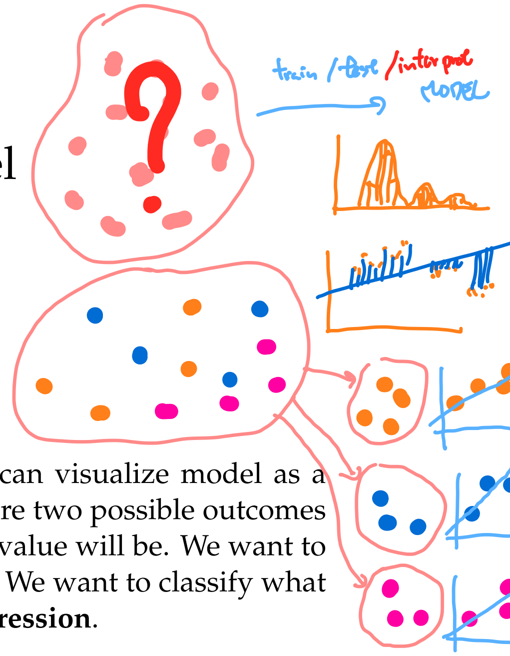
05_Logistic_Regression_Model

2021



1 Logistic Regression

Linear Regression predicts the value (y) of an unseen value (x). We can visualize model as a straight line. However, if we have to deal with the situation that there are two possible outcomes (ex: Head or Tail). Hence, we don't want to know how high or low the value will be. We want to know which outcome (H/T) it shall be. This is a classification problem. We want to classify what output it shall be from the given input. In this case we use **Logistic Regression**.

In real life, people use the classification to classify:

- Spam -or- Ham (e-mail)
- Love -or- Hate (product)
- Hit -or Miss

We can think about logistic regression as a modified linear regression where a curve is used instead of a line.

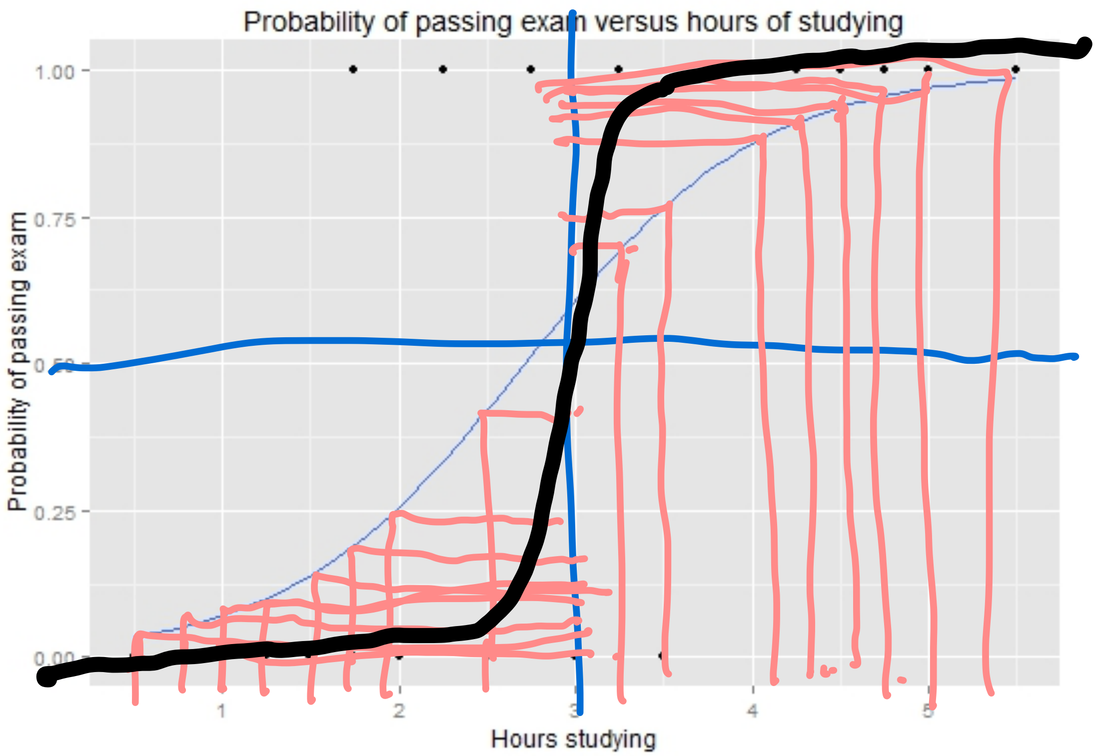


image source: wiki pedia

A characteristic “S”-shaped curve is also called **Sigmoid curve**. Logistic function is one that has the sigmoid curve.

$$S(x) = \frac{1}{1+e^{-x}}$$

There are also other functions.

