In [1]:
```
import pandas as pd
```

Documentation:

- 1 .Reading the dataset: The code reads the Titanic dataset from a CSV file using pandas and stores it in the 'data' DataFrame.
- 1. Displaying the dataset: The first few rows of the dataset are displayed using the 'head()' method.
- 1. Calculating statistics for 'Age':
  - Median: The median of the 'Age' column is calculated.
  - Mode: The mode of the 'Age' column is calculated.
- 1. Descriptive Statistics for 'Age' and 'Fare':
  - Median: The median of both 'Age' and 'Fare' columns is calculated.
  - General statistics: Descriptive statistics for 'Age' and 'Fare' columns, including mean, std, min, 25%, 50%, 75%, and max.
- 1. Aggregating statistics:
  - Aggregated statistics for 'Age' (median, max, min, skew) and 'Fare' (median, max, min, skew, mean) are calculated.
- 1. Grouping by gender:
  - The mean age for each gender is calculated using the 'groupby' method.

In [2]:
```
data = pd.read_csv("titanic.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [4]:
```
data["Age"].median()
```

Out[4]:
```
28.0
```

In [5]:
```
data["Age"].mode()
```

Out[5]:
```
0    24.0
Name: Age, dtype: float64
```

Descriptive Statistics for Age

- For one or more columns

In [6]:
```
data[["Age", "Fare"]].median()
```

Out[6]:
```
Age     28.0000
Fare    14.4542
dtype: float64
```

Descriptive Statistics for Age and Fare

In [7]:
```
data[["Age", "Fare"]].describe()
```

Out[7]:

	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

In [8]:
```
data.agg({
    "Age":["median", "max", "min","skew"],
    "Fare":["median", "max", "min","skew", "mean"],
})
```

Out[8]:

	Age	Fare
median	28.000000	14.454200
max	80.000000	512.329200
min	0.420000	0.000000
skew	0.389108	4.787317
mean	NaN	32.204208

Groupby

In [9]:
```
# Calculating the mean age for each gender
data[["Sex", "Age"]].groupby("Sex").mean()
```

Out[9]:

Sex	Age
female	27.915709
male	30.726645

- 1. Grouping by gender: The 'data' DataFrame is grouped by the "Sex" column.
- 1. Calculating mean: The mean values for numeric columns within each gender group are calculated.

In [10]:
```
data.groupby("Sex").mean(numeric_only = True)
```

Out[10]:

Sex	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
female	431.028662	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818
male	454.147314	0.188908	2.389948	30.726645	0.429809	0.235702	25.523893

In [11]:
```
data.groupby("Sex")["Age"].mean()
```

Out[11]:
```
Sex
female    27.915709
male      30.726645
Name: Age, dtype: float64
```

In [12]:
```
#Grouping can be done by multiple columns at the same time.
data.groupby(["Sex","Pclass"])["Fare"].mean()
```

Out[12]:
```
Sex     Pclass
female  1         106.125798
        2          21.970121
        3          16.118810
male    1          67.226127
        2          19.741782
        3          12.661633
Name: Fare, dtype: float64
```

Count number of records by category

- The value_counts() method counts the number of records for each category in a column.
- groupby provides the power of the split-apply-combine pattern.

In [13]:
```
data["Pclass"].value_counts()
```

Out[13]:
```
Pclass
3    491
1    216
2    184
Name: count, dtype: int64
```

In [14]:
```
data.groupby("Pclass")["Age"].count()
```

Out[14]:
```
Pclass
1    186
2    173
3    355
Name: Age, dtype: int64
```

Sorting

In [15]:
```
data.sort_values(by="Age").head()
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	NaN	C
755	756	1	2	Hamalainen, Master. Viljo	male	0.67	1	1	250649	14.5000	NaN	S
644	645	1	3	Baclini, Miss. Eugenie	female	0.75	2	1	2666	19.2583	NaN	C
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	2666	19.2583	NaN	C
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.0000	NaN	S

How to reshape the layout of tables

- Sort table rows
- Sort the Titanic dataset by 'Pclass' in descending order and, for rows with the same 'Pclass', sort by 'Age' in descending order.
- Display the first few rows of the sorted DataFrame.

In [19]:
```
# for descending order
data.sort_values(by=['Pclass', 'Age'], ascending=False).head()
```

Out[19]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	347060	7.7750	NaN	S
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q
280	281	0	3	Duane, Mr. Frank	male	65.0	0	0	336439	7.7500	NaN	Q
483	484	1	3	Turkula, Mrs. (Hedwig)	female	63.0	0	0	4134	9.5875	NaN	S
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0	0	0	345364	6.2375	NaN	S

In [21]:
```
# for ascending order
data.sort_values(by=['Pclass', 'Age'], ascending=True).head()
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
305	306	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500	C22 C26	S
297	298	0	1	Allison, Miss. Helen Loraine	female	2.00	1	2	113781	151.5500	C22 C26	S
445	446	1	1	Dodge, Master. Washington	male	4.00	0	2	33638	81.8583	A34	S
802	803	1	1	Carter, Master. William Thornton II	male	11.00	1	2	113760	120.0000	B96 B98	S
435	436	1	1	Carter, Miss. Lucile Polk	female	14.00	1	2	113760	120.0000	B96 B98	S

How to manipulate textual data

- Convert Text to Lowercase
  - Convert the "Name" column to lowercase.

In [29]:
```
data_name=data["Name"].str.lower()
print(data_name)
```

```
0                                braund, mr. owen harris
1      cumings, mrs. john bradley (florence briggs th...
2                                heikkinen, miss. laina
3           futrelle, mrs. jacques heath (lily may peel)
4                              allen, mr. william henry
                              ...
886                              montvila, rev. juozas
887                       graham, miss. margaret edith
888           johnston, miss. catherine helen "carrie"
889                              behr, mr. karl howell
890                                dooley, mr. patrick
Name: Name, Length: 891, dtype: object
```

2. Split Text Using a Delimiter

- Split the "Name" column into two parts using a comma as the delimiter

In [31]:
```
name_split = data['Name'].str.split(",")
name_split
```

Out[31]:
```
0                              [Braund,  Mr. Owen Harris]
1      [Cumings,  Mrs. John Bradley (Florence Briggs ...
2                               [Heikkinen,  Miss. Laina]
3        [Futrelle,  Mrs. Jacques Heath (Lily May Peel)]
4                             [Allen,  Mr. William Henry]
                              ...
886                              [Montvila,  Rev. Juozas]
887                       [Graham,  Miss. Margaret Edith]
888          [Johnston,  Miss. Catherine Helen "Carrie"]
889                              [Behr,  Mr. Karl Howell]
890                                [Dooley,  Mr. Patrick]
Name: Name, Length: 891, dtype: object
```

3. Extract Surnames from Names

- Extract surnames from the "Name" column.

In [32]:
```
data["Surename"]=data["Name"].str.split(",").str.get(0)
```

In [33]:
```
data["Surename"]
```

Out[33]:
```
0         Braund
1        Cumings
2      Heikkinen
3       Futrelle
4          Allen
         ...
886     Montvila
887       Graham
888     Johnston
889         Behr
890       Dooley
Name: Surename, Length: 891, dtype: object
```

Filter Rows Based on Substring

- Filter rows containing the substring "Countess" in the "Name" column.

In [37]:
```
data[data["Name"].str.contains("Countess")]
```

Out[37]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Surename
759	760	1	1	Rothes, the Countess. of (Lucy Noel Martha Dye...	female	33.0	0	0	110152	86.5	B77	S	Rothes

In [38]:
```
data["Name"].str.contains("Countess")
```

Out[38]:
```
0      False
1      False
2      False
3      False
4      False
       ...
886    False
887    False
888    False
889    False
890    False
Name: Name, Length: 891, dtype: bool
```

In [40]:
```
data["Name"].str.len()
```

Out[40]:
```
0      23
1      51
2      22
3      44
4      24
       ..
886    21
887    28
888    40
889    21
890    19
Name: Name, Length: 891, dtype: int64
```

5.Find the Index of the Longest String

- Find the index of the longest string in the "Name" column.

In [42]:
```
data["Name"].str.len().idxmax()
```

Out[42]:
```
307
```

In [43]:
```
data["Sex_short"] = data["Sex"].replace({"male": "M", "female": "F"})

data["Sex_short"]
```

Out[43]:
```
0      M
1      F
2      F
3      F
4      M
      ..
886    M
887    F
888    F
889    M
890    M
Name: Sex_short, Length: 891, dtype: object
```

7. Replace Substrings in a Column

- Replace substrings in the "Sex" column.

In [ ]:
```
data["Sex_short"] = data["Sex"].str.replace("female", "F")
data["Sex_short"] = data["Sex_short"].str.replace("male", "M")
```

In [45]:
```
data["Sex_short"]
```

Out[45]:
```
0      M
1      F
2      F
3      F
4      M
      ..
886    M
887    F
888    F
889    M
890    M
Name: Sex_short, Length: 891, dtype: object
```

In [ ]: