Contents

Data Analysis (python)

I've recently completed a data analysis project using Python, focusing on the Forbes billionaire dataset. This analysis delved into the wealth and attributes of the world's billionaires. Leveraging Python's powerful data manipulation libraries like pandas, I cleaned the dataset by addressing missing values and refining its structure. With the help of visualization tools such as matplotlib and seaborn, I uncovered intriguing insights about the distribution of wealth, industries with the highest representation, and correlations between factors like age, source of wealth, and net worth. The project not only provided a comprehensive snapshot of the billionaire landscape but also showcased the capabilities of Python in extracting valuable insights from complex datasets.

> Data Source

Data Description

```
df.describe()
```

The code df.describe() is a concise way to generate a summary of statistical information from a dataset stored in a DataFrame named df. When executed in Python using libraries like pandas, this command provides key statistics for each numerical column in the DataFrame. The output includes the count of non-null values, the mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum value. This summary offers a quick overview of the distribution and central tendencies of the dataset's numeric data, aiding in the initial exploration and understanding of the dataset's characteristics.

column drop

```

```

```
drop_colums =
['organization_name','position_in_organization','wealth_status','business_category
','birth_date']
df2=df.drop(columns=drop_colums)
```

df.drop(columns=drop_colums), a new DataFrame df2 is created, excluding the specified columns. This step simplifies the dataset by removing unnecessary features, focusing the analysis on more relevant data. The dropped columns, such as organization details, wealth status, and birth dates, were not critical for the specific analysis being conducted.

Dealing with null values

The code df2.isnull().sum() is used to compute the sum of missing values in each column of the DataFrame df2. By applying the isnull() function, it identifies which cells have missing (null) values, and then sum() tallies these occurrences for each column. This analysis helps understand the extent of missing data in the dataset.

```
df2.isnull().sum()
```

Null values can impact insights by causing inaccuracies in statistical measures and visualizations. They might lead to biased conclusions and hinder the reliability of patterns identified during data analysis. It's important to handle missing values appropriately, either by imputing them with estimated values or excluding them when their impact on the analysis is minimal.

Series	Total Null values
year	0
month	0
rank	18
net_worth	0
last_name	3689
first_name	3747
full_name	0
age	675
gender	3829
country_of_citizenship	9
country_of_residence	702
city_of_residence	935
business_industries	990

Series	Total Null values
self_made	3689

This inoformation tell about the total number of null values in a DataFrame. These values are too much. Removing these NaN is the first priority of the process

```
df3=df2.dropna()
```

After dropping Null values. The 16.8% data has been dropped which caused to reducing Rows from 31732 rows to 26401.

> Original Data Shape :(31732 rows × 14 columns)

> current Data shape: (26401 rows × 14 columns)

Data type Check

```
df3.dtypes
```

df3.dtypes is used to display the data types of each column in the DataFrame df3. This command helps us understand how the data is stored in the DataFrame – whether it's numeric, textual, or categorical. The detail is as follows:

Series	Data Type
year	int64
month	int64
rank	float64
net_worth	object
last_name	object
first_name	object
full_name	object
age	float64
gender	object
country_of_citizenship	object
country_of_residence	object
city_of_residence	object
business_industries	object

Series	Data Type
self_made	object

Change columns name

```
df3.rename(columns={'net_worth':'net_worth_Billions'},inplace = True)
```

The purpose of this code is to change the title of the series(columns) in DataFrame. Some times it becomes important to give different title to columns.

looking into insights

The most common business industries for billionaires

```
industry_counts = df3['business_industries'].value_counts().head(4)
print(industry_counts)
```

This code segment is intended to analyze and present the distribution of business industries among billionaires in the DataFrame df3. By using value_counts(), it calculates the frequency of each unique value in the 'business_industries' column. The .head(4) limits the output to the top four industries with the highest occurrences. The resulting industry_counts variable holds this summarized information. Finally, the print(industry_counts) statement displays these top industry counts, offering insights into the most common sectors that billionaires are involved in.

Industries	Total Count
Technology	2570
Manufacturing	2397
Fashion & Retail	2325
Real Estate	2188

self made vs inhereited billionaires

```
self_made_billionaires = df3[df3['self_made'] == 1]
inherited_billionaires = df3[df3['self_made'] == 0]

print(f'Number of self-made billionaires: {len(self_made_billionaires)}')
print(f'Number of inherited billionaires: {len(inherited_billionaires)}')

if len(self_made_billionaires) > len(inherited_billionaires):
  print('There are more self-made billionaires in the data set.')
```

```
else:
  print('There are more inherited billionaires in the data set.')
```

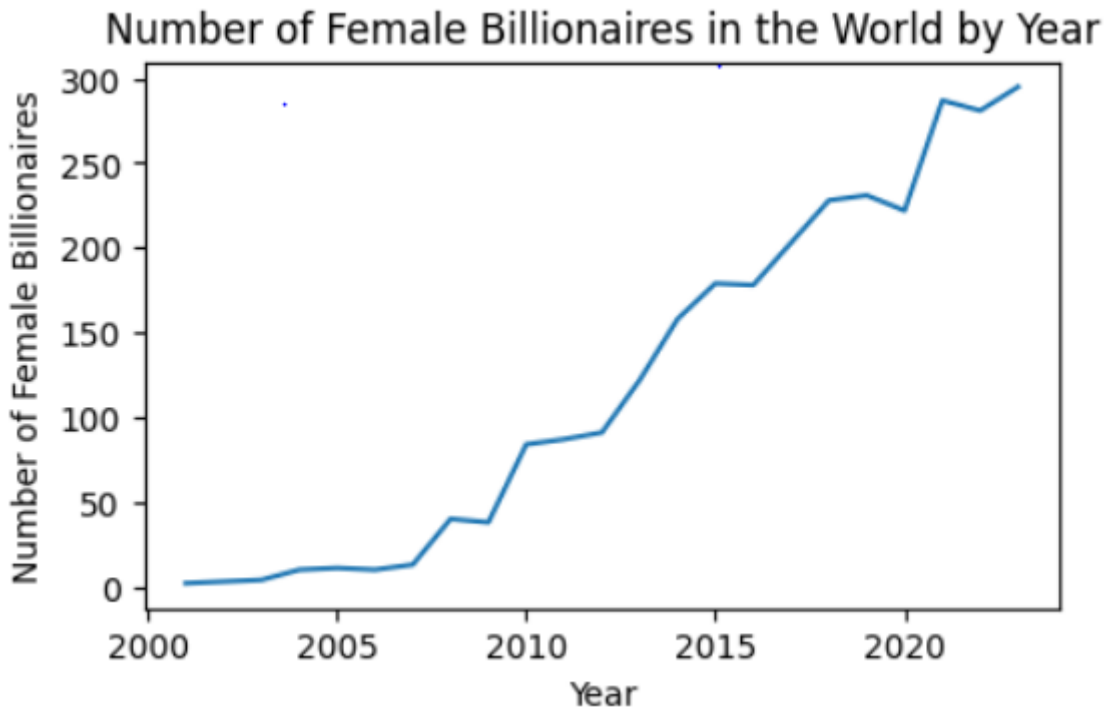
This code partitions the DataFrame df3 into two groups: self_made_billionaires and inherited_billionaires. The division is based on the 'self_made' column, where a value of 1 represents self-made billionaires and 0 represents those who inherited their wealth. The subsequent lines calculate and display the counts for both groups. By comparing the counts, the code determines whether there are more self-made or inherited billionaires and prints the corresponding result. This analysis sheds light on the prevalence of self-made versus inherited wealth in the dataset. The result is as follows:

Status	Total Count
Number of self-made billionaires	16927
Number of inherited billionaires	9474

There are more self-made billionaires in the data set.

Number of female billionaires

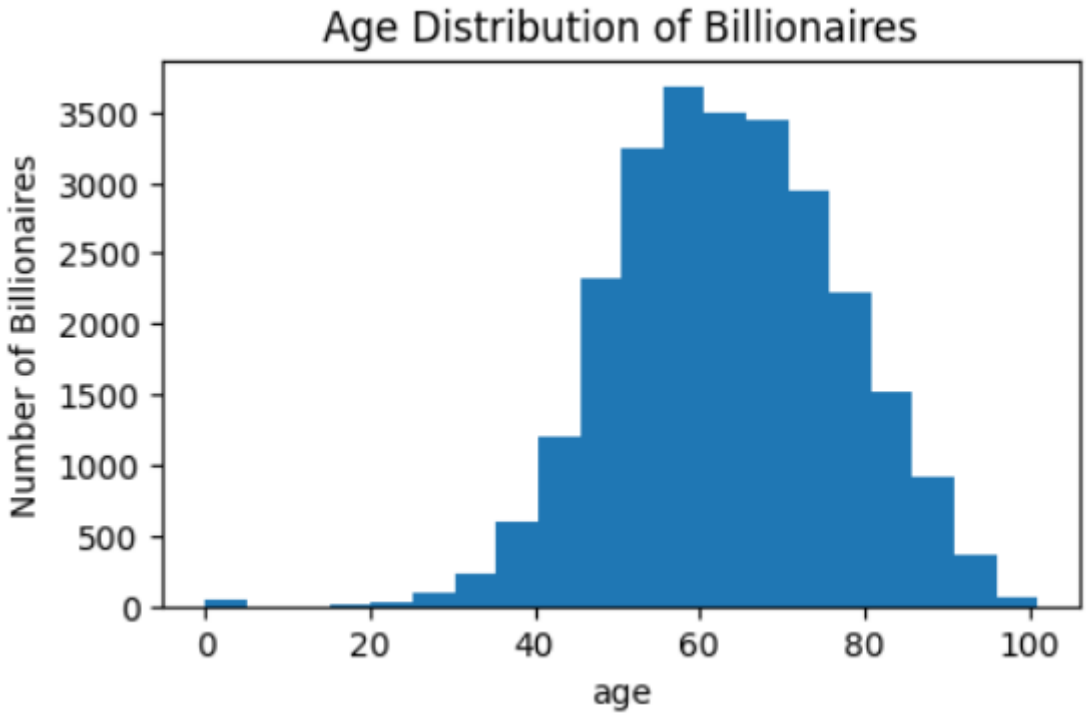
```
years = df3['year'].unique()

female_billionaires_per_year = {}
for year in years:
    female_billionaires_per_year[year] = len(df3[df3['year'] == year]
[df3['gender'] == 'Female'])

plt.figure(figsize=(5, 3))
plt.plot(years, female_billionaires_per_year.values())
plt.xlabel('Year')
plt.ylabel('Number of Female Billionaires')
plt.title('Number of Female Billionaires in the World by Year')
plt.show()
```



The age distribution of billionaires

```
age_distribution = df3['age']
number_of_bins = 20
plt.figure(figsize=(5, 3))
plt.hist(age_distribution, bins=number_of_bins)
plt.xlabel('age')
plt.ylabel('Number of Billionaires')
plt.title('Age Distribution of Billionaires')
plt.show()
```

is code segment visualizes the age distribution of billionaires using a histogram. It assumes that df3 contains the relevant age data. The age_distribution variable holds the age values. The histogram is constructed with 20 bins using plt.hist(), displaying the frequency of age occurrences. The plot is labeled with appropriate axis titles and a title, and then displayed using plt.show(). This visualization offers insights into the age distribution of billionaires in the dataset.

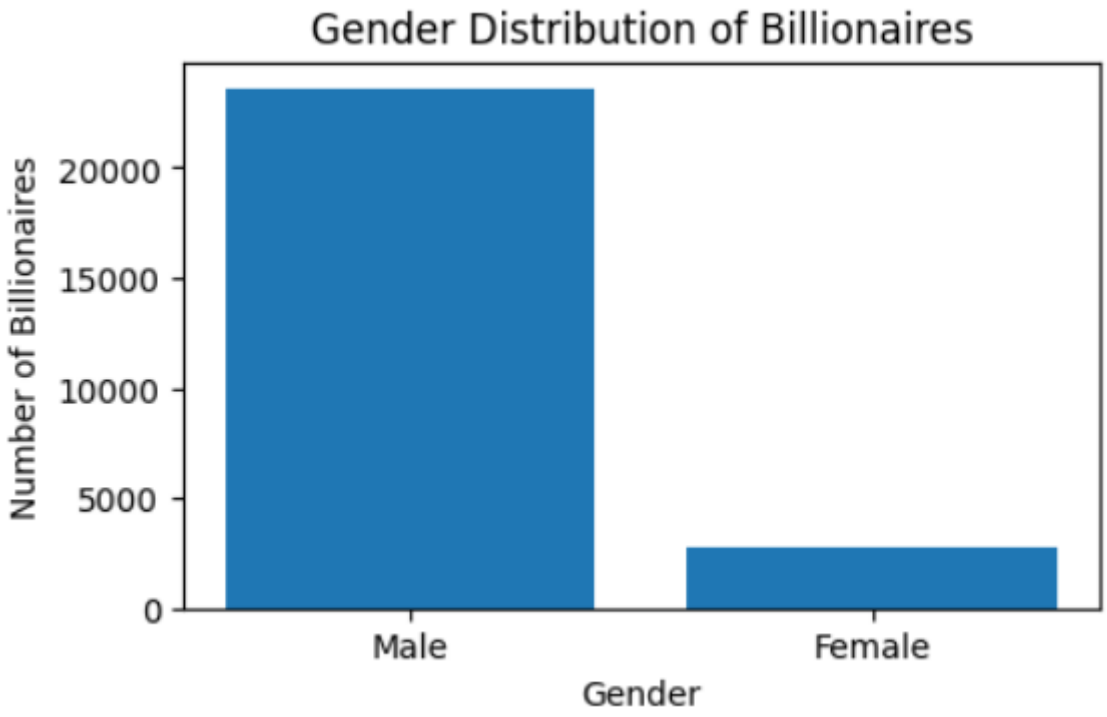


The gender distribution of billionaires

he gender distribution of billionaires is heavily skewed towards men. This is likely due to the fact that women have historically faced more challenges in business, such as lack of access to capital and discrimination

```
gender_distribution = df3['gender'].value_counts()

print('The gender distribution of billionaires is:')
print(gender_distribution)

plt.figure(figsize=(5, 3))
plt.bar(gender_distribution.index, gender_distribution.values)
plt.xlabel('Gender')
plt.ylabel('Number of Billionaires')
plt.title('Gender Distribution of Billionaires')
plt.show()
```

This code segment analyzes and visualizes the gender distribution of billionaires. The gender_distribution variable holds the count of each gender category. The first two print() statements display the gender distribution. The subsequent lines create a bar plot using plt.bar(), depicting the number of billionaires for each gender. The plot is labeled with suitable axis titles and a title, and then displayed using plt.show(). This visualization provides insights into the representation of genders among billionaires in the dataset.



The distribution of billionaires by country of citizenship

This code segment analyzes the distribution of billionaires by their countries of citizenship. The country_distribution variable stores the count of billionaires for each country. The most_common_countries variable selects the top three countries with the highest billionaire counts. The next lines print the list of most common countries and calculate the percentage of billionaires from these countries in relation to the total number of billionaires. This analysis provides insights into the concentration of billionaires in certain countries and their collective representation among all billionaires in the dataset.

```
country_distribution = df3['country_of_citizenship'].value_counts()
most_common_countries = country_distribution.index[:3]
print('The most common countries for billionaires are:', most_common_countries)
print('The percentage of billionaires from these countries is:',
print(country_distribution[most_common_countries].sum() /
country_distribution.sum() * 100))
```

> Top3 : **United States, China and Russia**
> 52.7% of billionaires are from these countries