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

In [2]:
```
df = pd.read_csv('titanic_train.csv')
df.head()
```

Out[2]:

	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 [3]:
```
df.shape
```

Out[3]:
```
(891, 12)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
```

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [6]:

```
df.describe()
```

Out[6]:

	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 [7]:

```
df.isnull().sum()
```

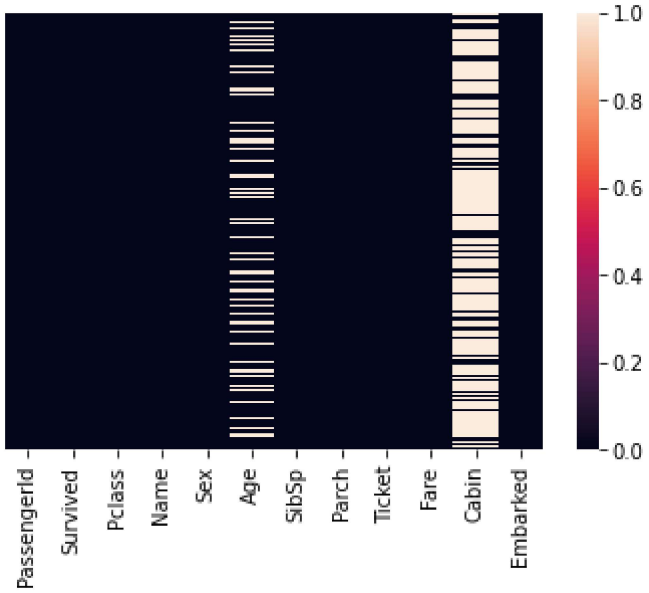
Out[7]:

```
PassengerId      0
Survived         0
Pclass           0
Name             0
```

```
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [10]:
```
sns.heatmap(df.isnull(),yticklabels=False)
```

Out[10]:
```
<AxesSubplot:>
```



As "Age" and "Cabin" are missing we should not drop it directly. If few numbers are missing we can impute with some other values if more are missing we can drop it

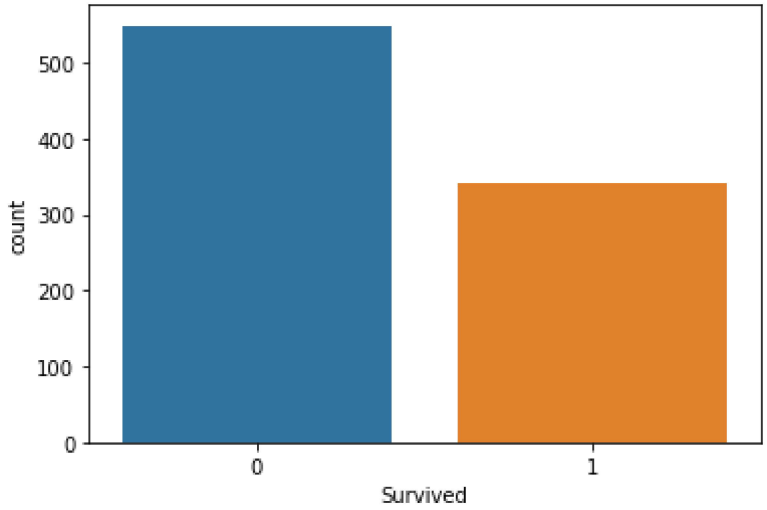
In [12]:
```
per_age_missing = df['Age'].isnull().sum() / len(df) * 100
print("Age data is missing: ",per_age_missing)
per_cabin_missing = df['Cabin'].isnull().sum()  /len(df)*100
print("Cabin data is missing: ",per_cabin_missing)
```

```
Age data is missing:  19.865319865319865
Cabin data is missing:  77.10437710437711
```

Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

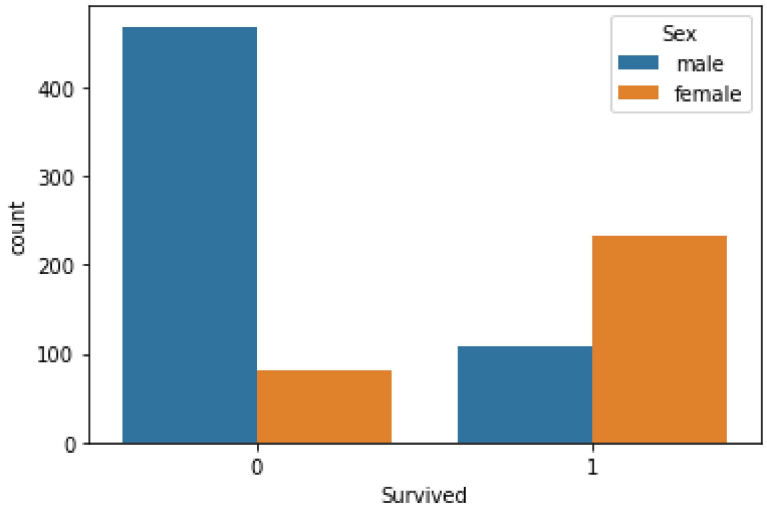
In [15]:
```
sns.countplot(x = 'Survived',data=df) #this indicates that more people have died 1-survived , 0-died
```

Out[15]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [16]:
```
sns.countplot(x='Survived',hue='Sex',data=df)
```

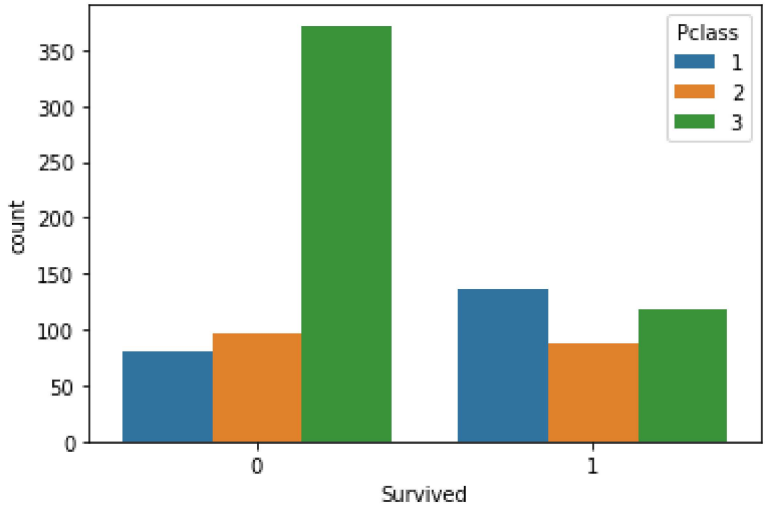
Out[16]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



This indicates that more male have died than female passengers and more females have survived

In [17]:
```
sns.countplot(x='Survived',hue='Pclass',data=df)
```

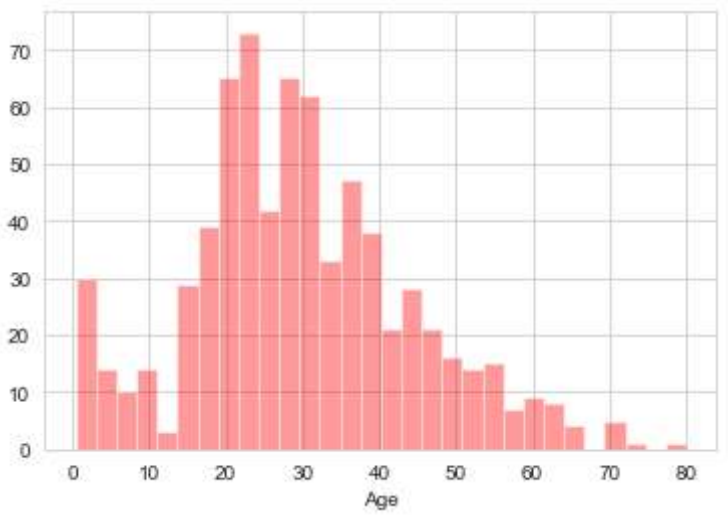
Out[17]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



This indicates that the passengers from 3rd class were died mostly.May be it is because that more people have travelled in third class as it is cheap. in 1st and 2nd class only rich people have been travelled.

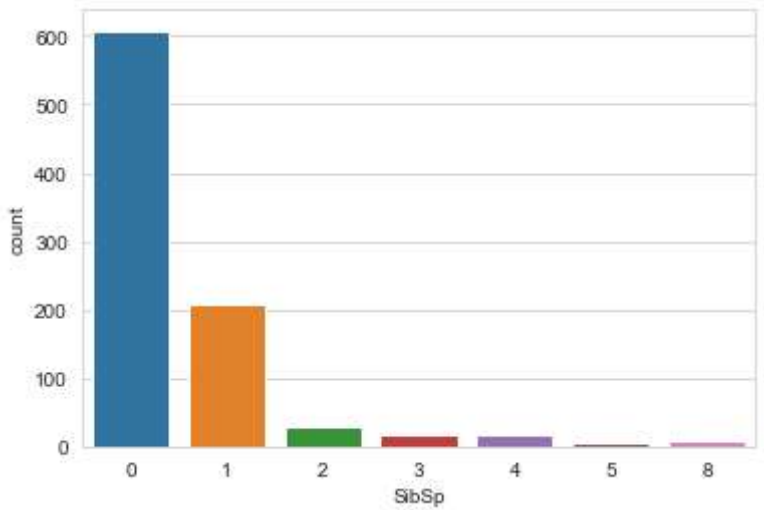
In [23]:
```
sns.set_style("whitegrid")
sns.distplot(df['Age'].dropna(),kde=False,bins=30,color='red')
```

Out[23]:
```
<AxesSubplot:xlabel='Age'>
```



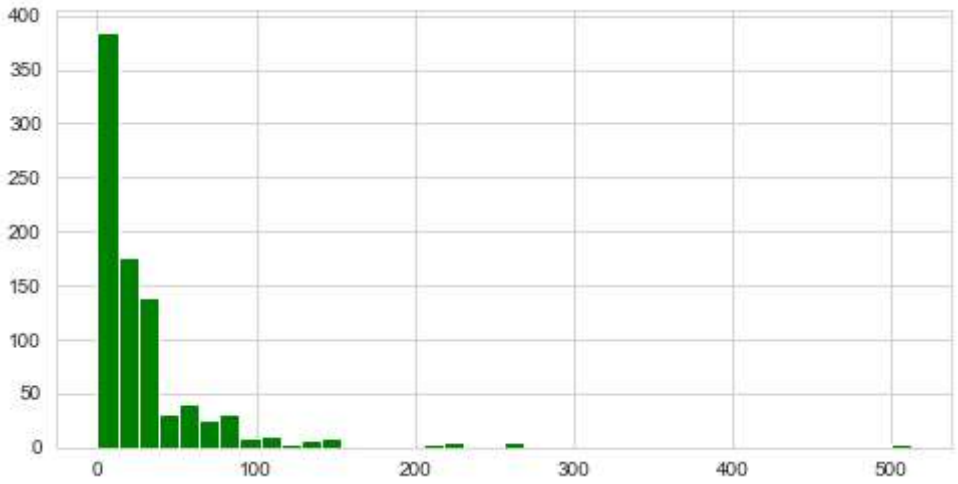
In [24]:
```
sns.countplot(x='SibSp',data=df)
```

Out[24]: <AxesSubplot:xlabel='SibSp', ylabel='count'>



In [25]:
```
df['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

Out[25]: <AxesSubplot:>

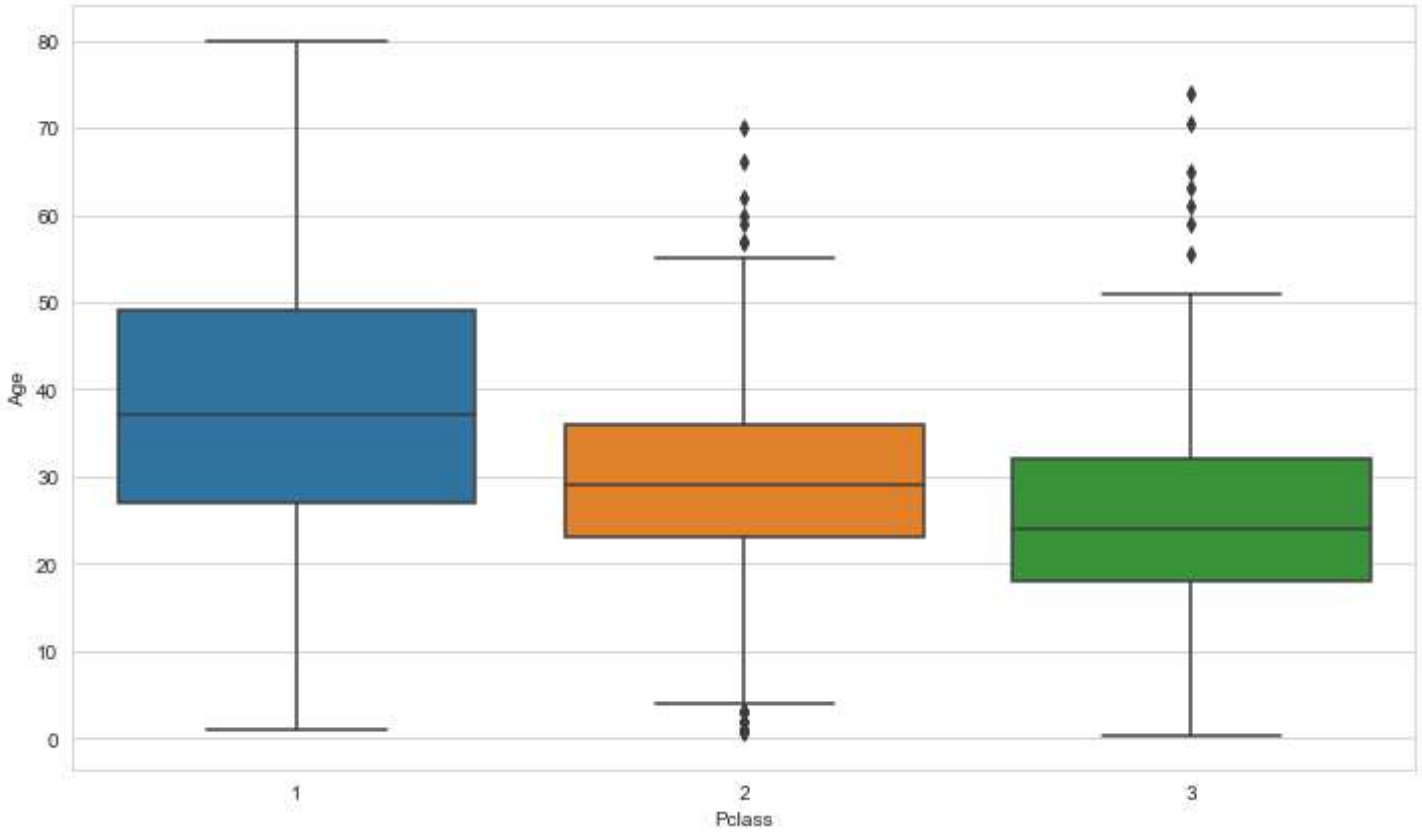


Data Cleaning

We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class. For example:

In [30]:
```
plt.figure(figsize=(12,7))
sns.boxplot(x='Pclass',y='Age',data=df)
```

Out[30]:
```
<AxesSubplot:xlabel='Pclass', ylabel='Age'>
```



We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

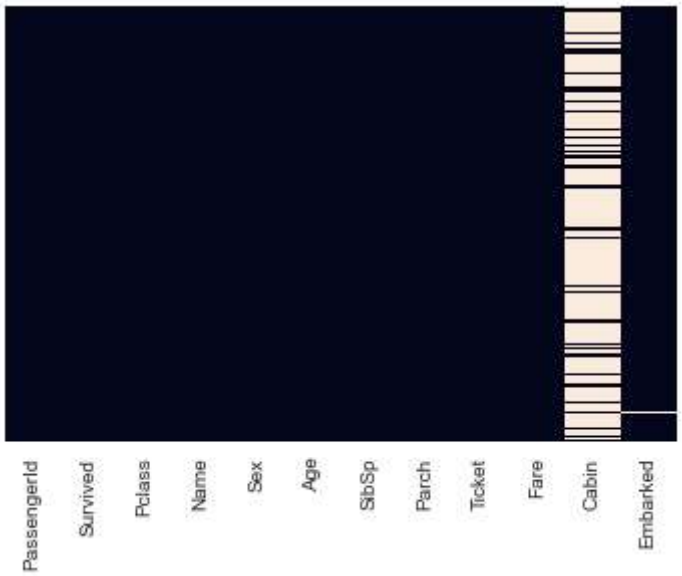
In [37]:
```
def impute_Age(cols):
    Age=cols[0]
    Pclass=cols[1]
```

```
    if pd.isnull(Age):
        if Pclass==1:
            return 37
        elif Pclass==2:
            return 29
        else:
            return 24
    else:
        return Age
#the return values 37,29,24 are the mean values taken from the boxplot given above
```

In [32]:
```
df['Age'] = df[['Age','Pclass']].apply(impute_Age,axis=1)
```

In [36]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

Out[36]: <AxesSubplot:>



In [38]:
```
df.drop('Cabin',axis=1,inplace=True)
```

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

Out[39]:

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

In [40]:
```
df.shape
```

Out[40]:
```
(891, 11)
```

In [44]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

Converting Categorical Features into Numerical

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

In [46]:
```
sex = pd.get_dummies(df['Sex'],drop_first=True)
embark = pd.get_dummies(df['Embarked'],drop_first=True)
```

In [50]:
```
df.drop(['Sex','Embarked'],axis=1,inplace=True)
```

In [51]:
```
df = pd.concat([df,sex,embark],axis=1)
```

In [52]:
```
df.head()
```

Out[52]:

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

Train and Test Data Split

In [53]:
```
from sklearn.model_selection import train_test_split
```

In [61]:
```
df_x = df.drop(['Survived','Name','Ticket'],axis=1)
y = df['Survived']
x = df_x
```

In [62]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=101)
```

Building Logistic Regression model

In [63]:
```
from sklearn.linear_model import LogisticRegression
```

In [64]:
```
log_reg = LogisticRegression()
```

In [65]:
```
log_reg.fit(x_train,y_train)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[65]:
```
LogisticRegression()
```

Making Prediction

In [67]:
```
predictions = log_reg.predict(x_test)
```

Evaluation metrics

In [68]:
```
from sklearn import metrics
```

In [69]:
```
print("mean squared error:",metrics.mean_squared_error(y_test,predictions))
print("root mean squared error:",metrics.r2_score(y_test,predictions))
```

```
mean squared error: 0.23134328358208955
root mean squared error: 0.053542948279790226
```

In [70]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.87      0.81       154
           1       0.78      0.63      0.70       114

    accuracy                           0.77       268
   macro avg       0.77      0.75      0.76       268
weighted avg       0.77      0.77      0.76       268
```

In [ ]: