In [58]:

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
%matplotlib inline
```

In [59]:

```
titanic_train = pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")
```

In [60]:

```
titanic_train
```

Out[60]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [61]:

```
titanic_train.head()
```

Out[61]:

	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 [62]:

```
#If you want to know the datatypes of each and every column we use dtypes
titanic_train.dtypes
#object is equivalent to a string
```

Out[62]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [63]:

```
#whichever column we have a numerical value we get those columns when we do describe()
titanic_train.describe()
```

Out[63]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare

count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [65]:

```
#Filter out all the columns having a data type as an object
titanic_train[['Name','Sex','Ticket','Cabin','Embarked']].describe()
#freq in the dataset tell you about the frequency of top data in the dataset
```

Out[65]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S
freq	1	577	7	4	644

In [68]:

```
#If we want to select the columns dynamically we do as follows
titanic_train.dtypes[titanic_train.dtypes == "object"].index
```

Out[68]:

```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

In [ ]:

```
type(titanic_train.dtypes[titanic_train.dtypes == "object"])
#in case if the datatype is series we can access it through the index.
#in the above series we created ae have all the column names as index
```

In [69]:

```
a = titanic_train.dtypes[titanic_train.dtypes == "object"].index
a
```

Out[69]:

```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

In [71]:

```
titanic_train[a]
```

Out[71]:

	Name	Sex	Ticket	Cabin	Embarked
0	Braund, Mr. Owen Harris	male	A/5 21171	NaN	S
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	PC 17599	C85	C
2	Heikkinen, Miss. Laina	female	STON/O2. 3101282	NaN	S
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	C123	S
4	Allen, Mr. William Henry	male	373450	NaN	S
...	...	...	...	...	...
886	Montvila, Rev. Juozas	male	211536	NaN	S
887	Graham, Miss. Margaret Edith	female	112053	B42	S
888	Johnston, Miss. Catherine Helen "Carrie"	female	W./C. 6607	NaN	S
889	Behr, Mr. Karl Howell	male	111369	C148	C
890	Dooley, Mr. Patrick	male	370376	NaN	Q

891 rows × 5 columns

In [72]:

```
titanic_train['Survived'][10:21]
```

Out[72]:

```
10    1
11    1
12    0
```

```
13    0
14    0
15    1
16    0
17    1
18    0
19    1
20    0
Name: Survived, dtype: int64
```

In [73]:

```
#sorting the given column.In order to sort we have to give the data in the form of an iterable object
sorted(titanic_train["Name"])[5:10:2]
```

Out[73]:

```
['Adahl, Mr. Mauritz Nils Martin',
 'Ahlin, Mrs. Johan (Johanna Persdotter Larsson)',
 'Albimona, Mr. Nassef Cassem']
```

- Categorical is a function available inside pandas which will return total number of unique data present inside that column and its datatype and the unique dataset

In [76]:

```
#Now we try read the data , take one column and try select the first character in a data and store it in another
import numpy as np
char_cabin= titanic_train["Cabin"].astype(str)#Converting the data to string type
new_cabin = [cabin[0] for cabin in char_cabin] #Takes the first letter
new_cabin = pd.Categorical(new_cabin)
new_cabin
```

Out[76]:

```
['n', 'C', 'n', 'C', 'n', ..., 'n', 'B', 'n', 'C', 'n']
Length: 891
Categories (9, object): ['A', 'B', 'C', 'D', ..., 'F', 'G', 'T', 'n']
```

In [77]:

```
titanic_train["cabin_1"] = new_cabin
```

In [78]:

```
titanic_train
```

Out[78]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	cabin_1
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	n
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	n
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	n
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S	n
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S	B
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S	n
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q	n

891 rows × 13 columns

In [82]:

```
# if you want to select indexes of all null values for a particular column
# where will always returns the indexes of the satisfying condition
missing = np.where(titanic_train["Age"].isnull()==True)
missing
```

Out[82]:

```
(array([  5,  17,  19,  26,  28,  29,  31,  32,  36,  42,  45,  46,  47,
         48,  55,  64,  65,  76,  77,  82,  87,  95, 101, 107, 109, 121,
        126, 128, 140, 154, 158, 159, 166, 168, 176, 180, 181, 185, 186,
        196, 198, 201, 214, 223, 229, 235, 240, 241, 250, 256, 260, 264,
        270, 274, 277, 284, 295, 298, 300, 301, 303, 304, 306, 324, 330,
        334, 335, 347, 351, 354, 358, 359, 364, 367, 368, 375, 384, 388,
        409, 410, 411, 413, 415, 420, 425, 428, 431, 444, 451, 454, 457,
        459, 464, 466, 468, 470, 475, 481, 485, 490, 495, 497, 502, 507,
        511, 517, 522, 524, 527, 531, 533, 538, 547, 552, 557, 560, 563,
        564, 568, 573, 578, 584, 589, 593, 596, 598, 601, 602, 611, 612,
        613, 629, 633, 639, 643, 648, 650, 653, 656, 667, 669, 674, 680,
        692, 697, 709, 711, 718, 727, 732, 738, 739, 740, 760, 766, 768,
        773, 776, 778, 783, 790, 792, 793, 815, 825, 826, 828, 832, 837,
        839, 846, 849, 859, 863, 868, 878, 888]),)
```

In [83]:

```
#wanted to extract where the fare is high
titanic_train["Fare"]
```

Out[83]:

```
0        7.2500
1       71.2833
2        7.9250
3       53.1000
4        8.0500
         ...
886     13.0000
887     30.0000
888     23.4500
889     30.0000
890      7.7500
Name: Fare, Length: 891, dtype: float64
```

In [85]:

```
#finding out the max fare
max(titanic_train['Fare'])
```

Out[85]:

```
512.3292
```

In [84]:

```
np.where(titanic_train['Fare'] == max(titanic_train['Fare']))
```

Out[84]:

```
(array([258, 679, 737]),)
```

Now we will see about row selection

- There are 3 functions that are used for row selections:

1. **loc**
2. **iloc**
3. **ix**

iloc(integer location)

In [87]:

```
row_index=np.where(titanic_train["Fare"] == max(titanic_train["Fare"]))
```

In [90]:

```
#iloc is used for row selection
titanic_train.iloc[row_index]
```

Out[90]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	cabin_1
258	259	1	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C	n
679	680	1	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C	B
737	738	1	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	B101	C	B

In [93]:

```
#Name and cabin values for person with min age
rows = np.where(titanic_train["Age"] == min(titanic_train["Age"]))
rows
```

Out[93]:

```
(array([803]),)
```

In [96]:

```
titanic_train.iloc[rows][["Name","Cabin"]]
```

Out[96]:

	Name	Cabin
803	Thomas, Master. Assad Alexander	NaN

In [97]:

```
#Concatinating two columns
titanic_train["Family"] = titanic_train["SibSp"] + titanic_train["Parch"]
titanic_train["Family"]
most_family = np.where(titanic_train["Family"]==max(titanic_train["Family"]))
most_family
```

Out[97]:

```
(array([159, 180, 201, 324, 792, 846, 863]),)
```

In [98]:

```
titanic_train.iloc[most_family]
```

Out[98]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	cabin_1	Family
159	160	0	3	Sage, Master. Thomas Henry	male	NaN	8	2	CA. 2343	69.55	NaN	S	n	10
180	181	0	3	Sage, Miss. Constance Gladys	female	NaN	8	2	CA. 2343	69.55	NaN	S	n	10
201	202	0	3	Sage, Mr. Frederick	male	NaN	8	2	CA. 2343	69.55	NaN	S	n	10
324	325	0	3	Sage, Mr. George John Jr	male	NaN	8	2	CA. 2343	69.55	NaN	S	n	10
792	793	0	3	Sage, Miss. Stella Anna	female	NaN	8	2	CA. 2343	69.55	NaN	S	n	10
846	847	0	3	Sage, Mr. Douglas Bullen	male	NaN	8	2	CA. 2343	69.55	NaN	S	n	10
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.55	NaN	S	n	10

In [100…

```
#The differnece between list and Series is we will not able to see indexes in list
labels = ['a','b','c']
my_data = [10,20,30]
arr=np.array(my_data)
d = {'a': 10,'b': 20,"c":30}


print("labels: ",labels)
print(" My data ",my_data)
print("Dictionary" ,d)

# you can provide own indexes using index parameter
#Even though we change indexes , system will be able to remember the default indexes
pd.Series(my_data ,index=labels)
```

Out[100…

```
labels:  ['a', 'b', 'c']
 My data  [10, 20, 30]
Dictionary {'a': 10, 'b': 20, 'c': 30}
a    10
b    20
c    30
dtype: int64
```

In [101…

```
# we can try to convert dictionary into a dataframe
#It will repeat the data ,depends upon the number of indexes of u
d={"a" : "khjh","b":20,"c":30}
d.items
pd.DataFrame(d,index=['s','k','m'])
```

Out[101…

	a	b	c
s	khjh	20	30
k	khjh	20	30

m	khjh	20	30

In [102…

```
print("\nHolding objects from a dictionary\n",'-'*40,sep='')
print(pd.Series([type,sum,max]))
```

```
Holding objects from a dictionary
----------------------------------------
0              <class 'type'>
1    <built-in function sum>
2    <built-in function max>
dtype: object
```

In [104…

```
ser1 = pd.Series([1,2,3,4],index=[2,4,6,8])
ser2 = pd.Series([1,2,5,4],['CA','OR','NV','AZ'])
ser2
```

Out[104…

```
CA    1
OR    2
NV    5
AZ    4
dtype: int64
```

In [105…

```
ser1 = pd.Series([1,2,3,4],['CA','OR','CO','CA'])
ser2 = pd.Series([1,2,5,4],['CA','NV','AZ','OR'])
ser3 = ser1 + ser2
```

In [106…

```
ser1
```

Out[106…

```
CA    1
OR    2
CO    3
CA    4
dtype: int64
```

In [107…

```
ser2
```

Out[107…

```
CA    1
NV    2
AZ    5
OR    4
dtype: int64
```

In [108…

```
#when we try to add anything with NaN
ser1+ser2
```

Out[108…

```
AZ    NaN
CA    2.0
CA    5.0
CO    NaN
NV    NaN
OR    6.0
dtype: float64
```

In [ ]: