Adding attributes

In [3]:
```
# importing pandas as pd
import pandas as pd
# Creating the DataFrame
df = pd.DataFrame({'Date':['10/2/2011', '11/2/2011', '12/2/2011', '13/2/2011'],
                   'Event':['Music', 'Poetry', 'Theatre', 'Comedy'],
                   'Cost':[10000, 5000, 15000, 2000]})
# Print the dataframe
print(df)
```

```
        Date    Event   Cost
0  10/2/2011    Music  10000
1  11/2/2011   Poetry   5000
2  12/2/2011  Theatre  15000
3  13/2/2011   Comedy   2000
```

In [4]:
```
#using apply function to create a new column
df['Discounted_Price'] = df.apply(lambda row: row.Cost * 0.9, axis = 1)
# Print the DataFrame after addition of new column
print(df)
```

```
        Date    Event   Cost  Discounted_Price
0  10/2/2011    Music  10000            9000.0
1  11/2/2011   Poetry   5000            4500.0
2  12/2/2011  Theatre  15000           13500.0
3  13/2/2011   Comedy   2000            1800.0
```

In [28]:
```
df = pd.DataFrame({'Name':['John','Ted','Dove','Brad','Rex'],
                   'Salary':[44000, 35000, 75000, 20000,6000]})
# Print the dataframe
print(df)
```

```
   Name  Salary
0  John   44000
1   Ted   35000
2  Dove   75000
3  Brad   20000
4   Rex    6000
```

In [29]:
```
def salary_stats(value):
    if value < 10000:
        return "very low"
    elif 10000 <= value < 25000:
        return "low"
    elif 25000 <= value < 40000:
        return "average"
    elif 40000 <= value < 50000:
        return "better"
    elif value >= 50000:
        return "very good"

df['salary_stats'] = df['Salary'].map(salary_stats)
df
```

Out[29]:

	Name	Salary	salary_stats
0	John	44000	better
1	Ted	35000	average
2	Dove	75000	very good
3	Brad	20000	low
4	Rex	6000	very low

In [85]:

```
import pandas as pd
data = pd.DataFrame({
    'Name' : ['A', 'B', 'C', 'D','E', 'F'],
    'Education' : ['High School', 'Masters', 'Doctorate', 'Bachelors','Masters', 'I
data
```

Out[85]:

	Name	Education
0	A	High School
1	B	Masters
2	C	Doctorate
3	D	Bachelors
4	E	Masters
5	F	High School

Binary encoding

In [86]:

```
education_data = pd.get_dummies(data.Education)
print(education_data)
```

```
   Bachelors  Doctorate  High School  Masters
0          0          0            1        0
1          0          0            0        1
2          0          1            0        0
3          1          0            0        0
4          0          0            0        1
5          0          0            1        0
```

Ranking Transformation

In [80]:

```
education_map = {
    'High School' : 1,
    'Bachelors' : 2,
    'Masters': 3,
    'Doctorate': 4
}
education_data = data['Education'].map(education_map)
data['Education'] = education_data
data
```

Out[80]:

	Name	Education
0	A	1
1	B	3
2	C	4
3	D	2
4	E	3
5	F	1

In [84]:

```
education_map = {
    'High School' : 12,
    'Bachelors' : 16,
    'Masters': 18,
    'Doctorate': 21
}
education_data = data['Education'].map(education_map)
data['Education'] = education_data
data
```

Out[84]:

	Name	Education
0	A	12
1	B	18
2	C	21
3	D	16
4	E	18
5	F	12

Adding data objects- rows

In [71]:

```
df.loc[len(df.index)]=['Hruthvik', 15000, 'low']
df
```

Out[71]:

	Name	Salary	salary_stats
0	John	44000	better
1	Ted	35000	average
2	Dove	75000	very good
3	Brad	20000	low
4	Rex	6000	very low
5	Hruthvik	15000	low
6	Hruthvik	15000	low
7	Hruthvik	15000	low

Combining two dataframes

In [48]:
```
import pandas as pd

d1 = {'Name': ['Pankaj', 'Meghna', 'Lisa'], 'Country': ['India', 'India', 'USA'],
df1 = pd.DataFrame(d1)
print('DataFrame 1:\n', df1,'\n')

df2 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Pankaj', 'Anupam', 'Amit']})
print('DataFrame 2:\n', df2,'\n')

df3 = pd.DataFrame({'Name': ['Priya'], 'Country': ['India'], 'Role': ['COO']})
print('DataFrame 3:\n', df3,'\n')
```

```
DataFrame 1:
      Name Country Role
0  Pankaj   India  CEO
1  Meghna   India  CTO
2    Lisa     USA  CTO

DataFrame 2:
    ID    Name
0   1  Pankaj
1   2  Anupam
2   3    Amit

DataFrame 3:
     Name Country Role
0  Priya   India  COO
```

In [49]:
```
same_cols_df = pd.concat([df1,df3],ignore_index=True)
same_cols_df
```

Out[49]:

	Name	Country	Role
0	Pankaj	India	CEO
1	Meghna	India	CTO
2	Lisa	USA	CTO
3	Priya	India	COO

In [50]:
```
a_df=df1.append(df2, ignore_index=True)
a_df
```

```
C:\Users\YASH\AppData\Local\Temp\ipykernel_13020\2048347772.py:1: FutureWarning: T
he frame.append method is deprecated and will be removed from pandas in a future v
ersion. Use pandas.concat instead.
  a_df=df1.append(df2, ignore_index=True)
```

Out[50]:

	Name	Country	Role	ID
0	Pankaj	India	CEO	NaN
1	Meghna	India	CTO	NaN
2	Lisa	USA	CTO	NaN
3	Pankaj	NaN	NaN	1.0
4	Anupam	NaN	NaN	2.0
5	Amit	NaN	NaN	3.0

In [51]:
```
c_df = pd.concat([df1,df2],ignore_index=True)
c_df
```

Out[51]:

	Name	Country	Role	ID
0	Pankaj	India	CEO	NaN
1	Meghna	India	CTO	NaN
2	Lisa	USA	CTO	NaN
3	Pankaj	NaN	NaN	1.0
4	Anupam	NaN	NaN	2.0
5	Amit	NaN	NaN	3.0

Default Merging - inner join

In [52]:
```
df_merged = df1.merge(df2)
print('Result:\n', df_merged)
```

```
Result:
      Name Country Role  ID
0  Pankaj   India  CEO   1
```

Merging DataFrames with Left, Right, and Outer Join

In [53]:
```
print('Result Left Join:\n', df1.merge(df2, how='left'))
print('Result Right Join:\n', df1.merge(df2, how='right'))
print('Result Outer Join:\n', df1.merge(df2, how='outer'))
```

```
Result Left Join:
      Name Country Role   ID
0  Pankaj   India  CEO  1.0
1  Meghna   India  CTO  NaN
2    Lisa     USA  CTO  NaN
Result Right Join:
      Name Country Role  ID
0  Pankaj   India  CEO   1
1  Anupam     NaN  NaN   2
2    Amit     NaN  NaN   3
Result Outer Join:
      Name Country Role   ID
0  Pankaj   India  CEO  1.0
1  Meghna   India  CTO  NaN
2    Lisa     USA  CTO  NaN
3  Anupam     NaN  NaN  2.0
4    Amit     NaN  NaN  3.0
```

Merging DataFrame on Specific Columns

In [56]:
```
dict1 = {'ID': [1, 2, 3],'Name': ['Pankaj', 'Meghna', 'Lisa'],  'Country': ['India
      'Role': ['CEO', 'CTO', 'CTO']}
df1 = pd.DataFrame(dict1)
```

```
df2 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Pankaj', 'Anupam', 'Amit']})

print(df1.merge(df2, on='ID'))
print('\n',df1.merge(df2, on='Name'))
```

```
   ID  Name_x Country Role  Name_y
0   1  Pankaj   India  CEO  Pankaj
1   2  Meghna   India  CTO  Anupam
2   3    Lisa     USA  CTO    Amit

    ID_x    Name Country Role  ID_y
0     1  Pankaj   India  CEO     1
```

In [7]:
```
# Package imports
import pandas as pd
import missingno as msno
%matplotlib inline
```

In [8]:
```
#Importing the required dataset

titanic_df = pd.read_csv("titanic.csv")
titanic_df
```

Out[8]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

In [10]:
```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   survived     891 non-null    int64  
 1   pclass       891 non-null    int64  
 2   sex          891 non-null    object 
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64  
 5   parch        891 non-null    int64  
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object 
 8   class        891 non-null    object 
 9   who          891 non-null    object 
 10  adult_male   891 non-null    bool   
 11  deck         203 non-null    object 
 12  embark_town  889 non-null    object 
 13  alive        891 non-null    object 
 14  alone        891 non-null    bool   
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
```

In [11]:
```
titanic_df.isnull()
```

Out[11]:

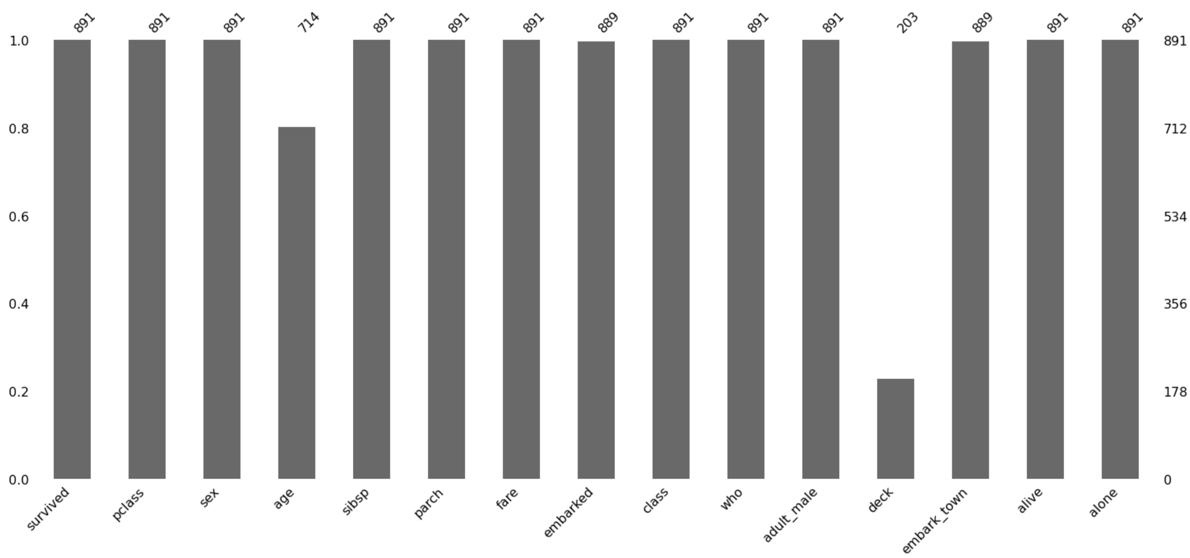
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	False	False	False	False	False	False	False	False	False	False	False	True
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	True
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	False	True
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	True	False	False	False	False	False	False	False	True
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	False	True

891 rows × 15 columns

Using Missingno to visualize missing values

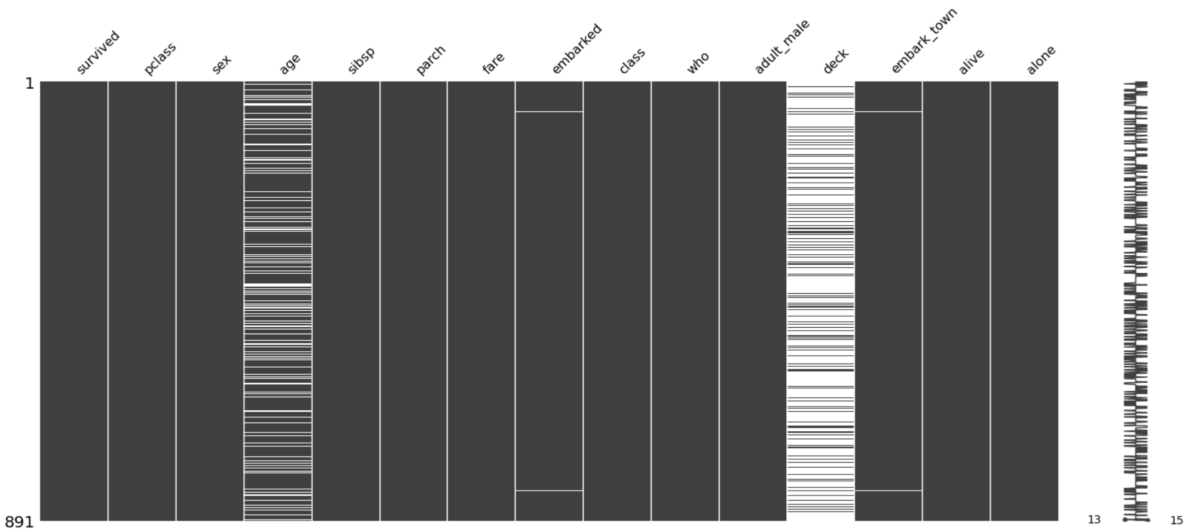
In [12]:
```
msno.bar(titanic_df)
```

Out[12]: <AxesSubplot:>



In [13]:
```
msno.matrix(titanic_df)
```

Out[13]: <AxesSubplot:>



Deleting the entire row

In [14]:
```
titanic_df.isnull().sum()
```

Out[14]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [15]:
```
df = titanic_df.dropna(axis=0)
df.isnull().sum()
```

Out[15]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 1 to 889
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   survived     182 non-null    int64  
 1   pclass       182 non-null    int64  
 2   sex          182 non-null    object 
 3   age          182 non-null    float64
 4   sibsp        182 non-null    int64  
 5   parch        182 non-null    int64  
 6   fare         182 non-null    float64
 7   embarked     182 non-null    object 
 8   class        182 non-null    object 
 9   who          182 non-null    object 
 10  adult_male   182 non-null    bool   
 11  deck         182 non-null    object 
 12  embark_town  182 non-null    object 
 13  alive        182 non-null    object 
 14  alone        182 non-null    bool   
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 20.3+ KB
```

Deleting the entire column

In [17]:

```
titanic_df.columns
```

Out[17]:

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [18]:

```
df = titanic_df.drop(['deck'],axis=1)
df.isnull().sum()
```

Out[18]:

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
embark_town      2
alive            0
alone            0
dtype: int64
```

Imputing the Missing Value

There are different ways of replacing the missing values.

Replacing With Arbitrary Value

If you can make an educated guess about the missing value then you can replace it with some arbitrary value using the following code.

In [20]:
```
titanic_df['deck'].unique()
```
Out[20]:
```
array([nan, 'C', 'E', 'G', 'D', 'A', 'B', 'F'], dtype=object)
```

In [21]:
```
titanic_df['deck'] = titanic_df['deck'].fillna('C')
```

In [23]:
```
titanic_df['deck'].isnull().sum() #missing values replaced
```
Out[23]:
```
0
```

In [24]:
```
titanic_df
```
Out[24]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

Replacing With Mean

This is the most common method of imputing missing values of numeric columns.

In [25]:
```
mean = titanic_df['age'].mean()
print(mean)
#Replace the missing values for numerical columns with mean
titanic_df['age'] = titanic_df['age'].fillna(mean)
titanic_df['age']
```
```
29.69911764705882
```

Out[25]:
```
0      22.000000
1      38.000000
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: age, Length: 891, dtype: float64
```

Replacing With Mode

Mode is the most frequently occurring value. It is used in the case of categorical features.

In [26]:
```
titanic_df = pd.read_csv("titanic.csv")
#Replace the missing values for categorical columns with mode
mode = titanic_df['deck'].mode()[0]
print(mode)
titanic_df['deck'] = titanic_df['deck'].fillna(mode)
```

```
C
```

In [27]:
```
titanic_df['deck']
```

Out[27]:
```
0      C
1      C
2      C
3      C
4      C
      ..
886    C
887    B
888    C
889    C
890    C
Name: deck, Length: 891, dtype: object
```

Replacing With Median

Median is the middlemost value. It's better to use the median value for imputation in the case of outliers.

In [28]:
```
titanic_df['age']= titanic_df['age'].fillna(titanic_df['age'].median())
titanic_df['age']
```

Out[28]:
```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       ...
886    27.0
887    19.0
888    28.0
889    26.0
890    32.0
Name: age, Length: 891, dtype: float64
```

Forward and backward filling of missing values

In [29]:
```
titanic_df = pd.read_csv("titanic2.csv")
titanic_df
```

Out[29]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0.0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1.0	1	female	38.0	1	0	71.2833	C	First	woman	False
2	NaN	3	female	NaN	0	0	7.9250	S	Third	woman	False
3	NaN	1	female	NaN	1	0	53.1000	S	First	woman	False
4	0.0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0.0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1.0	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0.0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1.0	1	male	26.0	0	0	30.0000	C	First	man	True
890	0.0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

In [30]:
```
new_df = titanic_df.fillna(method="ffill")
new_df
```

Out[30]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0.0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1.0	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1.0	3	female	38.0	0	0	7.9250	S	Third	woman	False
3	1.0	1	female	38.0	1	0	53.1000	S	First	woman	False
4	0.0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0.0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1.0	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0.0	3	female	19.0	1	2	23.4500	S	Third	woman	False
889	1.0	1	male	26.0	0	0	30.0000	C	First	man	True
890	0.0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

In [31]:
```
new_df = titanic_df.fillna(method="ffill",limit=1)
new_df
```

Out[31]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0.0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1.0	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1.0	3	female	38.0	0	0	7.9250	S	Third	woman	False
3	NaN	1	female	NaN	1	0	53.1000	S	First	woman	False
4	0.0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	..
886	0.0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1.0	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0.0	3	female	19.0	1	2	23.4500	S	Third	woman	False
889	1.0	1	male	26.0	0	0	30.0000	C	First	man	True
890	0.0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

In [33]:
```
new_df = titanic_df.fillna(method="bfill")
new_df
```

Out[33]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0.0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1.0	1	female	38.0	1	0	71.2833	C	First	woman	False
2	0.0	3	female	35.0	0	0	7.9250	S	Third	woman	False
3	0.0	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0.0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0.0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1.0	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0.0	3	female	26.0	1	2	23.4500	S	Third	woman	False
889	1.0	1	male	26.0	0	0	30.0000	C	First	man	True
890	0.0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

Numerosity Data Reduction

Random sampling

Example – Random sampling to speed up tuning

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [4]:
```
customer_df = pd.read_csv('Customer Churn.csv')
print(customer_df.shape)
print(customer_df.Churn.value_counts())
```

```
(3150, 9)
0    2655
1     495
Name: Churn, dtype: int64
```

In [5]:
```
customer_df_rs = customer_df.sample(1000,random_state=1)
y=customer_df_rs['Churn']
Xs = customer_df_rs.drop(columns=['Churn'])
print(customer_df_rs.shape)
```

```
(1000, 9)
```

In [6]:
```
customer_df_rs
```

Out[6]:

	Call Failure	Complains	Subscription Length	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Status	Churn
2001	0	0	37	0	0	0	0	0	0
943	0	0	24	2515	50	0	23	1	0
1611	4	0	37	2048	54	33	19	0	0
403	8	0	35	4018	58	0	30	1	0
1301	0	0	43	273	9	15	8	0	0
...	...	...	...	...	...	...	...	...	...
446	10	0	12	8753	163	62	35	1	0
2182	8	0	40	703	13	16	6	1	0
709	5	0	38	4325	82	0	22	1	0
1721	5	0	35	6603	70	110	38	1	0
1227	5	0	38	2043	40	37	33	0	1

1000 rows × 9 columns

In [49]:
```
print(customer_df_rs.Churn.value_counts())
```

```
0    856
1    144
Name: Churn, dtype: int64
```

Stratified sampling

Example – Stratified sampling for imbalanced dataset

In [59]:

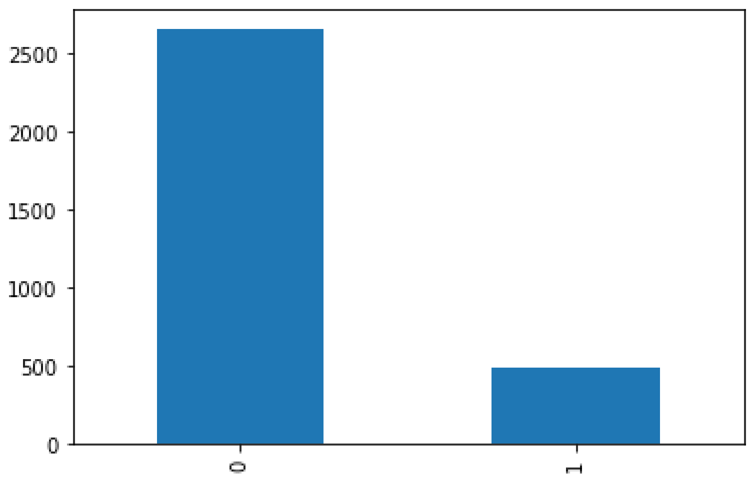
```
n,s=len(customer_df),1000
print(n,s)
r = s/n
print('Ratio of each Churn class in sample:',r)
sample_df = customer_df.groupby('Churn').apply(lambda sdf: sdf.sample(round(len(sd
print(sample_df.Churn.value_counts())
```

```
3150 1000
Ratio of each Churn class in sample: 0.31746031746031744
0    843
1    157
Name: Churn, dtype: int64
```

In [60]:

```
customer_df.Churn.value_counts().plot.bar()
```

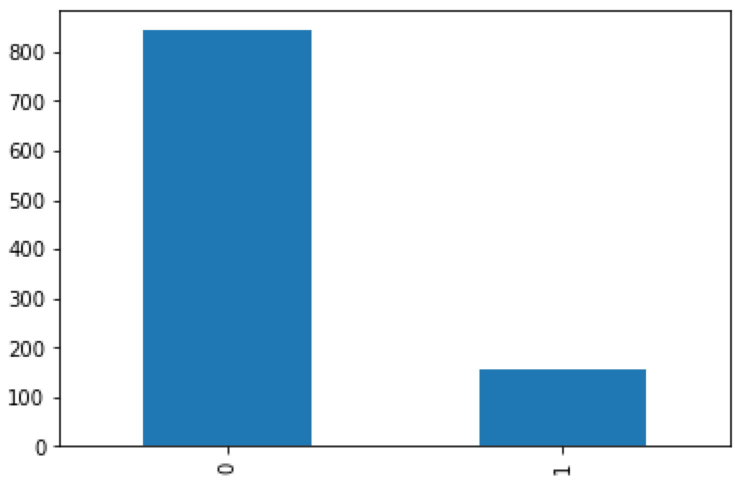
Out[60]: <AxesSubplot:>



In [61]:

```
sample_df.Churn.value_counts().plot.bar()
```

Out[61]: <AxesSubplot:>



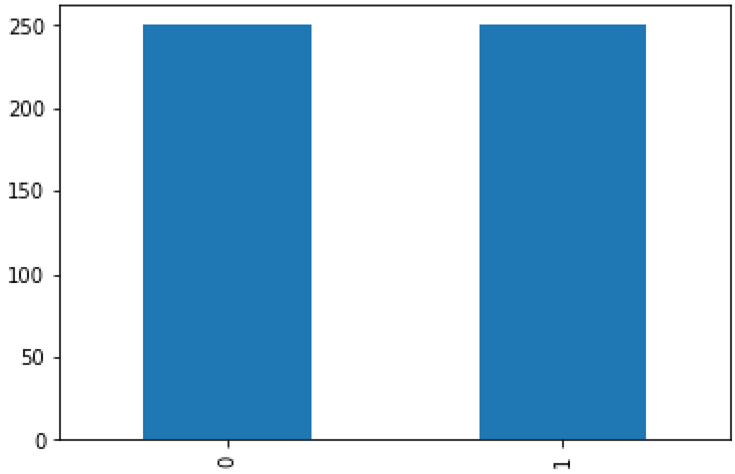
Random Over/Under-sampling

In [65]:
```
n,s=len(customer_df),500
sample_df = customer_df.groupby('Churn').apply(lambda sdf: sdf.sample(250))
print(sample_df.Churn.value_counts())
```

```
0    250
1    250
Name: Churn, dtype: int64
```

In [66]:
```
sample_df.Churn.value_counts().plot.bar()
```

Out[66]: <AxesSubplot:>



In [67]:
```
sample_df
```

Out[67]:

Churn		Call Failure	Complains	Subscription Length	Seconds of Use	Frequency of use	Frequency of SMS	Distinct Called Numbers	Status
0	**1815**	0	0	30	6195	74	171	23	1
	410	4	0	35	5738	87	0	7	1
	1455	4	0	33	3290	68	14	21	1
	2086	0	0	33	1360	38	31	18	0
	506	0	0	38	1760	29	264	3	1
...	**...**	...	...	...	...	...	...	...	...
1	**2178**	8	0	40	498	11	12	6	1
	1599	0	0	7	0	0	0	0	1
	1476	2	1	30	2505	27	0	9	0
	1382	0	1	28	0	0	0	0	1
	368	2	0	40	180	8	11	5	0

500 rows × 9 columns

Outliers Detection and handling

In [2]:
```
#Importing the necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

In [3]:
```
titanic_df = pd.read_csv("titanic.csv")
titanic_df
```

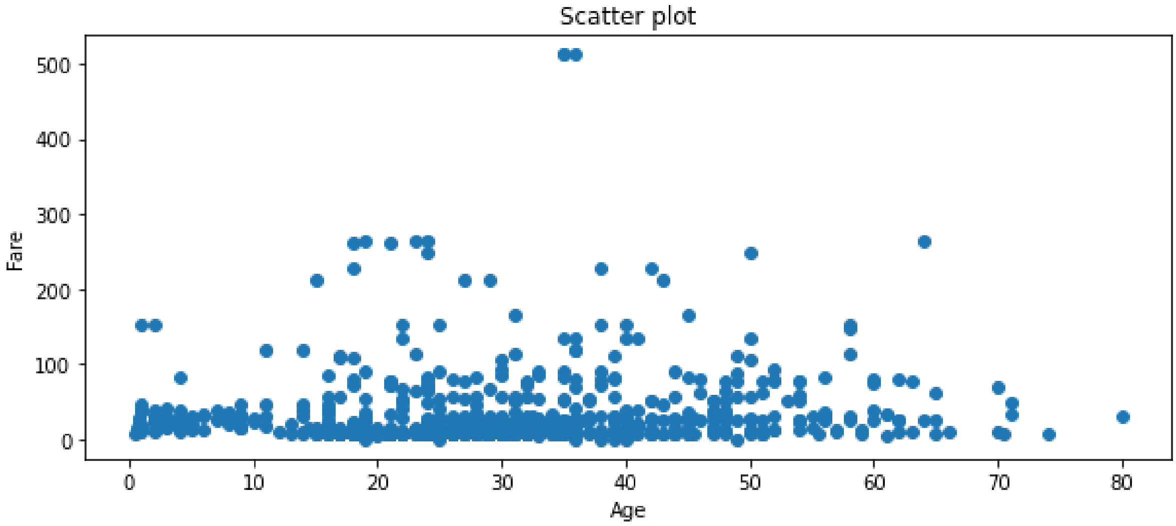
Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

Scatter plot to detect outliers

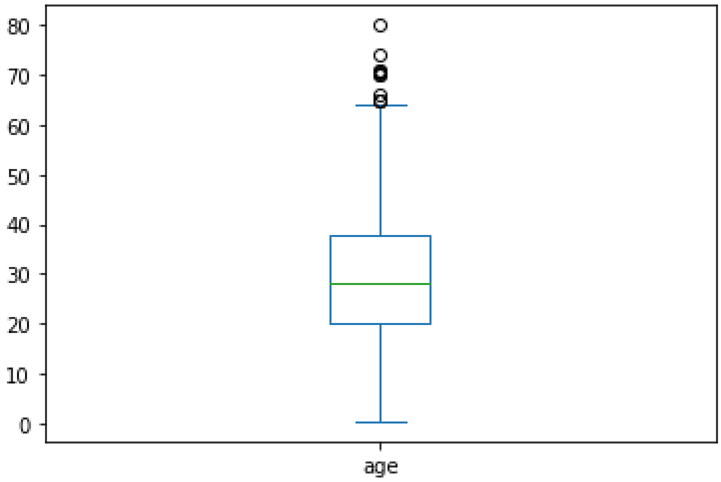
In [5]:
```
fig,ax = plt.subplots(figsize=(10,4))
ax.scatter(titanic_df['age'],titanic_df['fare'])
ax.set_xlabel('Age')
ax.set_ylabel('Fare')
plt.title("Scatter plot")
plt.show()
```



Box plot to detect outliers

In [6]:
```
titanic_df['age'].plot(kind='box')
```

Out[6]: <AxesSubplot:>



In [7]:
```
# finding the 1st quartile
q1 = titanic_df["age"].quantile(0.25)

# finding the 3rd quartile
q3 = titanic_df['age'].quantile(0.75)

# finding the iqr region
iqr = q3-q1

# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
```

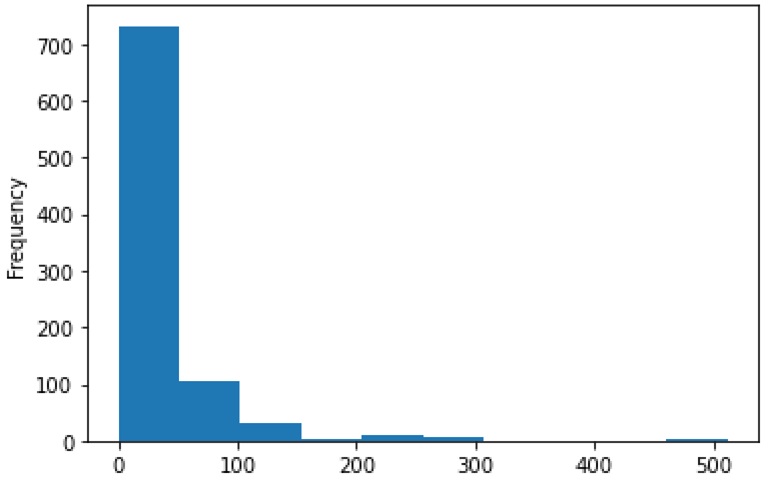
In [8]:
```
age_arr = titanic_df["age"]
outliers = age_arr[(age_arr <= lower_bound) | (age_arr >= upper_bound)]
print('The following are the outliers in the boxplot of age:\n',outliers)
```

```
The following are the outliers in the boxplot of age:
 33     66.0
54     65.0
96     71.0
116    70.5
280    65.0
456    65.0
493    71.0
630    80.0
672    70.0
745    70.0
851    74.0
Name: age, dtype: float64
```

Histogram plot to detect outliers

In [9]:
```
titanic_df['fare'].plot(kind='hist')
```

Out[9]: <AxesSubplot:ylabel='Frequency'>



Remove data objects with outliers

In [10]:
```
upperIndex = titanic_df[titanic_df['age']>upper_bound].index
titanic_df.drop(upperIndex,inplace=True)
lowerIndex = titanic_df[titanic_df['age']<lower_bound].index
titanic_df.drop(lowerIndex,inplace=True)
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 880 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     880 non-null    int64
 1   pclass       880 non-null    int64
 2   sex          880 non-null    object
 3   age          703 non-null    float64
 4   sibsp        880 non-null    int64
 5   parch        880 non-null    int64
 6   fare         880 non-null    float64
 7   embarked     878 non-null    object
 8   class        880 non-null    object
 9   who          880 non-null    object
 10  adult_male   880 non-null    bool
 11  deck         198 non-null    object
 12  embark_town  878 non-null    object
 13  alive        880 non-null    object
 14  alone        880 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 98.0+ KB
```

Replacing outliers with upper and lower cap:

Upper cap is 90% Lower cap is 1%

In [11]:
```
titanic_df = pd.read_csv("titanic.csv")
```

In [12]:
```
#upper and lower cap
# Winzorization method
fare_arr = titanic_df["fare"]
upper_cap = np.percentile(fare_arr,1)
lower_cap = np.percentile(fare_arr,99)
outliers = fare_arr[(fare_arr < upper_cap) | (fare_arr > lower_cap)]
print('The following are the outliers in the boxplot of fare:\n',outliers)
```

```
The following are the outliers in the boxplot of fare:
 27     263.0000
88     263.0000
258    512.3292
311    262.3750
341    263.0000
438    263.0000
679    512.3292
737    512.3292
742    262.3750
Name: fare, dtype: float64
```

In [13]:
```
for i in titanic_df['fare']:
    if i<lower_bound :
        titanic_df['fare'] = titanic_df['fare'].replace(i,lower_cap)
    elif i>upper_bound :
        titanic_df['fare'] = titanic_df['fare'].replace(i,upper_cap)
```

In [14]:
```
titanic_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   survived     891 non-null    int64  
 1   pclass       891 non-null    int64  
 2   sex          891 non-null    object 
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64  
 5   parch        891 non-null    int64  
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object 
 8   class        891 non-null    object 
 9   who          891 non-null    object 
 10  adult_male   891 non-null    bool   
 11  deck         203 non-null    object 
 12  embark_town  889 non-null    object 
 13  alive        891 non-null    object 
 14  alone        891 non-null    bool   
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
```

Replacing outliers with Mean

In [17]:

```
titanic_df = pd.read_csv("titanic.csv")
```

In [19]:

```
m = np.mean(titanic_df['age'])
print('mean:',m)
for i in titanic_df['age']:
    if i<lower_bound or i>upper_bound :
        titanic_df['age'] = titanic_df['age'].replace(i,m)
```

```
mean: 29.081737106607342
```

Replacing outliers with median

In [25]:

```
titanic_df = pd.read_csv("titanic.csv")
```

In [26]:

```
q1 = titanic_df["age"].quantile(0.25)

# finding the 3rd quartile
q3 = titanic_df['age'].quantile(0.75)

# finding the iqr region
iqr = q3-q1

# finding upper and lower whiskers
upper_bound = q3+(1.5*iqr)
lower_bound = q1-(1.5*iqr)
```

In [27]:

```
m = titanic_df['age'].median()
print(m)
for i in titanic_df['age']:
    if i<lower_bound or i>upper_bound :
        titanic_df['age'] = titanic_df['age'].replace(i,m)
```

```
28.0
```