	United States
2540	1000	Manufacturing	Zhang Gongyun	60.0	China	Gaomi	Tyre manufacturing machinery	Manufacturing	China
2540	1000	Real Estate	Zhang Guiping & family	71.0	China	Nanjing	Real estate	Real Estate	China
2540	1000	Diversified	Inigo Zobel	66.0	Philippines	Makati	Diversified	Diversified	Philippines

2640 rows × 34 columns

In [3]:

```
# find the shape of the dataset
billionaire_df.shape
```

Out[3]:

```
(2640, 34)
```

The given millionaore dataset has 2640 rows and 34 columns.

In [4]:

```
billionaire_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2640 entries, 1 to 2540
Data columns (total 34 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   finalWorth                               2640 non-null   int64
 1   category                                 2640 non-null   object
 2   personName                               2640 non-null   object
 3   age                                      2575 non-null   float64
 4   country                                  2602 non-null   object
 5   city                                     2568 non-null   object
 6   source                                   2640 non-null   object
 7   industries                               2640 non-null   object
 8   countryOfCitizenship                     2640 non-null   object
 9   organization                             325 non-null    object
 10  selfMade                                 2640 non-null   bool
 11  status                                   2640 non-null   object
```

```
 11  status                                    2640 non-null   object
 12  gender                                    2640 non-null   object
 13  birthDate                                 2564 non-null   object
 14  lastName                                  2640 non-null   object
 15  firstName                                 2637 non-null   object
 16  title                                     339 non-null    object
 17  date                                      2640 non-null   object
 18  state                                     753 non-null    object
 19  residenceStateRegion                      747 non-null    object
 20  birthYear                                 2564 non-null   float64
 21  birthMonth                                2564 non-null   float64
 22  birthDay                                  2564 non-null   float64
 23  cpi_country                               2456 non-null   float64
 24  cpi_change_country                        2456 non-null   float64
 25  gdp_country                               2476 non-null   object
 26  gross_tertiary_education_enrollment       2458 non-null   float64
 27  gross_primary_education_enrollment_country 2459 non-null  float64
 28  life_expectancy_country                   2458 non-null   float64
 29  tax_revenue_country_country               2457 non-null   float64
 30  total_tax_rate_country                    2458 non-null   float64
 31  population_country                        2476 non-null   float64
 32  latitude_country                          2476 non-null   float64
 33  longitude_country                         2476 non-null   float64
dtypes: bool(1), float64(14), int64(1), object(18)
memory usage: 703.8+ KB
```

From the above, the billionaire dataset ideally should have 2640 records across 34 columns, which means ideally each column should have 2640 values.

In [5]:

```
# displaying the first 5 entries
billionaire_df.head()
```

Out[5]:

rank	finalWorth	category	personName	age	country	city	source	industries	countryOfCitizenship	organization
1	211000	Fashion & Retail	Bernard Arnault & family	74.0	France	Paris	LVMH	Fashion & Retail	France	LVMH Moët Hennessy Louis Vuitton
2	180000	Automotive	Elon Musk	51.0	United States	Austin	Tesla, SpaceX	Automotive	United States	Tesla
3	114000	Technology	Jeff Bezos	59.0	United States	Medina	Amazon	Technology	United States	Amazon
4	107000	Technology	Larry Ellison	78.0	United States	Lanai	Oracle	Technology	United States	Oracle
5	106000	Finance & Investments	Warren Buffett	92.0	United States	Omaha	Berkshire Hathaway	Finance & Investments	United States	Berkshire Hathaway Inc. (Cl A)

5 rows × 34 columns

In [6]:

```
# displaying the last 5 records
billionaire_df.tail()
```

Out[6]:

rank	finalWorth	category	personName	age	country	city	source	industries	countryOfCitizenship
2540	1000	Healthcare	Yu Rong	51.0	China	Shanghai	Health clinics	Healthcare	China

	finalWorth	category	personName	age	country	city	source	industries	countryOfCitizenship
2540	1000	Food & Beverage	Richard Yuengling, Jr.	80.0	United States	Pottsville	Beer	Food & Beverage	United States
2540	1000	Manufacturing	Zhang Gongyun	60.0	China	Gaomi	Tyre manufacturing machinery	Manufacturing	China
2540	1000	Real Estate	Zhang Guiping & family	71.0	China	Nanjing	Real estate	Real Estate	China
2540	1000	Diversified	Inigo Zobel	66.0	Philippines	Makati	Diversified	Diversified	Philippines

5 rows × 34 columns

Handling Missing Values

In [7]:

```
# check for missing values
billionaire_df.isnull().sum()
```

Out[7]:

```
finalWorth                                    0
category                                      0
personName                                    0
age                                          65
country                                      38
city                                         72
source                                        0
industries                                    0
countryOfCitizenship                          0
organization                               2315
selfMade                                      0
status                                        0
gender                                        0
birthDate                                    76
lastName                                      0
firstName                                     3
title                                      2301
date                                          0
state                                      1887
residenceStateRegion                       1893
birthYear                                    76
birthMonth                                   76
birthDay                                     76
cpi_country                                 184
cpi_change_country                          184
gdp_country                                 164
gross_tertiary_education_enrollment         182
gross_primary_education_enrollment_country  181
life_expectancy_country                     182
tax_revenue_country_country                 183
total_tax_rate_country                      182
population_country                          164
latitude_country                            164
longitude_country                           164
dtype: int64
```

In [8]:

```
billionaire_df['gdp_country'].isnull().sum()
```

Out[8]:

```
164
```

In [9]:

```
# replace the missing age values with the next age value
```

```
billionaire_df['age'].fillna(method='ffill', inplace=True)

#replace the country billionaire resides with Not Known
billionaire_df['country'].fillna("Not Known", inplace=True)

# replace the latitude and longitude  with 0
billionaire_df['latitude_country'].fillna(0, inplace=True)
billionaire_df['longitude_country'].fillna(0, inplace=True)

# replace the missing values with the mean rate
billionaire_df['gross_tertiary_education_enrollment'].fillna(billionaire_df['gross_tertia
ry_education_enrollment'].mean(), inplace=True)
billionaire_df['cpi_country'].fillna(billionaire_df['cpi_country'].mean(), inplace=True)
billionaire_df['tax_revenue_country_country'].fillna(billionaire_df['tax_revenue_country_
country'].mean(), inplace=True)
billionaire_df['total_tax_rate_country'].fillna(billionaire_df['total_tax_rate_country'].
mean(), inplace=True)
billionaire_df['life_expectancy_country'].fillna(billionaire_df['life_expectancy_country'
].mean(), inplace=True)
billionaire_df['population_country'].fillna(billionaire_df['population_country'].mean(),
inplace=True)
```

In [10]:

```
billionaire_df.columns
```

Out[10]:

```
Index(['finalWorth', 'category', 'personName', 'age', 'country', 'city',
       'source', 'industries', 'countryOfCitizenship', 'organization',
       'selfMade', 'status', 'gender', 'birthDate', 'lastName', 'firstName',
       'title', 'date', 'state', 'residenceStateRegion', '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'],
      dtype='object')
```

In [11]:

```
billionaire_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2640 entries, 1 to 2540
Data columns (total 34 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   finalWorth                                2640 non-null   int64
 1   category                                  2640 non-null   object
 2   personName                                2640 non-null   object
 3   age                                       2640 non-null   float64
 4   country                                   2640 non-null   object
 5   city                                      2568 non-null   object
 6   source                                    2640 non-null   object
 7   industries                                2640 non-null   object
 8   countryOfCitizenship                      2640 non-null   object
 9   organization                              325 non-null    object
 10  selfMade                                  2640 non-null   bool
 11  status                                    2640 non-null   object
 12  gender                                    2640 non-null   object
 13  birthDate                                 2564 non-null   object
 14  lastName                                  2640 non-null   object
 15  firstName                                 2637 non-null   object
 16  title                                     339 non-null    object
 17  date                                      2640 non-null   object
 18  state                                     753 non-null    object
 19  residenceStateRegion                      747 non-null    object
 20  birthYear                                 2564 non-null   float64
 21  birthMonth                                2564 non-null   float64
 22  birthDay                                  2564 non-null   float64
 23  cpi_country                               2640 non-null   float64
```

```
 24  cpi_change_country                           2456 non-null   float64
 25  gdp_country                                  2476 non-null   object
 26  gross_tertiary_education_enrollment          2640 non-null   float64
 27  gross_primary_education_enrollment_country   2459 non-null   float64
 28  life_expectancy_country                      2640 non-null   float64
 29  tax_revenue_country_country                  2640 non-null   float64
 30  total_tax_rate_country                       2640 non-null   float64
 31  population_country                           2640 non-null   float64
 32  latitude_country                             2640 non-null   float64
 33  longitude_country                            2640 non-null   float64
dtypes: bool(1), float64(14), int64(1), object(18)
memory usage: 703.8+ KB
```

In [12]:

```
billionaire_df.describe()
```

Out[12]:

	finalWorth	age	birthYear	birthMonth	birthDay	cpi_country	cpi_change_country	gross_tertiary_ed
count	2640.000000	2640.000000	2564.000000	2564.000000	2564.000000	2640.000000	2456.000000	
mean	4623.787879	65.038636	1957.183307	5.740250	12.099844	127.755204	4.364169	
std	9834.240939	13.252811	13.282516	3.710085	9.918876	25.514096	3.623763	
min	1000.000000	18.000000	1921.000000	1.000000	1.000000	99.550000	-1.900000	
25%	1500.000000	56.000000	1948.000000	2.000000	1.000000	117.240000	1.700000	
50%	2300.000000	65.000000	1957.000000	6.000000	11.000000	117.240000	2.900000	
75%	4200.000000	75.000000	1966.000000	9.000000	21.000000	125.080000	7.500000	
max	211000.000000	101.000000	2004.000000	12.000000	31.000000	288.570000	53.500000	

In [13]:

```
#validating the data type of finalWorth -- int
billionaire_df['finalWorth'].dtype
```

Out[13]:

```
dtype('int64')
```

--- Retaining relevant columns for data exploration

In [14]:

```
# drop the irrelevant columns
billionaire_df = billionaire_df.drop(columns=['category'])

# keep only columns with no missing values
billionaire_df = billionaire_df.dropna(axis=1)
```

In [15]:

```
billionaire_df.columns
```

Out[15]:

```
Index(['finalWorth', 'personName', 'age', 'country', 'source', 'industries',
       'countryOfCitizenship', 'selfMade', 'status', 'gender', 'lastName',
       'date', 'cpi_country', 'gross_tertiary_education_enrollment',
       'life_expectancy_country', 'tax_revenue_country_country',
       'total_tax_rate_country', 'population_country', 'latitude_country',
       'longitude_country'],
      dtype='object')
```

Handling duplicated values in our dataset

1. Checking whether there are duplicated values in the dataset
2. Handling duplicates

In [16]:

```
#check for duplicates
duplicates = billionaire_df.duplicated()
billionaire_df[duplicates]
```

Out[16]:

	finalWorth	personName	age	country	source	industries	countryOfCitizenship	selfMade	status	gender	lastName	da

rank

Exploratory Data Analysis

This method is used to visualize the different relationships between data points.

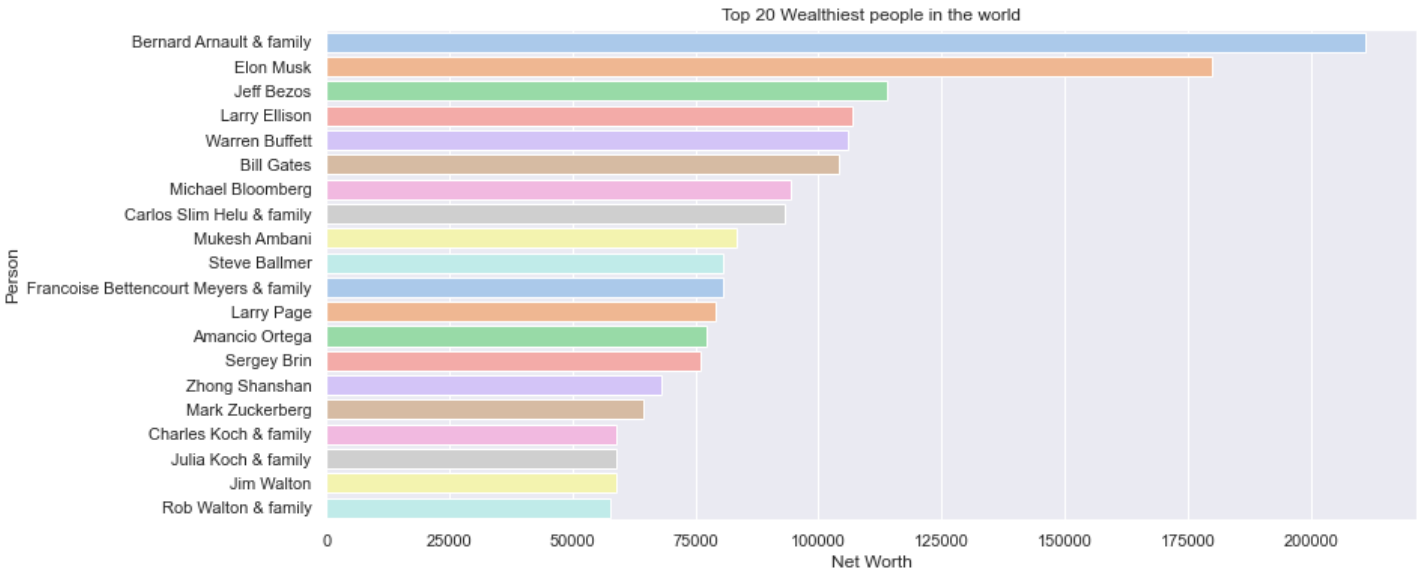
1. Wealthiest people in the world

In [17]:

```
# display the first 20 wealthiest people

billionaire_20 = billionaire_df.head(20)
sns.set(rc={"figure.figsize":(13,6)})
sns.barplot(x=billionaire_20['finalWorth'],
            y=billionaire_20['personName'],
            palette='pastel').set(title='Top 20 Wealthiest people in the world',
                                  xlabel="Net Worth",
                                  ylabel="Person")
```

Out[17]:

```
[Text(0.5, 1.0, 'Top 20 Wealthiest people in the world'),
 Text(0.5, 0, 'Net Worth'),
 Text(0, 0.5, 'Person')]
```



2. Age Distribution for Billionaires

This section displays the billionaires distribution in the various age and countries represented

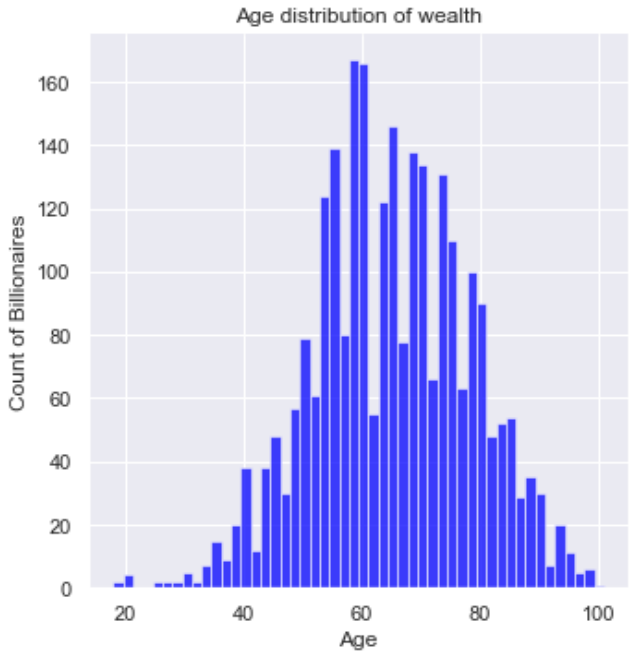
In [18]:

```
# display age distribution
```

```
# Plot the first graph in the first subplot
sns.displot(billionaire_df['age'],
            bins=50, color='blue').set(title='Age distribution of wealth',
                                       xlabel="Age",
                                       ylabel="Count of Billionaires")
```

Out[18]:

```
<seaborn.axisgrid.FacetGrid at 0x1dd1effe9d0>
```



In [19]:

```
# display industire count of wealth
country_df = billionaire_df[['countryOfCitizenship']].groupby('countryOfCitizenship').si
ze().reset_index(name='counts')
country_df.sort_values('counts', inplace=True, ascending=True)

country_df
```

Out[19]:

	countryOfCitizenship	counts
0	Algeria	1
74	Venezuela	1
67	Tanzania	1
63	St. Kitts and Nevis	1
54	Portugal	1
...	...	...
57	Russia	104
24	Germany	126
30	India	169
13	China	491
73	United States	735

77 rows × 2 columns

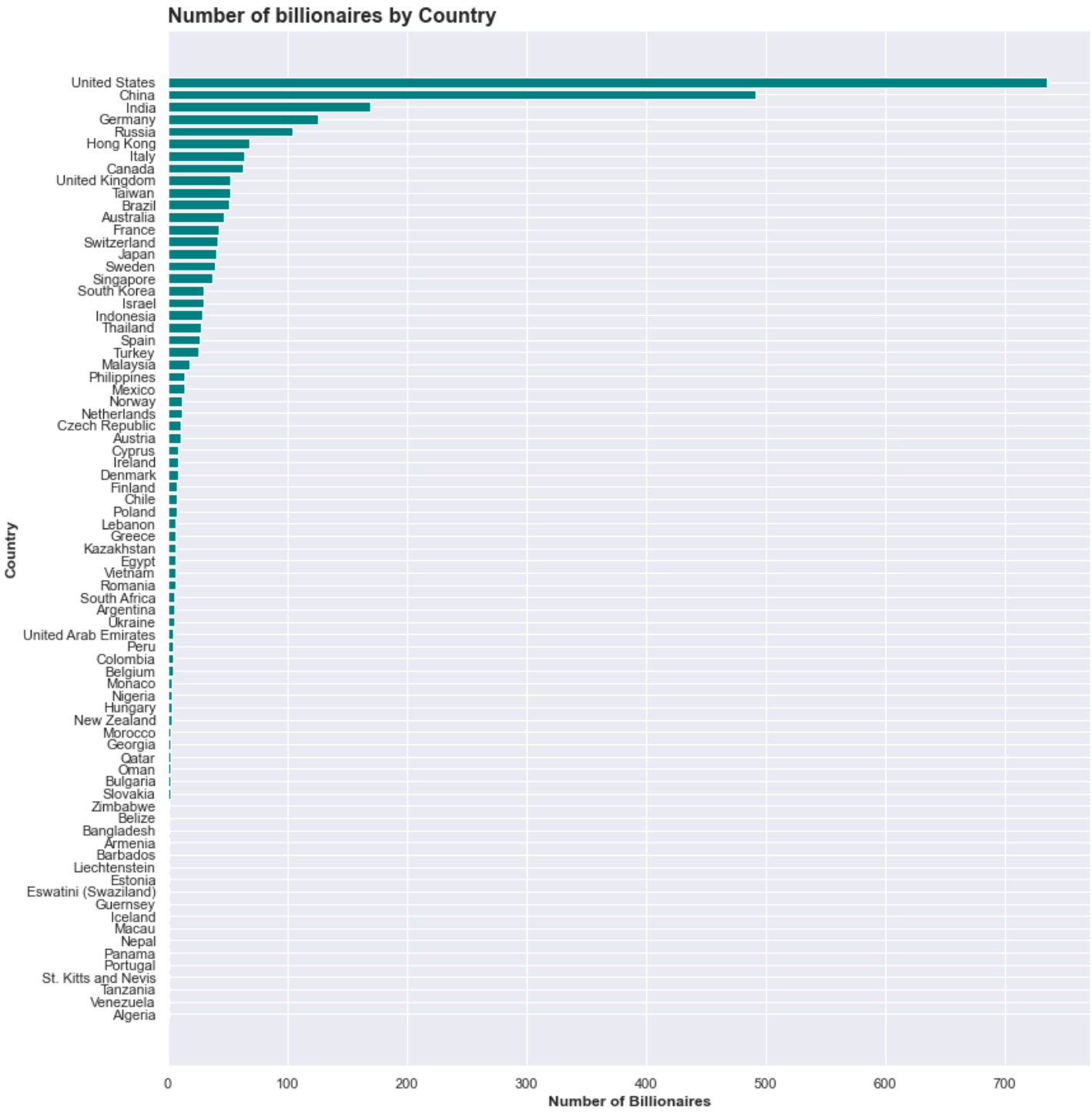
In [20]:

```
# display country vs billionaire

fig, ax = plt.subplots(figsize = (13,15))
```

```
ax.barh(country_df['countryOfCitizenship'], country_df['counts'], color='teal')
ax.set_title("Number of billionaires by Country", fontweight = "bold", fontsize=16, loc="
left")
plt.xlabel("Number of Billionaires", fontweight = "bold", fontsize=12)
plt.ylabel("Country", fontweight = "bold", fontsize=12)
```

Out[20]:

```
Text(0, 0.5, 'Country')
```



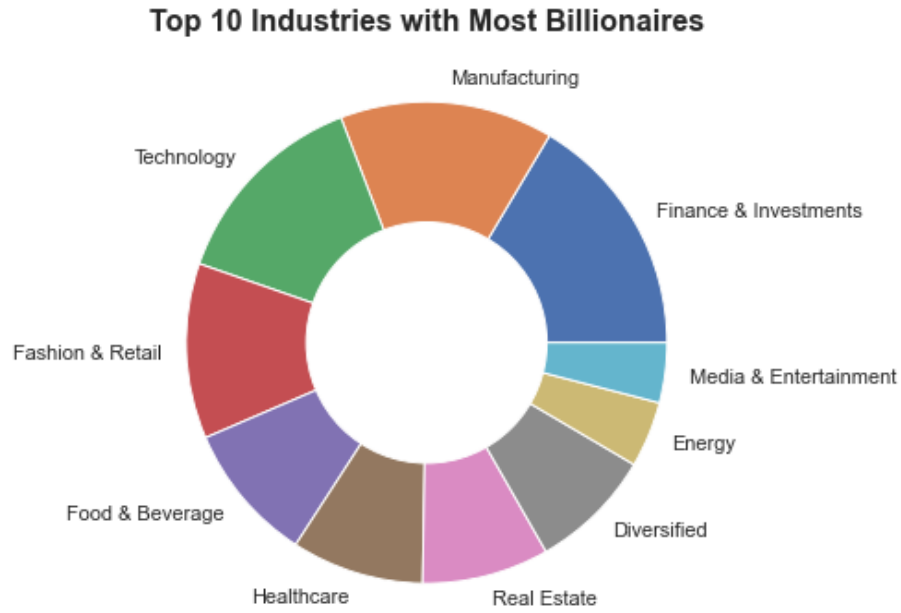
3. Wealth Distribution Analysis

This should show the distribution of wealth in the various industires and sources

In [21]:

```
# display the top 10 industries with most number of billioonaires

industry = billionaire_df['industries'].value_counts().head(10)

index = industry.index
source = industry.values
```

```
plt.figure(figsize=(13,6))
plt.pie(source, labels=index)

central_circle = plt.Circle((0,0), 0.5, color="white")
fig = plt.gcf()
fig.gca().add_artist(central_circle)
plt.rc('font', size = 12)

plt.title("Top 10 Industries with Most Billionaires", fontweight="bold", fontsize=16)

plt.show()
```



In [22]:

```
# display industire count of wealth
industry_df = billionaire_df[['industries']].groupby('industries').size().reset_index(name='counts')
industry_df.sort_values('counts', inplace=True, ascending=True)

industry_df
```

Out[22]:

	industries	counts
7	Gambling & Casinos	25
17	Telecom	31
15	Sports	39
9	Logistics	40
1	Construction & Engineering	45
14	Service	53
0	Automotive	73
12	Metals & Mining	74
11	Media & Entertainment	91
3	Energy	100
2	Diversified	187
13	Real Estate	193
8	Healthcare	201
6	Food & Beverage	212
4	Fashion & Retail	266

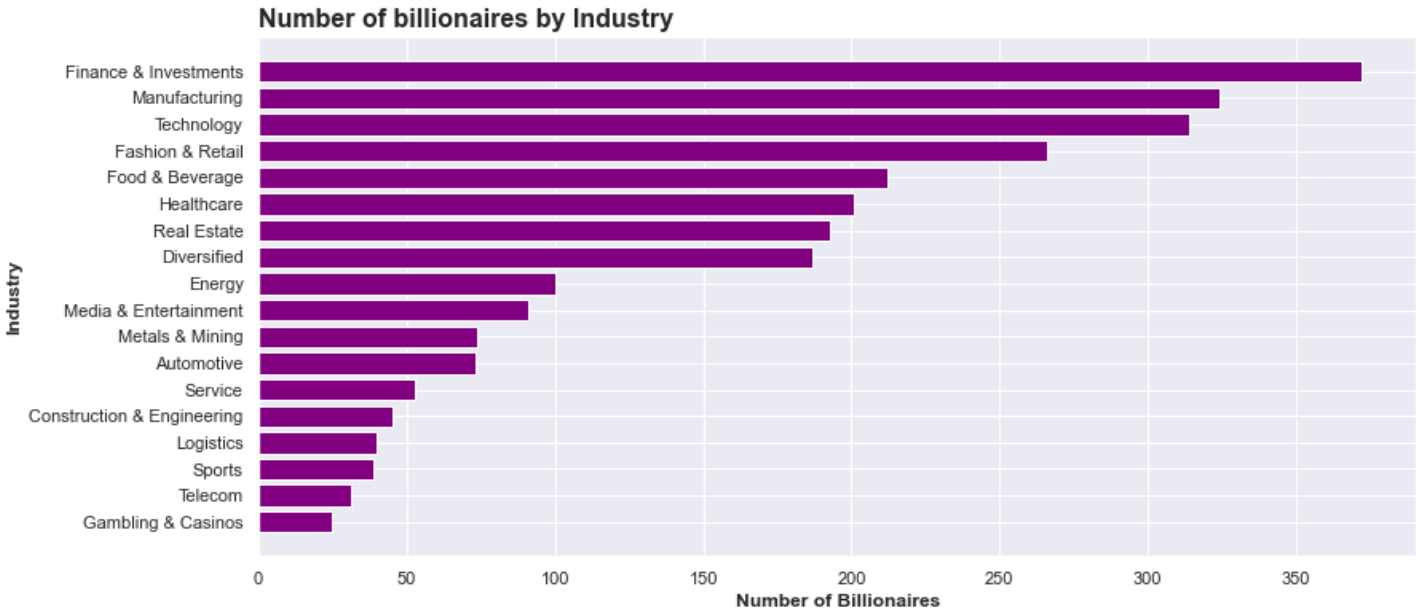
16	Technology	314
10	Manufacturing	324
5	Finance & Investments	372

In [23]:

```
# display number billionaire vs industry

fig, ax = plt.subplots(figsize = (13,6))
ax.barh(industry_df['industries'], industry_df['counts'], color='purple')
ax.set_title("Number of billionaires by Industry", fontweight = "bold", fontsize=16, loc=
"left")
plt.xlabel("Number of Billionaires", fontweight = "bold", fontsize=12)
plt.ylabel("Industry", fontweight = "bold", fontsize=12)
```

Out[23]:

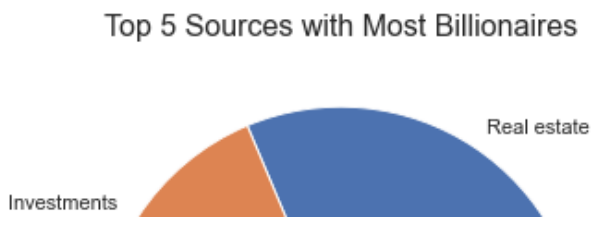
```
Text(0, 0.5, 'Industry')
```



Most profitable Sources of wealth

In [24]:

```
# display the top 10 industries with most number of billioonaires

sources = billionaire_df['source'].value_counts().head()

index = sources.index
source = sources.values
# print(source)    [151  92  91  85  63]
# print(index)
plt.figure(figsize=(13,6))
plt.pie(source, labels=index)

central_circle = plt.Circle((0,0), 0.5, color="white")
fig = plt.gcf()
fig.gca().add_artist(central_circle)

plt.title("Top 5 Sources with Most Billionaires", fontsize=16)

plt.show()
```



4. Self-made vs. inherited wealth

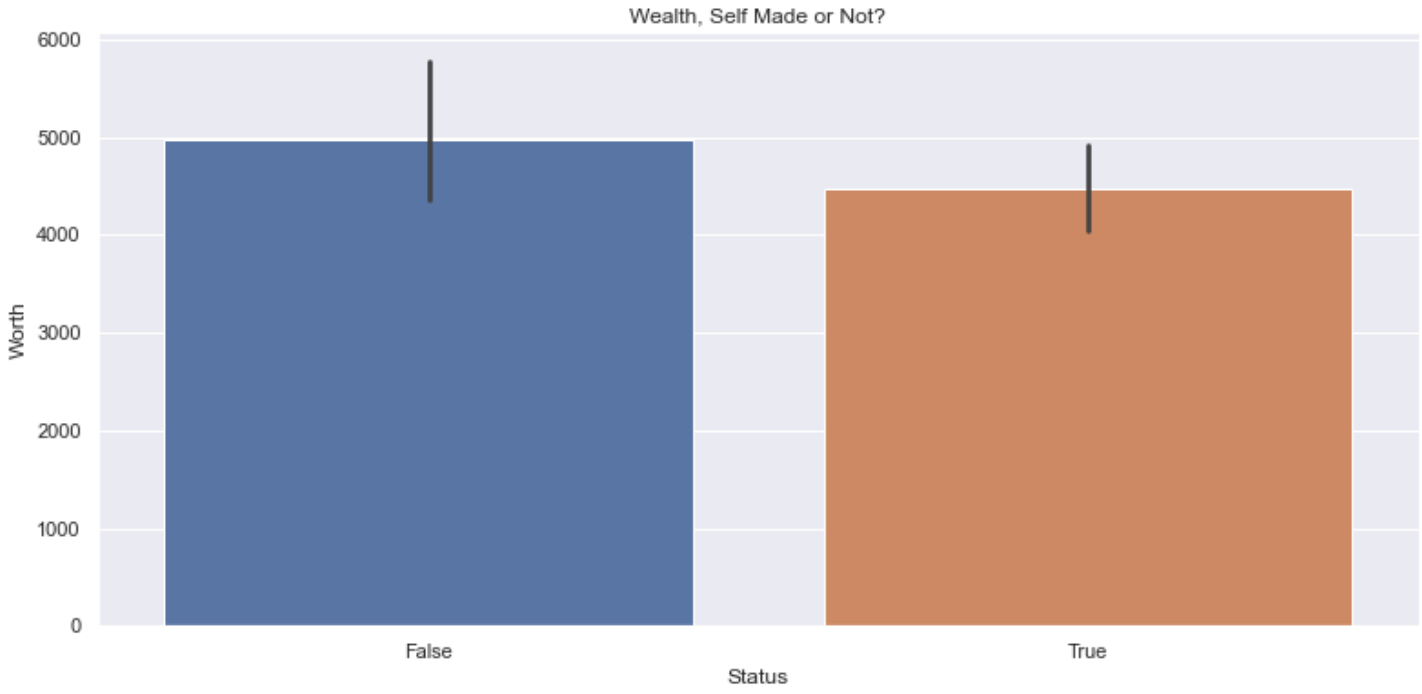
This should show the proportopn or distribution of the self-made billionaires and those who inherited wealth

In [25]:

```
# self made billionaires vs those who inherited
plt.subplots(figsize=(13,6))
sns.barplot(x='selfMade', y='finalWorth', data=billionaire_df).set(xlabel="Status", ylab
el="Worth", title="Wealth, Self Made or Not?" )
```

Out[25]:

```
[Text(0.5, 0, 'Status'),
 Text(0, 0.5, 'Worth'),
 Text(0.5, 1.0, 'Wealth, Self Made or Not?')]
```

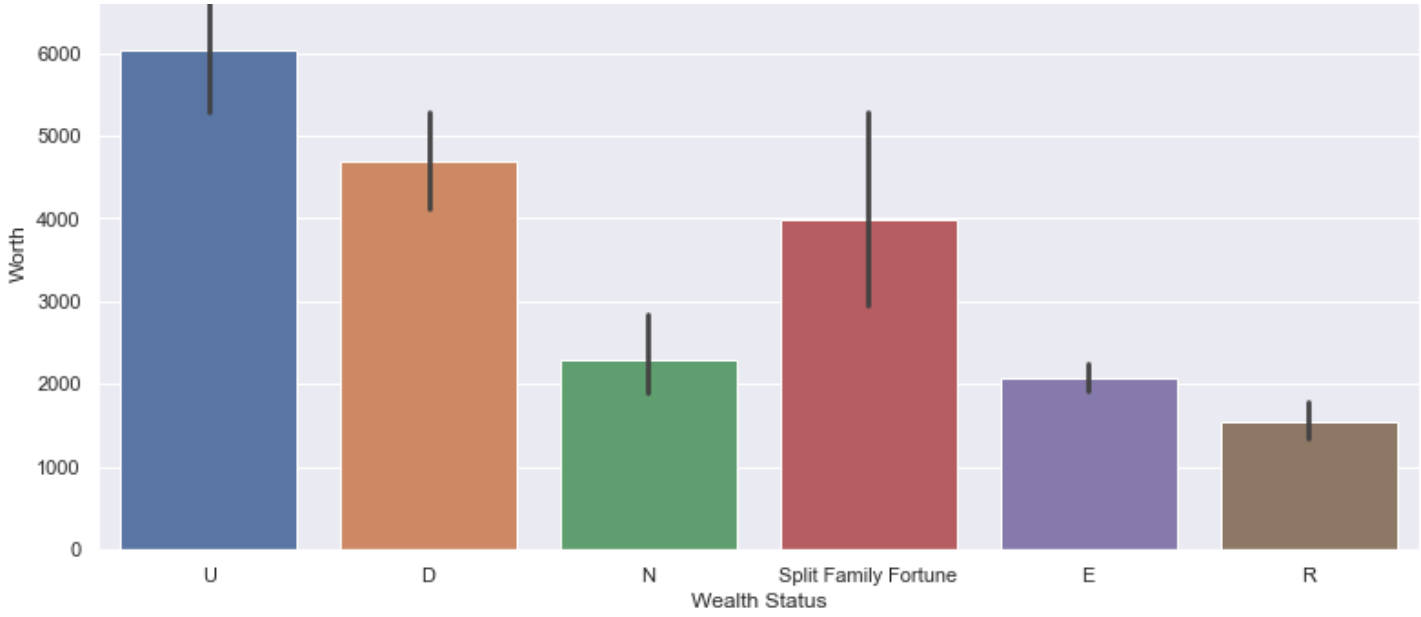


In [26]:

```
# self made billionaires vs those who inherited
plt.subplots(figsize=(13,6))
sns.barplot(x='status', y='finalWorth', data=billionaire_df).set(xlabel="Wealth Status",
ylabel="Worth", title="Wealth Status Distribution" )
```

Out[26]:

```
[Text(0.5, 0, 'Wealth Status'),
 Text(0, 0.5, 'Worth'),
 Text(0.5, 1.0, 'Wealth Status Distribution')]
```



Determine what model to use

In [27]:

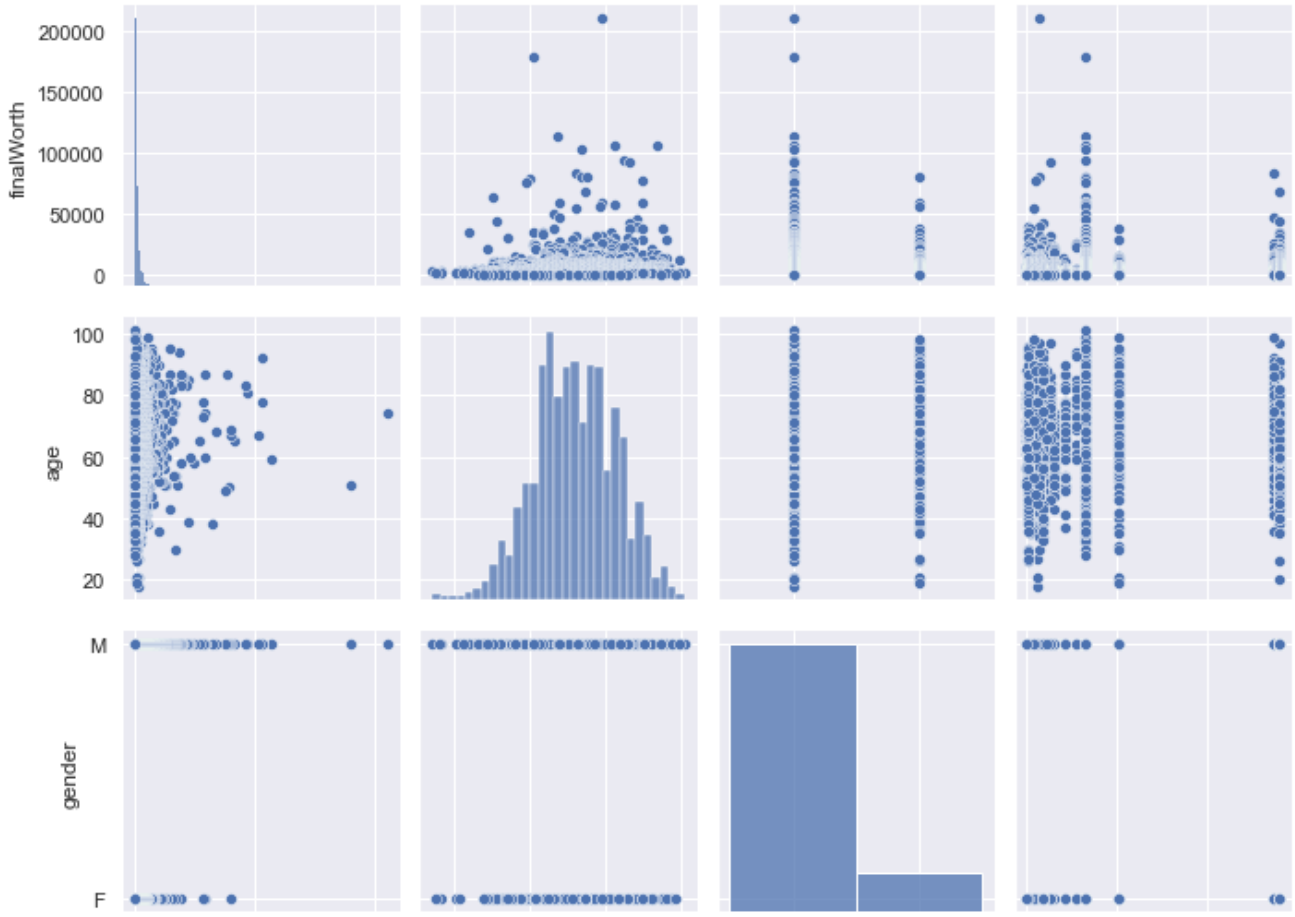
```
# define a pairplot visualization function

# sns.pairplot(billionaire_df)
```

In [28]:

```
# demographic analysisi for wealth distribution
demo_df = billionaire_df[['finalWorth','age', 'gender', 'population_country']]

sns.pairplot(demo_df, vars=demo_df)
# demo_df.columns
```

Out[28]:

```
<seaborn.axisgrid.PairGrid at 0x1dd1fb191c0>
```



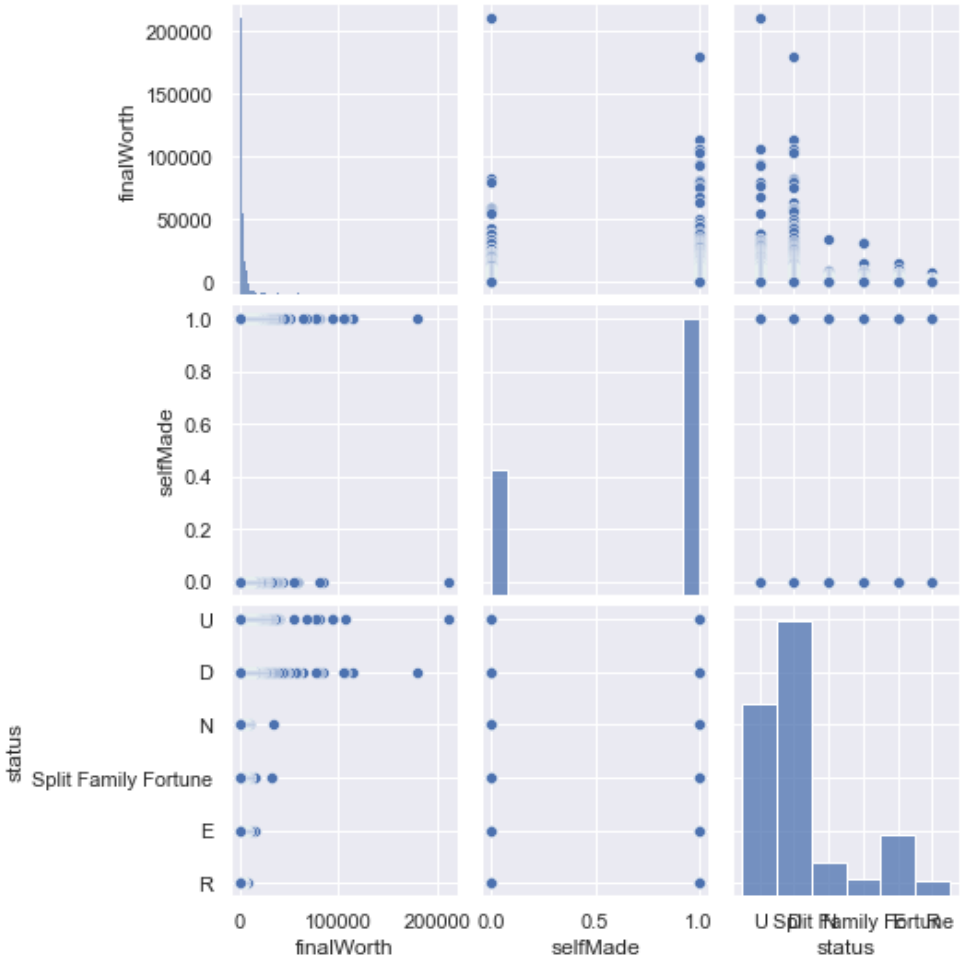
In [29]:

```
# self_made, status
status_df = billionaire_df[['finalWorth', 'selfMade', 'status']]

sns.pairplot(status_df, vars= status_df)
# status_df
```

```
<__array_function__ internals>:180: RuntimeWarning: Converting input from bool to <class
'numpy.uint8'> for compatibility.
```

Out[29]:

```
<seaborn.axisgrid.PairGrid at 0x1dd21ece340>
```



In [30]:

```
fig, ax = plt.subplots(figsize=(13,6))
sns.heatmap(billionaire_df.corr(), center=0, cmap='Blues', linecolor='white', linewidths
=.1, cbar=True)
ax.set_title('Multi-Collinearity of Billionaire Features', fontweight="bold", fontsize=1
6)
```

Out[30]:

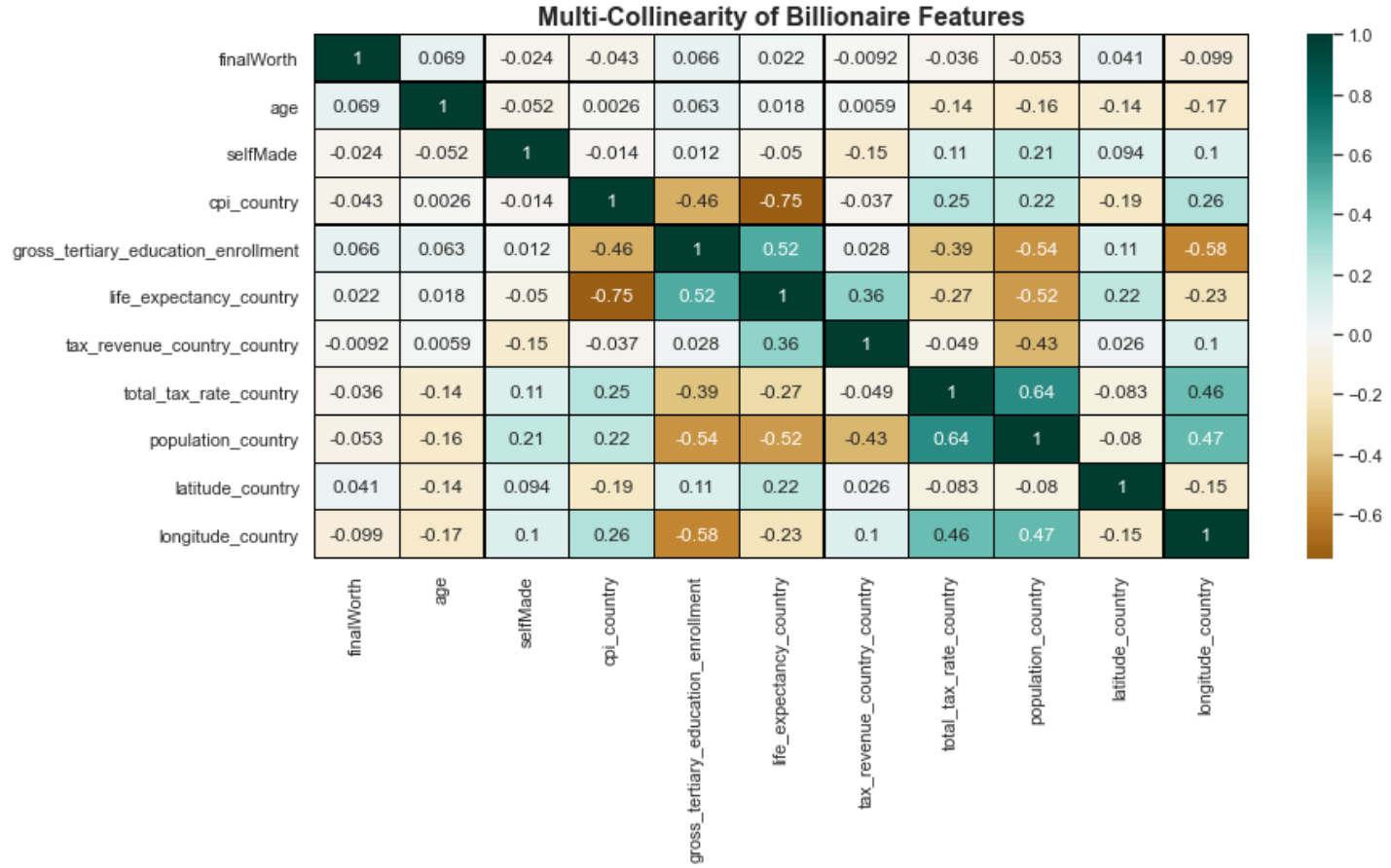
```
Text(0.5, 1.0, 'Multi-Collinearity of Billionaire Features')
```

In [31]:

```
fig, ax = plt.subplots(figsize=(13,6))
sns.heatmap(billionaire_df.corr(), center=0, cmap='BrBG', annot=True, linecolor='black',
linewidths=.1)
ax.set_title('Multi-Collinearity of Billionaire Features', fontweight="bold", fontsize=1
6)
```

Out[31]:

```
Text(0.5, 1.0, 'Multi-Collinearity of Billionaire Features')
```

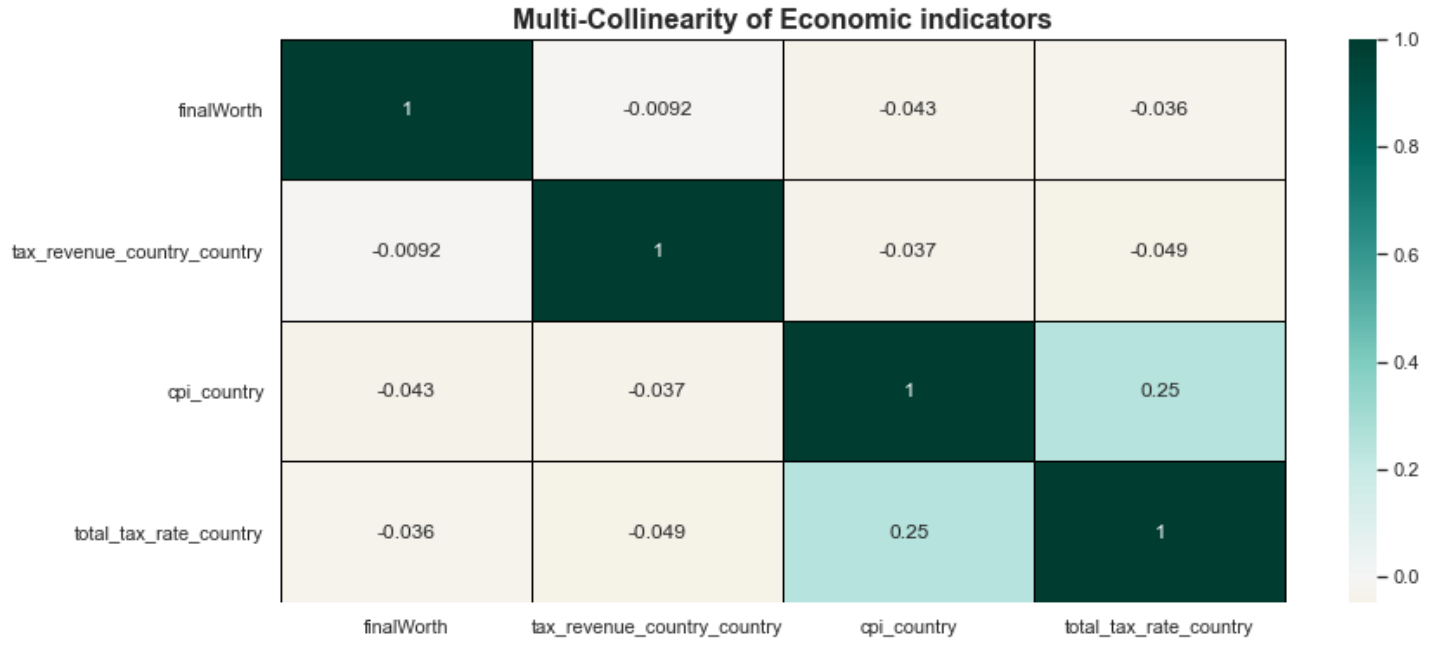


In [32]:

```
# correlations between wealth and ecocnomic indicators (CPI,TAX, GDP)

economic_df = billionaire_df[['finalWorth', 'tax_revenue_country_country','cpi_country',
'total_tax_rate_country']]
# economic_df.info()

# check for collinearity between the variables
fig, ax = plt.subplots(figsize=(13,6))
sns.heatmap(economic_df.corr(), center=0, cmap='BrBG', annot=True, linecolor='black', li
newidths=.1)
ax.set_title('Multi-Collinearity of Economic indicators', fontweight="bold", fontsize=16
)
```

Out[32]:

```
Text(0.5, 1.0, 'Multi-Collinearity of Economic indicators')
```



Modelling

a. Wealth Classification

We can classify billionaires into different wealth categories such as "Low Net Worth," "Average Net Worth," and "High Net Worth."

1. **Perform a Train-Test Split**

This step is used to seperate the dataframe set into two sets (the X and y variables)

X variable contains all columns in the dataframe except the target

y variable contains the target variable

In [33]:

```
# 1. Data Splitting

# Define wealth status categories based on finalWorth
# # Define the bin edges
bins = 3
labels = ["Low Net Worth", "Mid Net Worth", "High Net Worth"]
data = billionaire_df['finalWorth']
billionaire_df['WealthStatus'] = pd.cut(data, bins, labels=labels, right=False, include_
lowest=False, precision=3, duplicates='raise')

# print(billionaire_df['WealthStatus'])
```

In [34]:

```
# examine the distribution of the target var
print(billionaire_df['WealthStatus'].value_counts())
```

```
Low Net Worth     2626
Mid Net Worth       12
High Net Worth       2
Name: WealthStatus, dtype: int64
```

Bearing in mind that the training set is used to train the model whiel the test set is used to evaluate its performance, we allocate 40% of the data as the test-size as below.

In [35]:

```
# assign variables

X = billionaire_df.drop(columns=['personName', 'lastName','WealthStatus'], axis=1)
y = billionaire_df['WealthStatus']
```

In [36]:

```
# billionaire_df.columns
# print(X, y)
```

In [37]:

```
from sklearn.model_selection import train_test_split

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42
)
```

1. **Preprocess Data**

For our given dataset, we are required to encode the different columns we are dealing with from categorical values to numeric values In this case, we will implement:

i. One-Hot encoding technique to handle the different variable features

ii. LabelEncoder technique to handle the target variable

Encoding Categorical Variables

In [38]:

```
# convert the categorial variable to numeric values

from sklearn.preprocessing import OneHotEncoder
# one-hot encoder
features_encoder = OneHotEncoder(handle_unknown='ignore')

X_train_encoded = features_encoder.fit_transform(X)
X_test_encoded = features_encoder.fit_transform(X)
```

In [39]:

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y_train_encoded = encoder.fit_transform(y)
y_test_encoded = encoder.fit_transform(y)
```

1. **Model Fitting**

In [ ]:

```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression(max_iter=5000, multi_class='multinomial', solver='lbfgs')

#Train the model
model.fit(X_train_encoded, y_train_encoded)
```

In [ ]:

```
# check for consistency
print(X_train_encoded.shape)
print(y_train_encoded.shape)
```

1. **Model Evaluation**

In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, conf
usion_matrix

# Make predictions
y_pred = model.predict(X_test_encoded)

# Evaluate the model
accuracy = accuracy_score(y_test_encoded, y_pred)
precision = precision_score(y_test_encoded, y_pred, average='weighted')
recall = recall_score(y_test_encoded, y_pred, average='weighted')
f1 = f1_score(y_test_encoded, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test_encoded, y_pred)
print({"Accuracy": accuracy,
       "Precision": precision,
       "Recall": recall,
       "f1_score": f1,
       "confusion_matrix": conf_matrix})
```

Non parametric and Ensembling Modelling

This section is used to prepare non parametric modelling solutions that will be used to compare with the logistic regression earlier modelled

In [ ]:

```
# 1. Create a decision tree classifier

from sklearn.tree import DecisionTreeClassifier

# Initialized DecisionTree
dt_classifier  = DecisionTreeClassifier(max_depth=3, min_samples_split=100, random_state
= 42)

# 2. Train a decision tree classifier
dt_classifier.fit(X_train_encoded, y_train_encoded)

# 3. Make predictions
yd_pred = dt_classifier.predict(X_test_encoded)
```

1. **Evaluate Model**

In [ ]:

```
from sklearn.metrics import accuracy_score
dt_accuracy = accuracy_score(y_test_encoded, yd_pred)

print({'Accuracy': dt_accuracy})
```

Model Intepreation

For the Wealth Classification as stipulated, we use Multinomial Logistic regression since we have the 3 types of

For the Wealth Classification as stipulated, we use Multinomial Logistic regression since we have the 3 types of classification we are handling for the target variable.

`Summary`:

1. **Given that the `accuracy_score = 0.995` accuracy is high, it indicates that the model is good at predicting the Wealth Status.**
2. **The Precision is the ability of the classifier not to label as positive a sample that is negative. Given that our `precision_score = 0.994` precision is high, it means that the false positive rate is low.**
3. **Recall is the ability of the classifier to find all the positive samples. Given that our `recall_score = 0.995` recall is high, it means that the false negative rate is low.**
4. **The F1 Score `= 0.9929` is the harmonic mean of precision and recall. It is a good measure to use if you need to seek a balance between precision and recall.**
5. **The confusion matrix provides a summary of the performance of the classifier. From the given matrix array, the results show the number of true positive, true negative, false positive, and false negative predictions for each class.**

In [ ]:

```
```

b. Industry Analysis

1. **Perform a Train-Test Split**

This step is used to seperate the dataframe set into two sets (the X and y variables)

`X variable`: **contains all columns in the dataframe except the target**

`y variable`: **contains the target variable(industry)**

In [ ]:

```
# Assign variables
feat = billionaire_df.drop(columns=['personName', 'lastName','selfMade', 'status', 'date
','industries'], axis=1)
ind = billionaire_df['industries']
```

In [ ]:

```
# print(feat)
```

In [ ]:

```
# Data Splitting
from sklearn.model_selection import train_test_split

# Split the data
F_train, F_test, i_train, i_test = train_test_split(feat, ind, test_size=0.4, random_sta
te=42)
```

1. **Preprocess Data**

For our given dataset, we are required to encode the different columns we are dealing with from categorical values to numeric values In this case, we will implement:

`i. One-Hot encoding technique` **to handle the different variable features**

`ii. LabelEncoder technique` **to handle the target variable**

Encoding Categorical Variables

In [ ]:

```
# convert the categorial variable to numeric values
```

```
# convert the categorial variable to numeric values

from sklearn.preprocessing import OneHotEncoder
# one-hot encoder
features_encoder = OneHotEncoder(handle_unknown='ignore')

F_train_encoded = features_encoder.fit_transform(feat)
F_test_encoded = features_encoder.fit_transform(feat)
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
i_train_encoded = encoder.fit_transform(ind)
i_test_encoded = encoder.fit_transform(ind)
```

1. **Model Training**

   Train a logistic regression model on the training data.

In [ ]:

```
from sklearn.linear_model import LogisticRegression

# Initialize the logistic regression model
model = LogisticRegression(max_iter=5000, multi_class='multinomial', solver='lbfgs')

# Train the model
model.fit(F_train_encoded, i_train_encoded)
```

1. **Model Evaluation**

Evaluate the performance of the model using the test data and suitable metrics (e.g., accuracy, precision, recall).

In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, conf
usion_matrix

# Make predictions
y_pred = model.predict(F_test_encoded)

# Evaluate the model
accuracy = accuracy_score(i_test_encoded, y_pred)
precision = precision_score(i_test_encoded, y_pred, average='weighted')
recall = recall_score(i_test_encoded, y_pred, average='weighted')
f1 = f1_score(i_test_encoded, y_pred, average='weighted')
con_matrix = confusion_matrix(i_test_encoded, y_pred)

print({"Accuracy": accuracy,"Precision": precision,"Recall": recall,"f1_score": f1, "con
fusion_matrix": con_matrix})
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# Instantiate the model
model = LogisticRegression(random_state=42)

# Fit the model on the scaled data
model.fit(F_train_encoded, i_train_encoded)


yd_train_pred = model.predict(F_train_encoded)

# Display the confusion matrix
```

```
conf_mat = confusion_matrix( i_train_encoded, yd_train_pred)
sns.heatmap(conf_mat, annot=True, fmt='g', cmap='Blues', )
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

In [ ]:

```
# Get the coefficients
coefficients = model.coef_
# print(feat)
print(coefficients)
```

In [ ]:

```
# Get the intercept
intercept = model.intercept_

# Print the intercept
print(intercept)

# the intercept represents the log-odds of the target variable being 1 when all the featu
res are 0
```

5. Model Intepretation

1. **The sign of the coefficient indicates the direction of the relationship:**

If the `coefficient is positive`, it indicates that as the predictor variable increases, the log-odds of the target variable being 1 increases, and hence, the probability of the target variable being 1 increases.

If the `coefficient is negative`, it indicates that as the predictor variable increases, the log-odds of the target variable being 1 decreases, and hence, the probability of the target variable being 1 decreases.

1. **The magnitude of the coefficient indicates the strength of the relationship:**

A `larger magnitude (further away from 0)` indicates a stronger relationship between the predictor variable and the target variable.

A `smaller magnitude (closer to 0)` indicates a weaker relationship between the predictor variable and the target variable.

Findings and Recommendations

Findings

1. **From The age distribution analysis, Most Billionaires in the world lie in the age group bracket of 50 -60 years**
2. **The top five countries of citizenship for most billionaires include : United States, China, India, Germany, Russia**
3. **The top industry with most number of billionaires in the world are: Finance and Investments, Manufacturing, Technology, Fashion & Retail, Food & Beverage**
4. **The major sources of wealth include: Investments, Real Estate, Software, Pharmaceuticals**
5. **On the question on whether Most billionaire acquired their wealth themselves or Inherited, we have seen that most of them have inherited**

Recommendations

To the Research Agency, I would recommend investing in the Finance & Investments, Technolgy industries, as it shows potential of growing, not only for the younger generation but also the aging. Also,

Conclusion

From the project above, we are able to see that the industry with a high number of wealth distributed is the

Finance & Investments industry. Hence we are able to conclude that, based on the wealth age distribution, wealth is accumulated over the years, having best invested in the right industry.

The various logistic modelling, present an almost perfect accuracy score based on the variables indicated. The technologies used for this project include: Data Preparation, Exploratory Data Analysis methods, Data Modelling