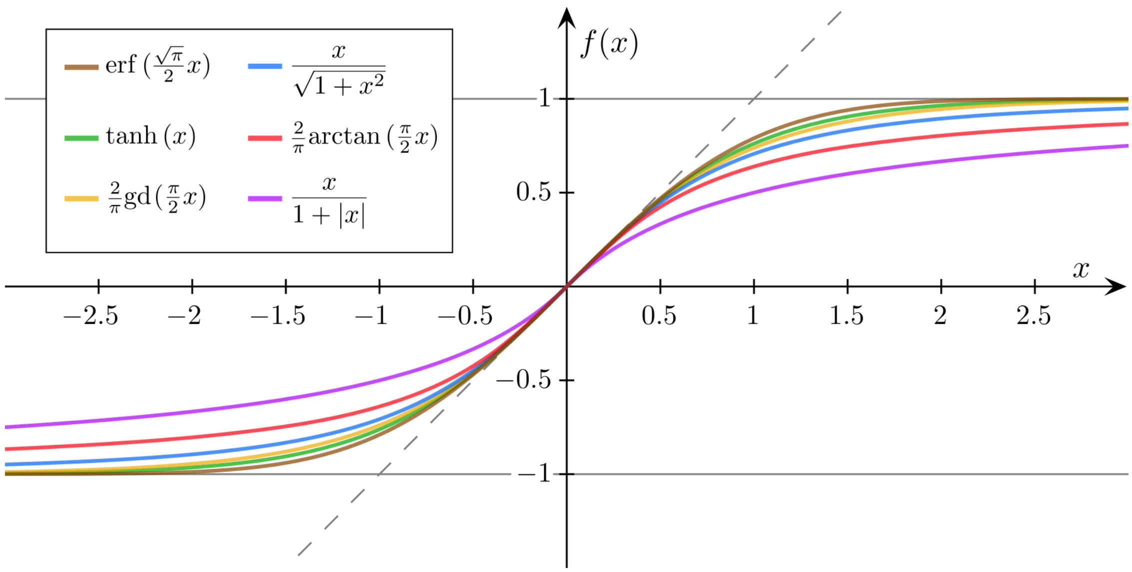


image source: wiki pedia

[14]:
```
#import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1.1 1. Load the data

[15]:
```
df = sns.load_dataset('titanic')
```

1.2 2. Visualize the data (the big picture)

[16]:
```
df.isnull()
```

[16]:
```
     survived  pclass    sex    age  sibsp  parch   fare  embarked  class  \
0       False   False  False  False  False  False  False     False  False
1       False   False  False  False  False  False  False     False  False
2       False   False  False  False  False  False  False     False  False
3       False   False  False  False  False  False  False     False  False
4       False   False  False  False  False  False  False     False  False
..        ...     ...    ...    ...    ...    ...    ...       ...    ...
886     False   False  False  False  False  False  False     False  False
887     False   False  False  False  False  False  False     False  False
888     False   False  False   True  False  False  False     False  False
889     False   False  False  False  False  False  False     False  False
890     False   False  False  False  False  False  False     False  False
```

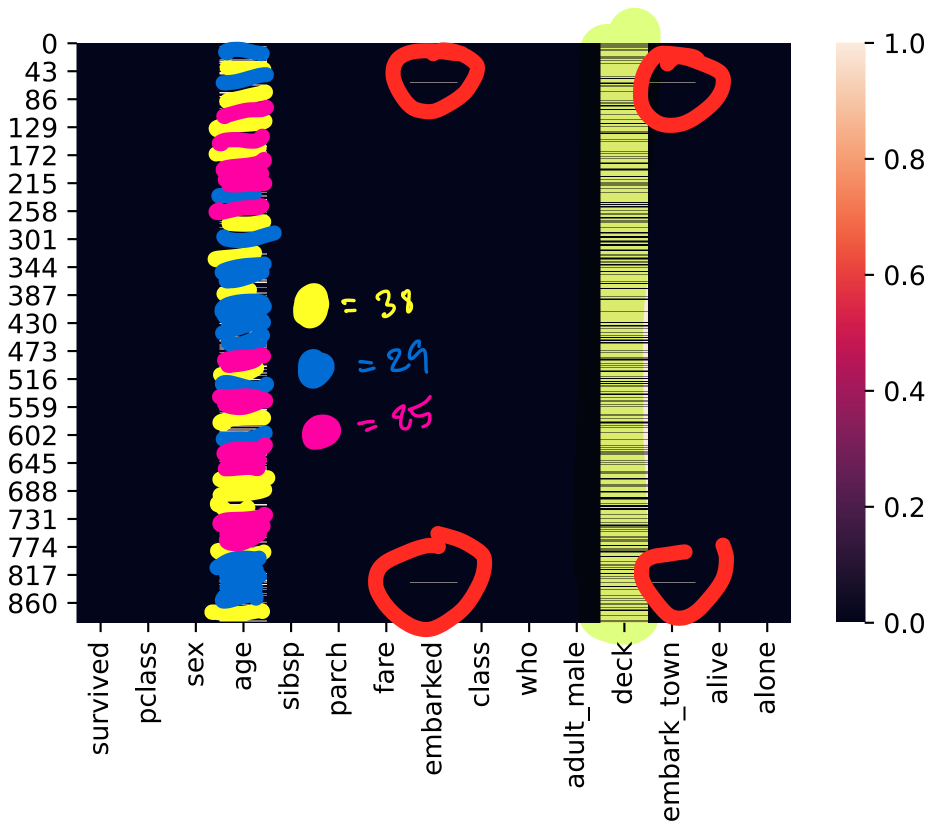
```
       who  adult_male   deck  embark_town  alive  alone
0    False       False   True        False  False  False
1    False       False  False        False  False  False
2    False       False   True        False  False  False
3    False       False  False        False  False  False
4    False       False   True        False  False  False
..     ...         ...    ...          ...    ...    ...
886  False       False   True        False  False  False
887  False       False  False        False  False  False
888  False       False   True        False  False  False
889  False       False  False        False  False  False
890  False       False   True        False  False  False

[891 rows x 15 columns]
```

[17]:

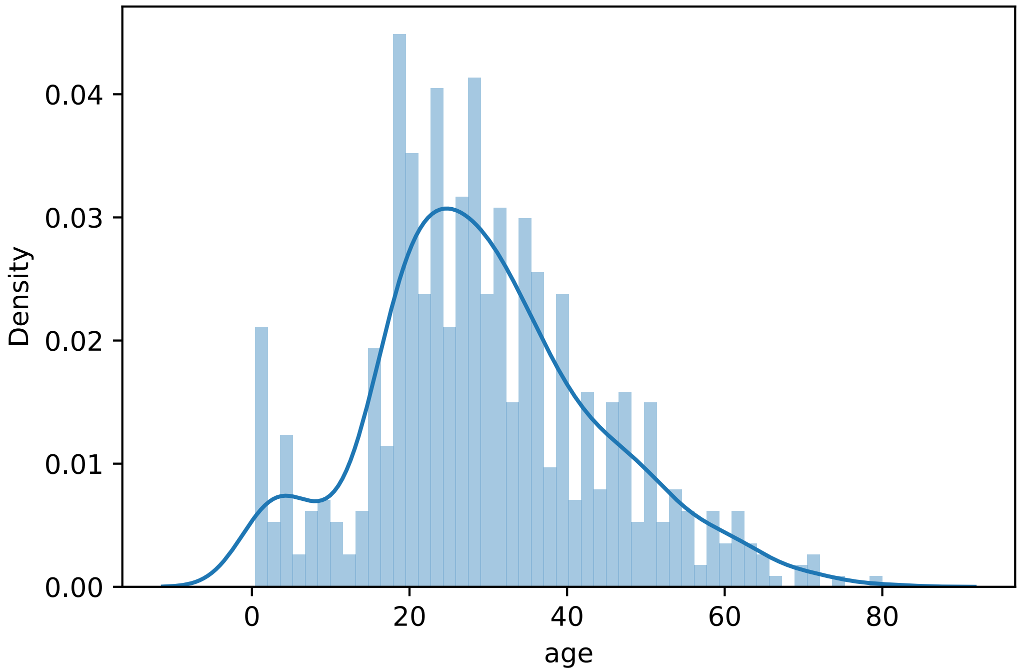
```python
sns.heatmap(df.isnull())
# We see missing data in age and deck
```

[17]: <AxesSubplot:>



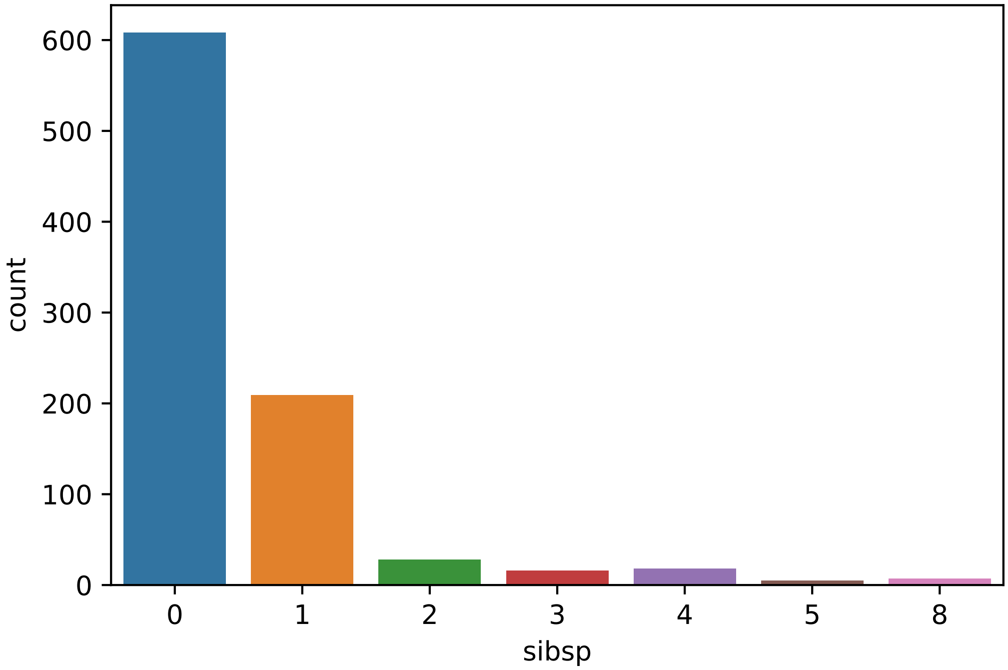
[18]:
```
sns.distplot(df['age'],bins=50)
# Mostly we have adult and a group of children
```

[18]: <AxesSubplot:xlabel='age', ylabel='Density'>



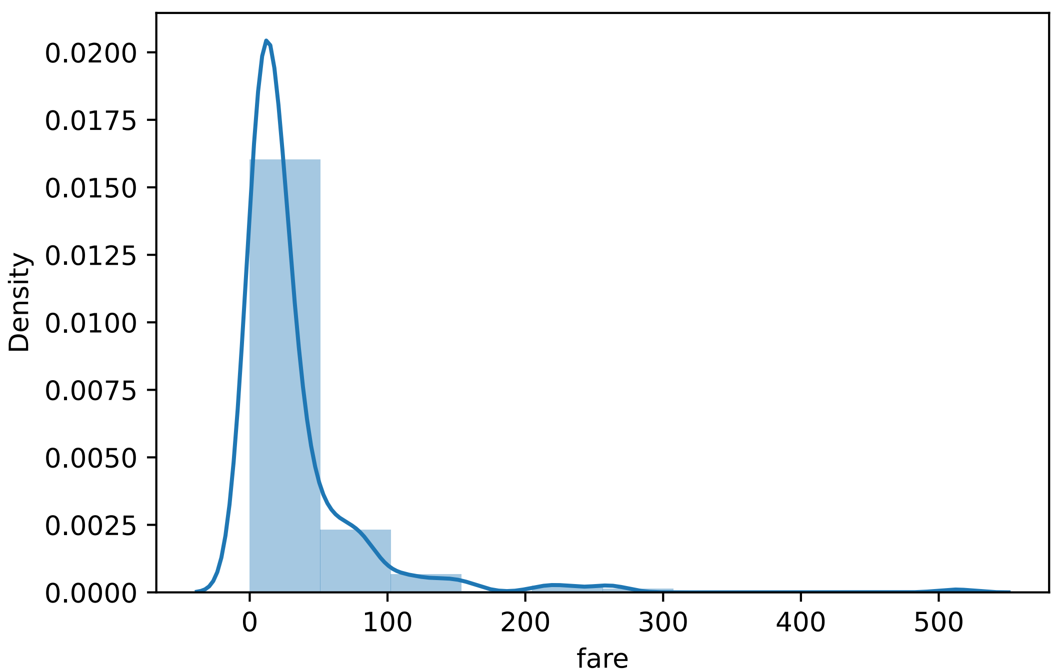
[19]:
```
sns.countplot(df['sibsp'])
# Most passengers don't have sibling or spouse
```

[19]: <AxesSubplot:xlabel='sibsp', ylabel='count'>



[20]:
```
# Here is what I asked you before
sns.distplot(df['fare'],bins=10)
```

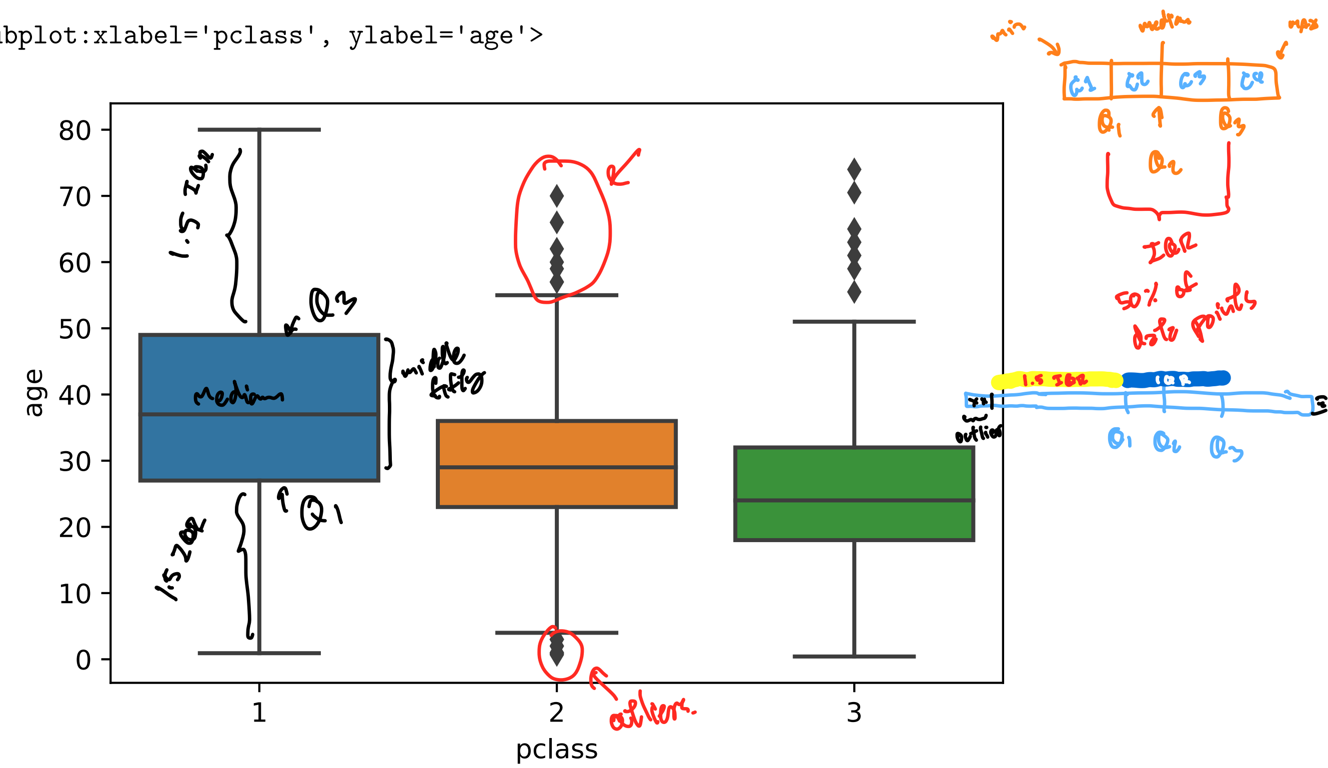
[20]: <AxesSubplot:xlabel='fare', ylabel='Density'>



1.3 3. Clean the data

[21]:
```
sns.boxplot(x='pclass',y='age',data=df)
```

[21]: <AxesSubplot:xlabel='pclass', ylabel='age'>



[22]:
```
c1mean = df[ df['pclass']==1]['age'].mean()
c1mean = int(c1mean)
c1mean
```

[22]: 38

[23]:
```
c2mean = df[ df['pclass']==2]['age'].mean()
c2mean = int(c2mean)
c2mean
```

[23]: 29

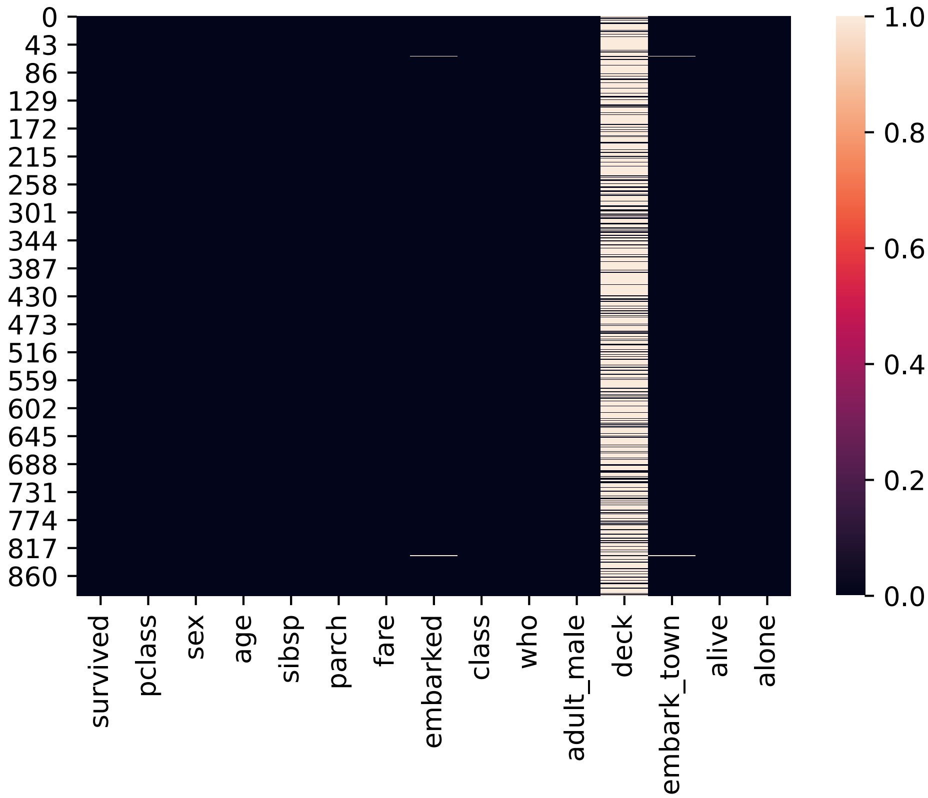
[24]:
```
c3mean = df[ df['pclass']==3]['age'].mean()
c3mean = int(c3mean)
c3mean
```

[24]: 25

[25]:
```
df.loc[(df['pclass']==1) & df['age'].isnull(),'age'] = c1mean
df.loc[(df['pclass']==2) & df['age'].isnull(),'age'] = c2mean
df.loc[(df['pclass']==3) & df['age'].isnull(),'age'] = c3mean
```

[26]:
```
sns.heatmap(df.isnull())
```

[26]: <AxesSubplot:>



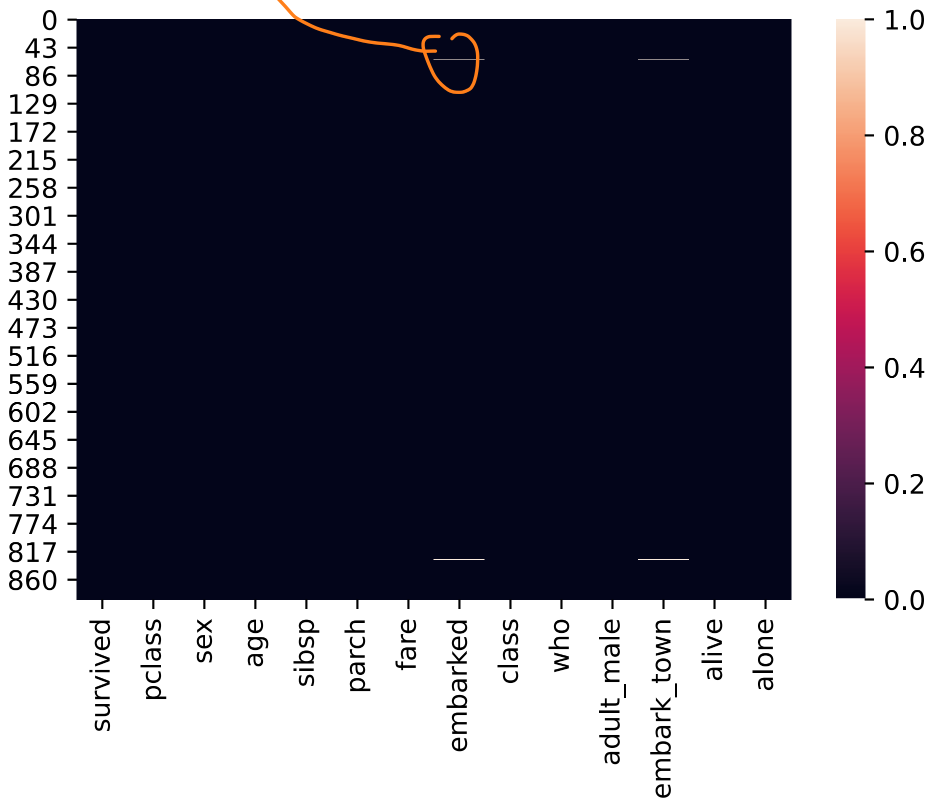
[27]:
```
# For deck, the missing too much to make any reasonable guess.

df.drop('deck',axis=1,inplace=True)
```

[28]:
```
sns.heatmap(df.isnull())
```

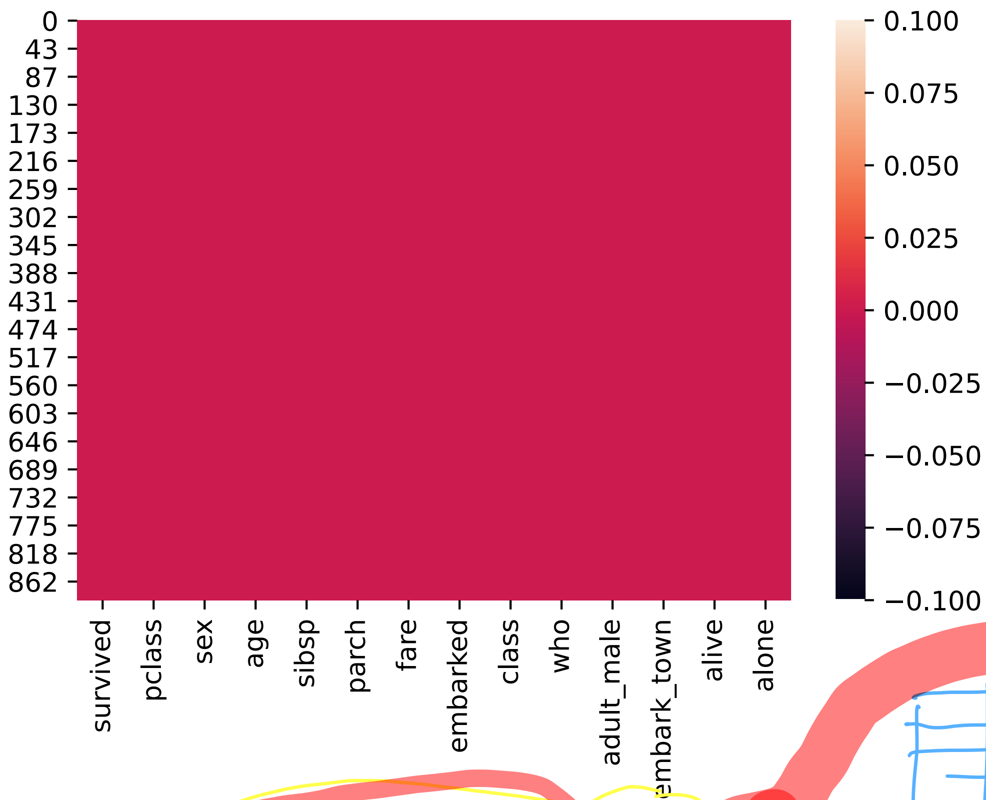
[28]: <AxesSubplot:>

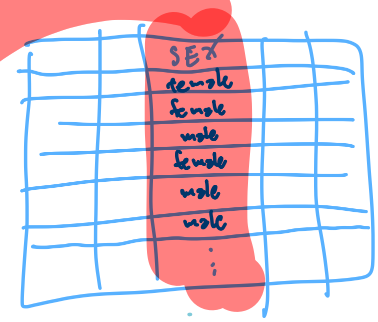
MAY replace with the mode



[29]:
```
# No major missing data now.. clean the minor ones
df.dropna(inplace=True)
sns.heatmap(df.isnull())
```

[29]:
```
<AxesSubplot:>
```





female

male

categorical

1.3.1 Let's convert the categorical variable to values

[30]:
```
pd.get_dummies(df['sex'])
```

[30]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
..	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

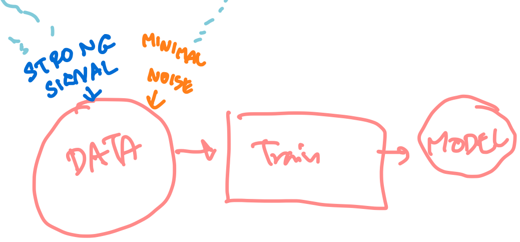
[889 rows x 2 columns]

SNR

signal to noise ratio

informative

garbage.



Since "female" provides information → aka signal
"male" provides the same info ... no
new information from having (female).
Hence male → noise.

DF 1 col.

```
[31]: # since we don't need both
      sex = pd.get_dummies(df['sex'],drop_first=True)
      sex
```

drop one, keep one.

```
[31]:      male
      0       1
      1       0
      2       0
      3       0
      4       1
      ..    ...
      886     1
      887     0
      888     0
      889     1
      890     1

      [889 rows x 1 columns]
```

```
[32]: pd.get_dummies(df['embarked'])
```

We need 2 columns to convey this information (for training).

[32]:

	C	Q	S
0	[redacted]	0	1
1	[redacted]	0	0
2	[redacted]	0	1
3	[redacted]	0	1
4	[redacted]	0	1
..	[redacted]	..	..
886	[redacted]	0	1
887	[redacted]	0	1
888	[redacted]	0	1
889	[redacted]	0	0
890	[redacted]	1	0

[889 rows x 3 columns]

DF - 2 col.

```
[33]: embark = pd.get_dummies(df['embarked'],drop_first = True)
      embark
```

drop C

keep Q, S.

```
[33]:    Q  S
      0    0  1
      1    0  0
      2    0  1
      3    0  1
      4    0  1
      ..  .. ..
      886  0  1
```

```
887  0  1
888  0  1
889  0  0
890  1  0

[889 rows x 2 columns]
```

[34]:
```
pd.get_dummies(df['alone'])
```

[34]:
```
     False  True
0        1     0
1        1     0
2        0     1
3        1     0
4        0     1
..     ...   ...
886      0     1
887      0     1
888      1     0
889      0     1
890      0     1

[889 rows x 2 columns]
```

Df 1 col

[35]:
```
isalone = pd.get_dummies(df['alone'],drop_first = True)
isalone.rename(columns={True:'isalone'})
```

[35]:
```
     isalone
0          0
1          0
2          1
3          0
4          1
..       ...
886        1
887        1
888        0
889        1
890        1

[889 rows x 1 columns]
```

[36]:
```
df.head(1)
```

[36]:
```
   survived  pclass   sex   age  sibsp  parch  fare embarked  class  who  \
0         0       3  male  22.0      1      0  7.25        S  Third  man
```

```
   adult_male   embark_town alive  alone
0        True   Southampton    no  False
```

[37]:
```
df = pd.concat([df,sex,embark,isalone],axis=1)
df.head(3)
```

DF (1) DF (1) DF (1)

[37]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third

     who  adult_male  embark_town alive  alone  male  Q  S  True
0    man        True  Southampton    no  False     1  0  1     0
1  woman       False    Cherbourg   yes  False     0  0  0     0
2  woman       False  Southampton   yes   True     0  0  1     1
```

[38]:
```
# We don't need the categorical columns that we just created dummy columns
# Adult_male is also an interpretation of age & sex, hence we don't need it.
# Embark_town is a text, we don't need it
df.
↪drop(['sex','embarked','class','adult_male','embark_town','who','alive','alone'],axis=1,inpla
```

[39]:
```
df.head(3)
```

[39]:
```
   survived  pclass   age  sibsp  parch     fare  male  Q  S  True
0         0       3  22.0      1      0   7.2500     1  0  1     0
1         1       1  38.0      1      0  71.2833     0  0  0     0
2         1       3  26.0      0      0   7.9250     0  0  1     1
```

1.4 4. Split the data

[40]:
```
X = df.drop(['survived'], axis = 1)
y = df.loc[:,'survived']
```

← features

← label

[41]:
```
# Check the dimension
X.shape
```

[41]: (889, 9)

[42]:
```
y.shape
```

[42]: (889,)

[43]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
↪3,random_state=161)
```

[44]:
```
X_train.shape
```

[44]: (622, 9)

[45]:
```
X_test.shape
```

[45]: (267, 9)

[46]:
```
y_train.shape
```

[46]: (622,)

[47]:
```
y_test.shape
```

[47]: (267,)

1.5 5. Create a Model

[48]:
```
from sklearn.linear_model import LogisticRegression
```

[49]:
```
lgm = LogisticRegression()
```

1.6 6. Train the Model

[50]:
```
lgm.fit(X_train,y_train)
```

[50]: LogisticRegression()

1.7 7. Test the Model

[51]:
```
predictions = lgm.predict(X_test)
```

1.8 8. Access the accuracy

One of the popular method to assess the model is to use a confusion matrix.

α = Type I error. : False Positive

β = Type II error.: False Negative

to be 0

High

		Actual class	
		Cat	Dog
Predicted class	Cat	5	2
	Dog	3	3

We have terms to call this:

α

		Actual class	
		P	N
Predicted class	P	TP	FP
	N	FN	TN

β

Where * P = Positive * N = Negative * TP = True Positive * FP = False Positive * TN = True Negative * FN = False Negative

[52]:
```
from sklearn.metrics import confusion_matrix
```

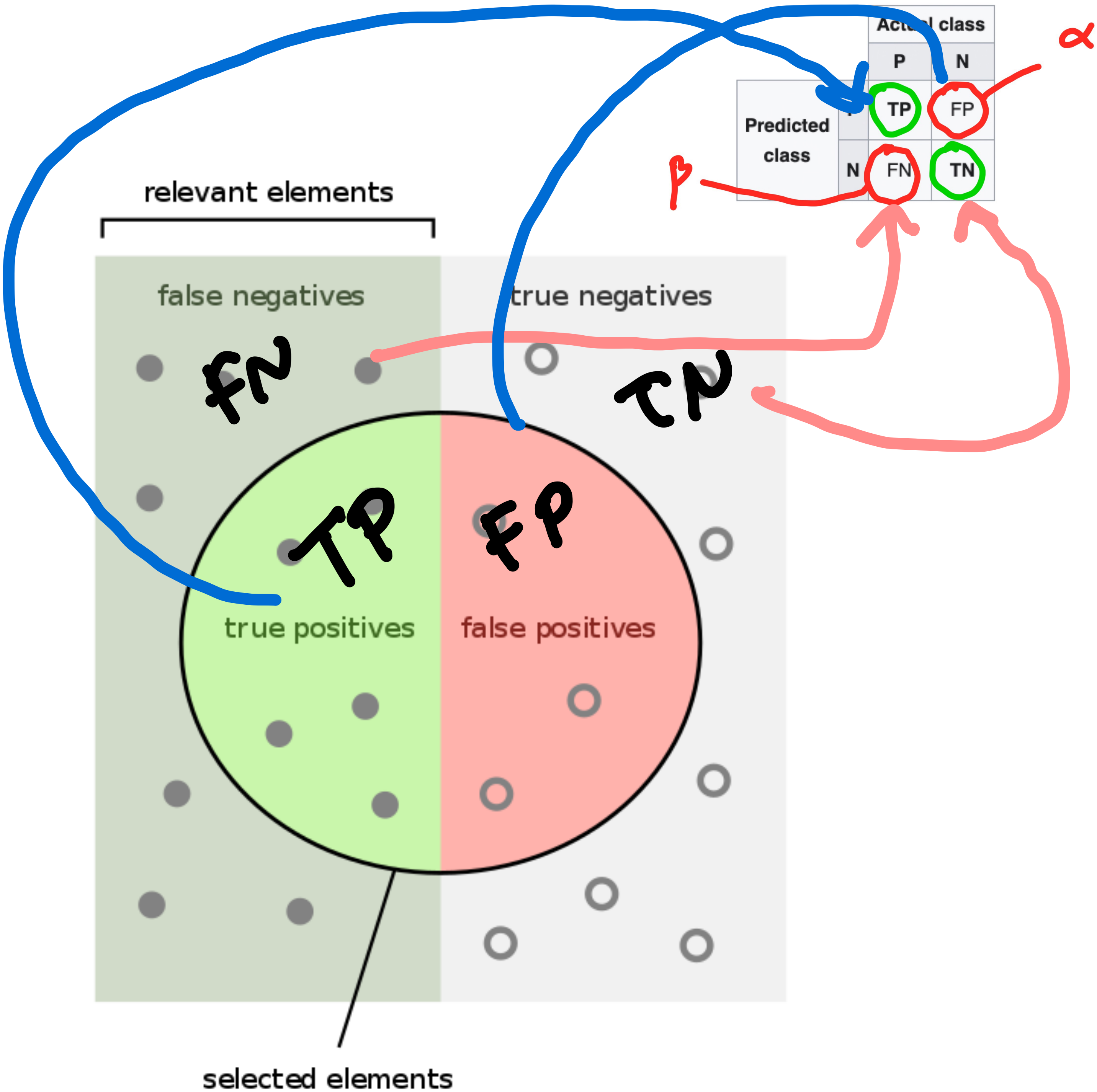
[53]:
```
print(confusion_matrix(y_test,predictions))
```

T P

```
[[147  24]
 [ 28  68]]
```

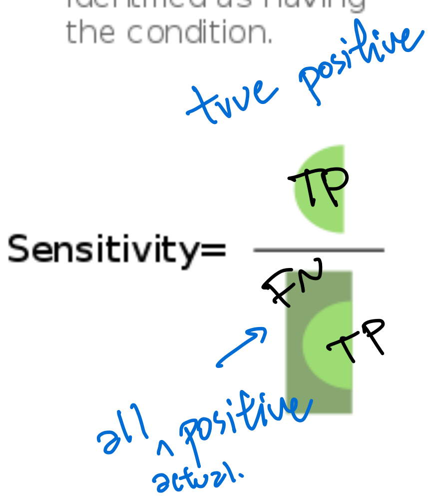
The confusion metrix may require reading to perform center interpretation. There are also other measurements that can we used.

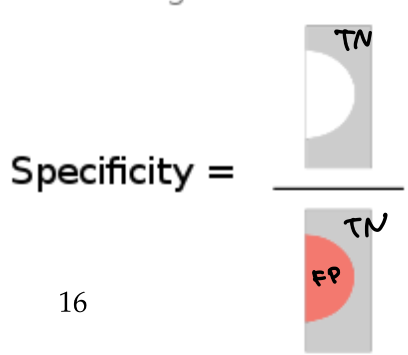
sensitivity & specificity



How many relevant items are selected? e.g. How many sick people are correctly identified as having the condition.

How many negative selected elements are truly negative? e.g. How many healthy peple are identified as not having the condition.





$$\frac{TN}{TN + FP}$$

Sensitivity : $\dfrac{TP}{TP + FN}$

If FN is 0, it means no false negative.
It means we catch all the positive ones.

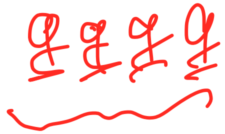
$$\frac{TP}{TP + 0} = 1.0$$

Specificity : $\dfrac{TN}{TN + FP}$

If FP is 0, it means no false positive.
It means you don't claim any innocent to be the criminal.

$$\frac{TN}{TN + 0} = 1.0$$

Extreme examples.



100

50 → COVID
50 → NO COVID

1. claim all as COVID 19 positive

$$\text{Sensitivity} = \frac{TP}{TP + \cancel{FN}^{\,0}} = 1.0$$

SENSITIVITY IS PERFECT.

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{0}{0 + FP} = 0$$

SPECIFICITY IS THE WORST POSSIBLE VALUE.

* High FP → precision will be low.

* High FN → recall will be low.

Image Source: Wiki Pedia

precision & recall

We have terms to call this:

		Actual class	
		P	N
Predicted class	P	TP	FP
	N	FN	TN

α

β

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

[54]:
```
from sklearn.metrics import classification_report
```

[55]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       171
           1       0.74      0.71      0.72        96

    accuracy                           0.81       267
   macro avg       0.79      0.78      0.79       267
weighted avg       0.80      0.81      0.80       267
```

[ ]: