titanic-data-analysis

May 11, 2023

[2]:
```python
# importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import math
```

[13]:
```python
import warnings
warnings.filterwarnings("ignore")
```

[4]:
```python
titanic = pd.read_csv("titanic.csv")
```

[5]:
```python
titanic.head()
```

[5]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[6]:
```python
# number of passengers travelling in the ship
print("no of passengers are --->>>",len(titanic))
```

```
no of passengers are --->>> 891
```

[7]:
```python
titanic.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
```

[9]:
```python
# check for duplicate data
titanic.duplicated().sum()   # no duplicate data
```

[9]:
```
0
```

[10]:
```python
# check for missing values
titanic.isnull().sum()
```

[10]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
```

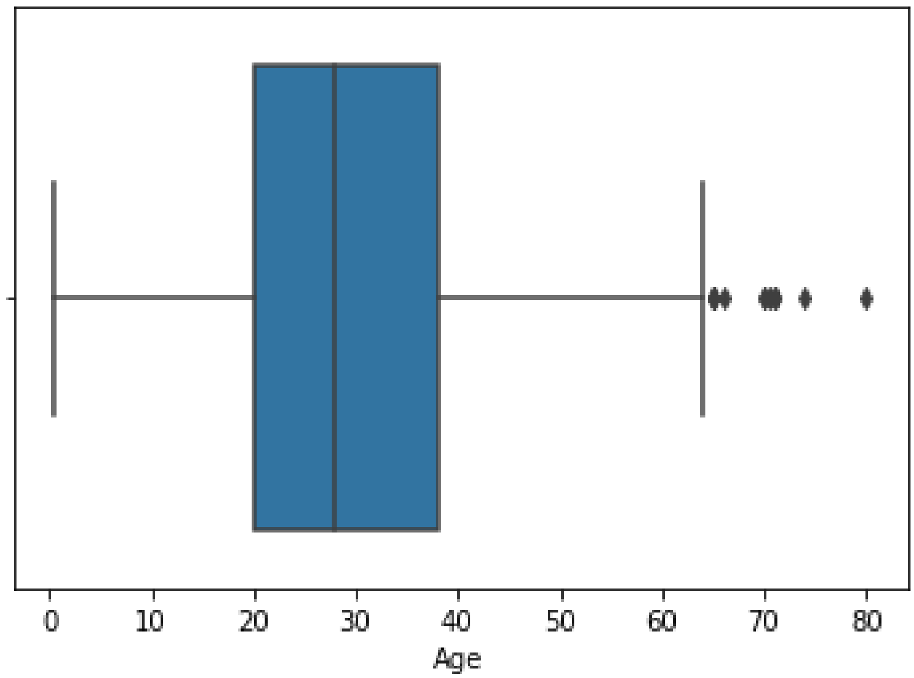
```
dtype: int64
```

[11]:
```python
# Age, Cabin and Embarked columns have missing values
# checking the % of missing data
titanic.isnull().mean()*100
```

[11]:
```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

[12]:
```python
# dropping the Cabin column because it has more than 75% of mv
titanic.drop("Cabin", axis=1, inplace=True)
```

[15]:
```python
# imputing missing values in age column
sns.boxplot(titanic.Age)
plt.show()
# the boxplot is showing more than 65 values are outliers but the age
# should be near 80 so it should be ok
```



[16]:
```
titanic.Age.mean()
```

[16]:
```
29.69911764705882
```

[17]:
```
# imputing the Age nan values with mean
titanic.Age = titanic.Age.replace({np.nan:30})
```

[18]:
```
titanic.Age.head()  # it is in float so converting to integer
```

[18]:
```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

[19]:
```
titanic.Age.isnull().sum()
```

[19]:
```
0
```

[22]:
```
titanic.Age = titanic.Age.astype(np.int64)
```

[23]:
```
titanic.Age.dtype
```

[23]:
```
dtype('int64')
```

[25]:
```
# Imputing the missing values of Embarked column
# the column is object so replacing nan values with mode
titanic.Embarked.value_counts()
```

[25]:
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

[26]:
```
titanic.Embarked = titanic.Embarked.replace({np.nan:"S"})
```

[27]:
```
titanic.isnull().sum()   # no missing values
```

[27]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

[28]:
```
numeric = []
categ = []
for i in titanic.columns:
    if titanic[i].dtype =="int64" or titanic[i].dtype =="float64":
        numeric.append(i)
    else:
        categ.append(i)
```

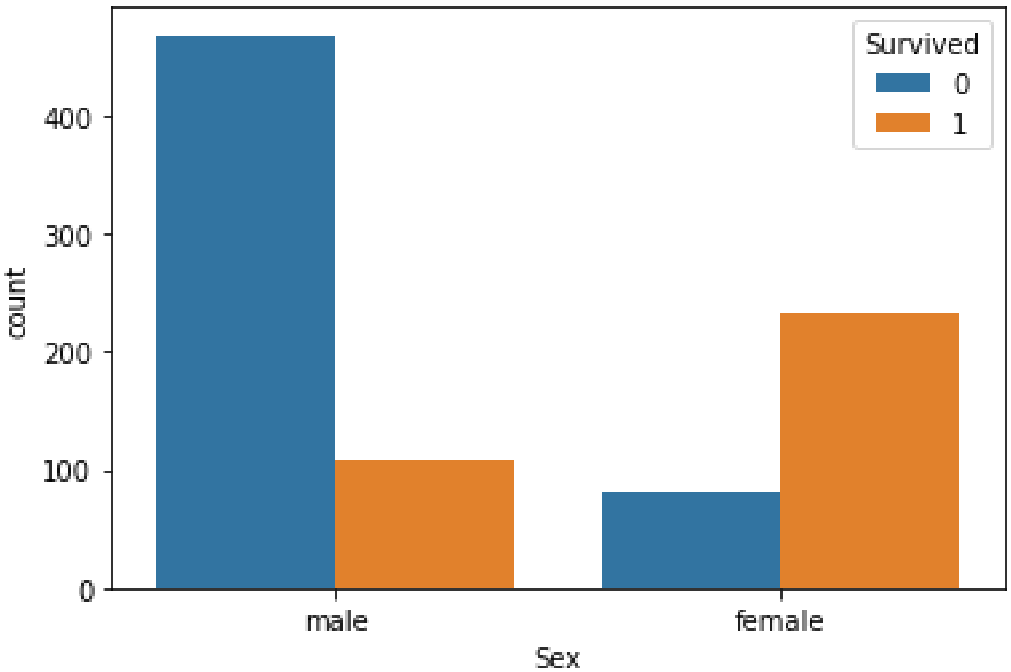
[29]:
```
numeric
```

[29]:
```
['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

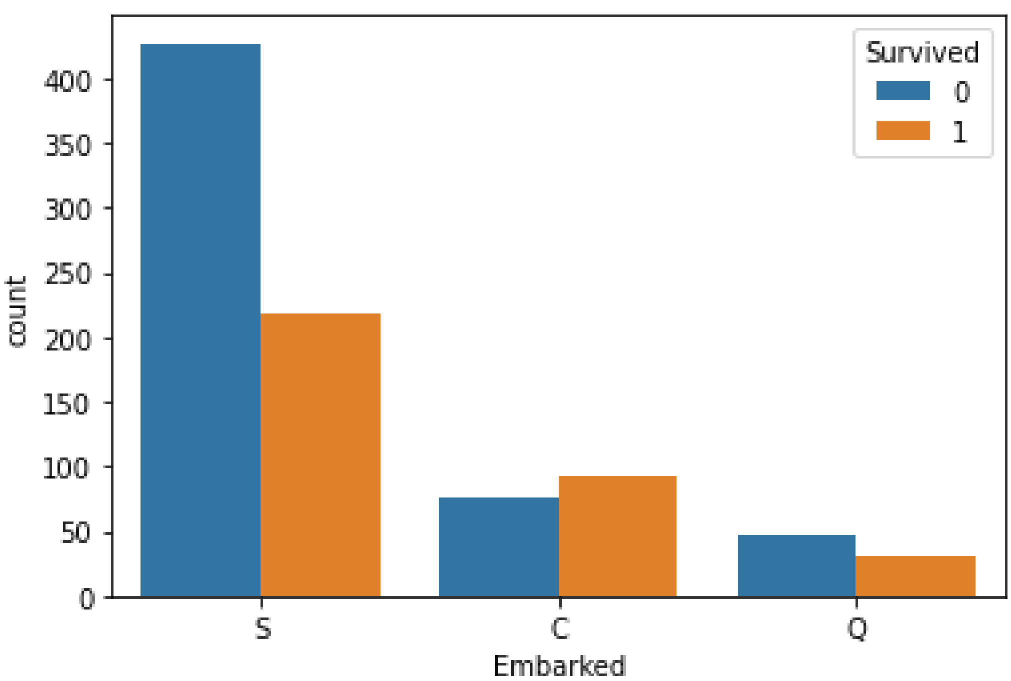
[30]:
```
categ
```

[30]:
```
['Name', 'Sex', 'Ticket', 'Embarked']
```

[39]:
```
sns.countplot(titanic.Sex, hue=titanic.Survived)
plt.show() # more females are survived compare to male
```

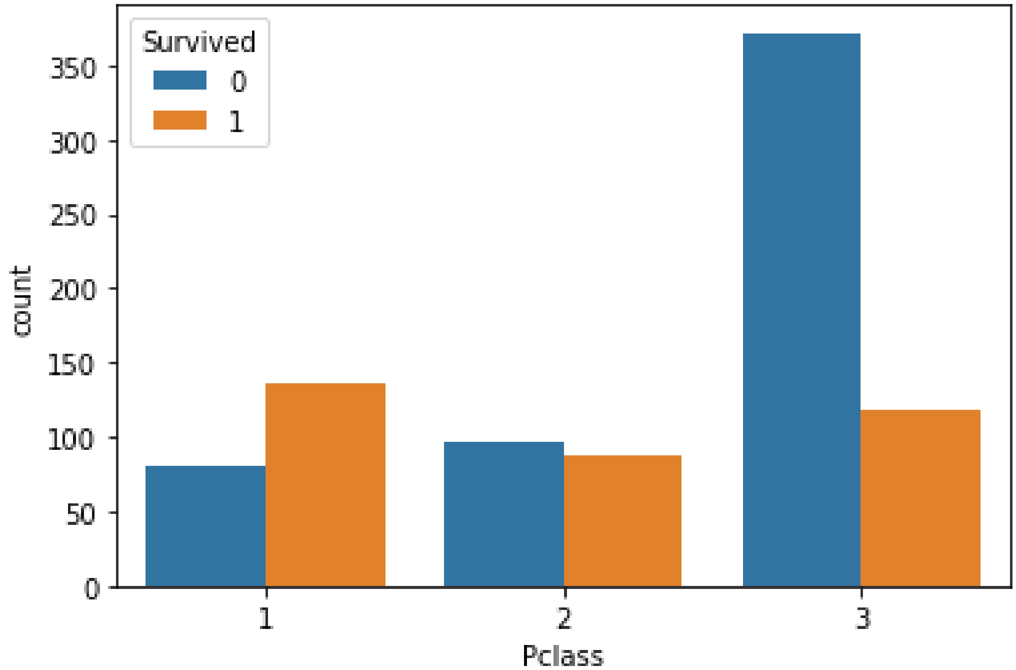


[40]:
```
sns.countplot(titanic.Embarked, hue=titanic.Survived)
plt.show()    # most of the passengers are ported from S
```



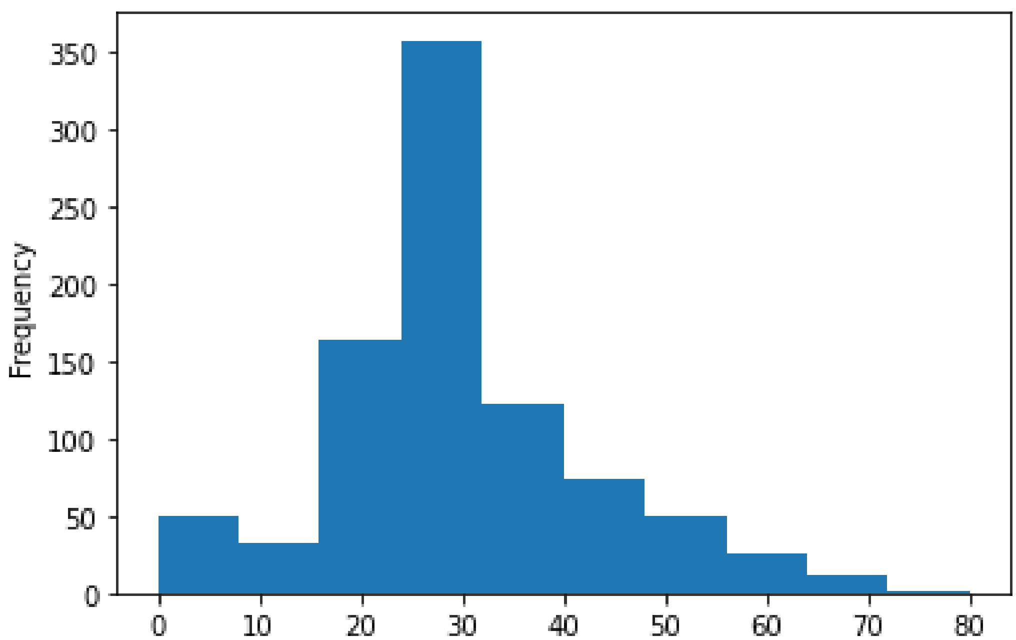
[41]:

```
sns.countplot(titanic.Pclass, hue=titanic.Survived)
plt.show()  # passenger of class 1 are survived more than other class
# passengers who are not survived are majorly from class 3
```

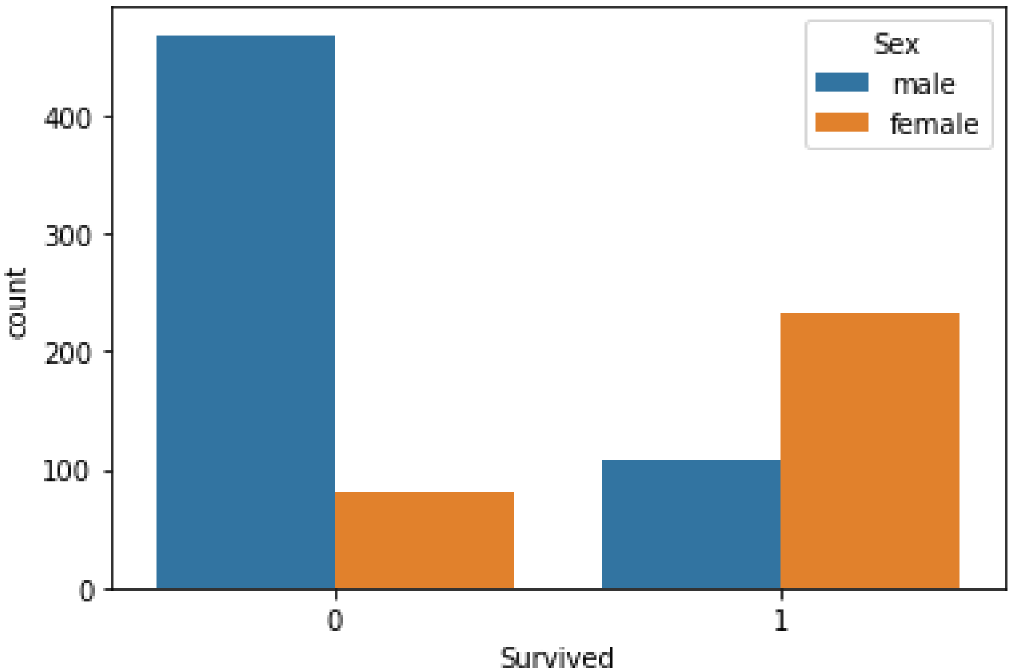


[43]:

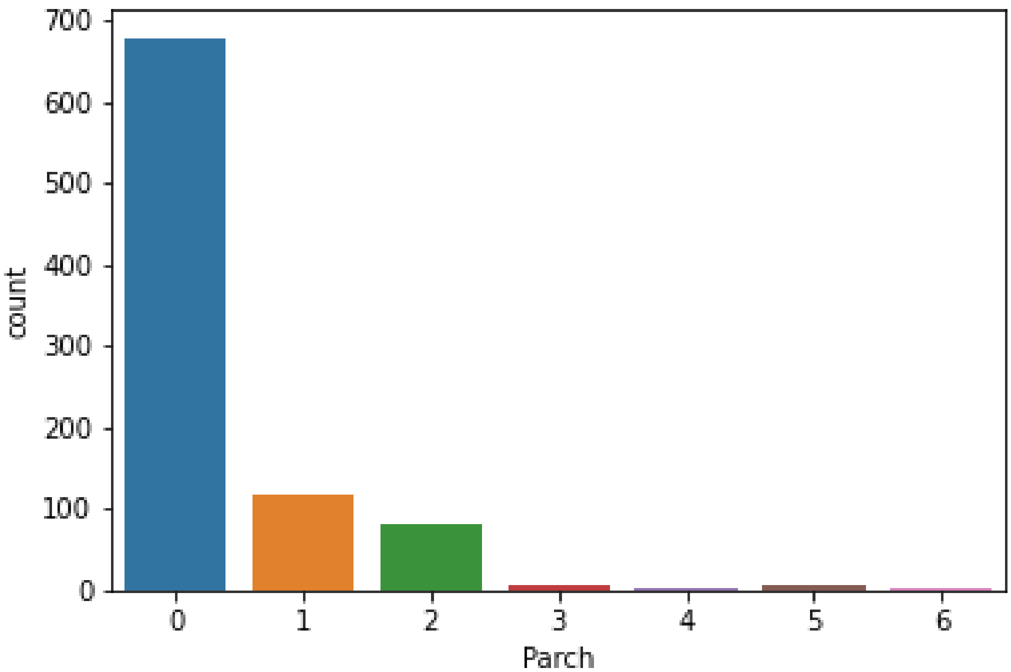
```
titanic.Age.plot.hist()
plt.show()  # most of the passengers are having age between 20 to 40
```



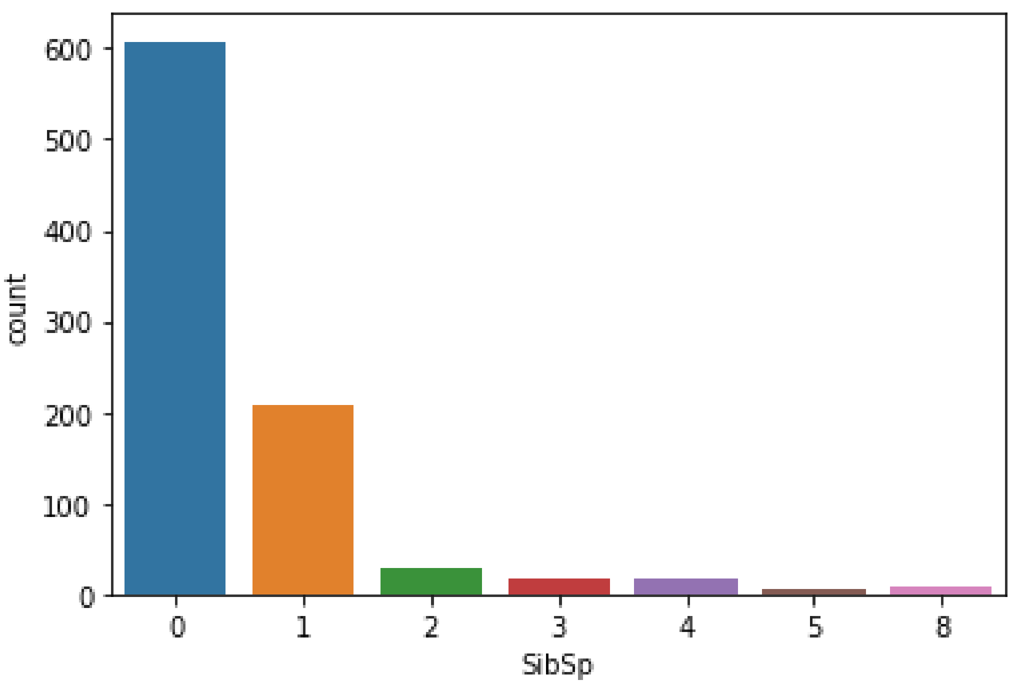
[45]:
```
sns.countplot(titanic.Survived, hue=titanic.Sex)
plt.show()  # out of 891 ,around 340 are survived
# in surving females get more weightage than males
```



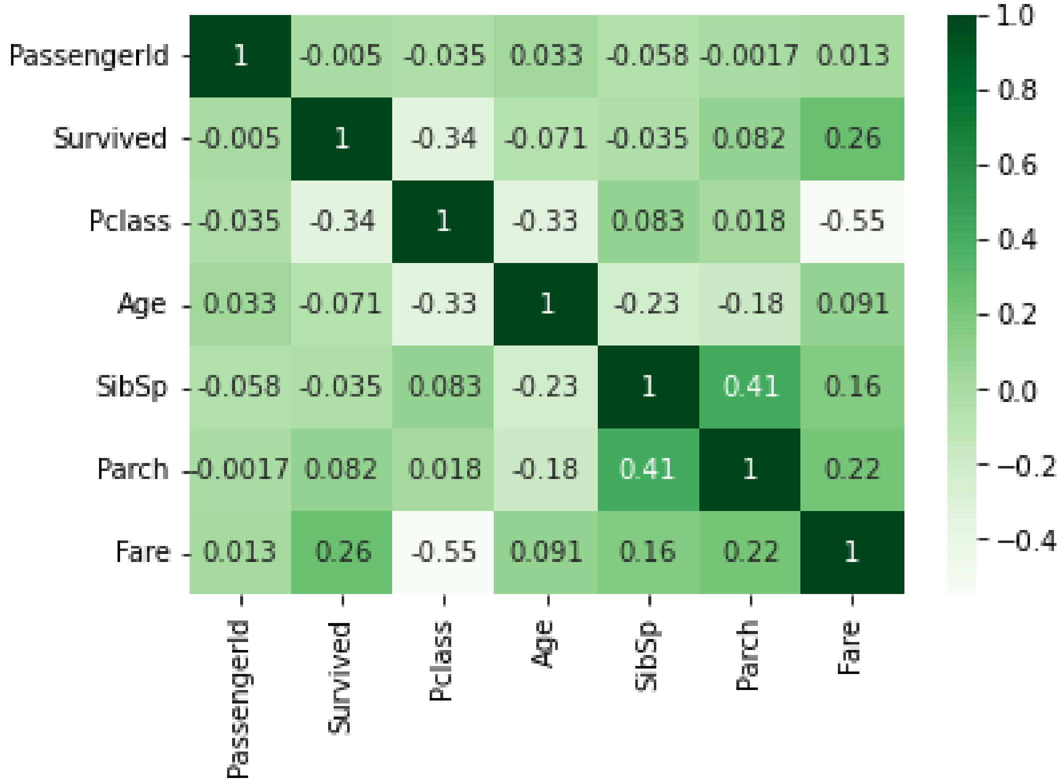
[46]:
```
sns.countplot(titanic.Parch)
plt.show()  # most of them are without any children or parents
```



[47]:
```
sns.countplot(titanic.SibSp)
plt.show()    # most of them are without any siblings or spouse
```



[49]:
```
sns.heatmap(titanic.corr(), annot=True, cmap="Greens")
plt.show()
```



[50]:
```
titanic.columns
```

[50]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked'],
      dtype='object')
```

[52]:
```
# create dummies
# column "Pclass" it has 3 values 1,2,3
pcl = pd.get_dummies(titanic.Pclass, drop_first=True)
pcl.head()
```

[52]:

	2	3
0	0	1
1	0	0
2	0	1
3	0	0
4	0	1

[53]:
```python
# sex column
# replacing values of males with 1 and female with 0
titanic.Sex = titanic.Sex.replace({"male":1, "female":0})
```

[54]:
```python
# Embarked column
embark = pd.get_dummies(titanic.Embarked, drop_first=True)
embark.head()
```

[54]:

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

[55]:
```python
titanic_new = pd.concat([titanic,pcl,embark], axis=1)
```

[56]:
```python
# dropping the useless columns
titanic_new.drop(['PassengerId','Pclass','Name','Ticket','Embarked'],axis=1,
                inplace=True)
```

[57]:
```python
titanic_new.head()
```

[57]:

	Survived	Sex	Age	SibSp	Parch	Fare	2	3	Q	S
0	0	1	22	1	0	7.2500	0	1	0	1
1	1	0	38	1	0	71.2833	0	0	0	0
2	1	0	26	0	0	7.9250	0	1	0	1
3	1	0	35	1	0	53.1000	0	0	0	1
4	0	1	35	0	0	8.0500	0	1	0	1

Train and Test data

[58]:
```python
X = titanic_new.drop("Survived",axis=1)
y = titanic_new.Survived
```

[61]:
```python
from sklearn.model_selection import train_test_split
```

[136]:
```python
x_train,x_test,y_train,y_test = train_test_split(X,y, train_size=0.75,␣
↪random_state=100)
```

[137]:
```python
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

[137]: ((668, 9), (223, 9), (668,), (223,))

[138]:
```python
from sklearn.linear_model import LogisticRegression
```

[139]:
```python
lr = LogisticRegression()
```

[140]:
```
lr.fit(x_train, y_train)
```

[140]:
```
LogisticRegression()
```

[142]:
```
y_pred = lr.predict(x_test)
```

[143]:
```
y_pred
```

[143]:
```
array([1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 1], dtype=int64)
```

[145]:
```
lr.predict_proba(x_test)
```

[145]:
```
array([[0.24498507, 0.75501493],
       [0.29740863, 0.70259137],
       [0.90555105, 0.09444895],
       [0.29199896, 0.70800104],
       [0.79719063, 0.20280937],
       [0.25138113, 0.74861887],
       [0.91273187, 0.08726813],
       [0.79429422, 0.20570578],
       [0.07201488, 0.92798512],
       [0.66023096, 0.33976904],
       [0.02991357, 0.97008643],
       [0.88977604, 0.11022396],
       [0.80787651, 0.19212349],
       [0.58620948, 0.41379052],
       [0.4745412 , 0.5254588 ],
       [0.84251599, 0.15748401],
       [0.09256795, 0.90743205],
       [0.66826481, 0.33173519],
       [0.60627472, 0.39372528],
       [0.88592897, 0.11407103],
       [0.91466345, 0.08533655],
       [0.05060341, 0.94939659],
       [0.87834135, 0.12165865],
       [0.3604996 , 0.6395004 ],
       [0.9082167 , 0.0917833 ],
```

```
[0.9146019 , 0.0853981 ],
[0.24124133, 0.75875867],
[0.90540386, 0.09459614],
[0.92794392, 0.07205608],
[0.85909046, 0.14090954],
[0.99116047, 0.00883953],
[0.1930689 , 0.8069311 ],
[0.60161681, 0.39838319],
[0.74004035, 0.25995965],
[0.07503033, 0.92496967],
[0.92528054, 0.07471946],
[0.10474279, 0.89525721],
[0.08034538, 0.91965462],
[0.25478045, 0.74521955],
[0.91471164, 0.08528836],
[0.41711692, 0.58288308],
[0.04651352, 0.95348648],
[0.84453918, 0.15546082],
[0.87182948, 0.12817052],
[0.99116047, 0.00883953],
[0.33230373, 0.66769627],
[0.8858622 , 0.1141378 ],
[0.04336588, 0.95663412],
[0.82028211, 0.17971789],
[0.14463011, 0.85536989],
[0.87011604, 0.12988396],
[0.72550251, 0.27449749],
[0.92322973, 0.07677027],
[0.9027845 , 0.0972155 ],
[0.91763461, 0.08236539],
[0.87100174, 0.12899826],
[0.92515657, 0.07484343],
[0.92166276, 0.07833724],
[0.82999979, 0.17000021],
[0.78005803, 0.21994197],
[0.78143661, 0.21856339],
[0.64673872, 0.35326128],
[0.88937431, 0.11062569],
[0.8999849 , 0.1000151 ],
[0.74004035, 0.25995965],
[0.35181033, 0.64818967],
[0.88589879, 0.11410121],
[0.23133112, 0.76866888],
[0.18598234, 0.81401766],
[0.93015661, 0.06984339],
[0.7735693 , 0.2264307 ],
[0.94104645, 0.05895355],
```

```
[0.9200851 , 0.0799149 ],
[0.38588116, 0.61411884],
[0.90894035, 0.09105965],
[0.79959657, 0.20040343],
[0.65013505, 0.34986495],
[0.87694862, 0.12305138],
[0.93198589, 0.06801411],
[0.87695094, 0.12304906],
[0.74249147, 0.25750853],
[0.81402474, 0.18597526],
[0.16223886, 0.83776114],
[0.11709219, 0.88290781],
[0.15218735, 0.84781265],
[0.89989021, 0.10010979],
[0.9111617 , 0.0888383 ],
[0.15638383, 0.84361617],
[0.70410623, 0.29589377],
[0.95787246, 0.04212754],
[0.89316678, 0.10683322],
[0.57323484, 0.42676516],
[0.91466345, 0.08533655],
[0.88945184, 0.11054816],
[0.05109101, 0.94890899],
[0.39831677, 0.60168323],
[0.26818811, 0.73181189],
[0.09717053, 0.90282947],
[0.9146019 , 0.0853981 ],
[0.34251262, 0.65748738],
[0.89997915, 0.10002085],
[0.89470124, 0.10529876],
[0.88179455, 0.11820545],
[0.08339893, 0.91660107],
[0.9200851 , 0.0799149 ],
[0.17809994, 0.82190006],
[0.92760136, 0.07239864],
[0.38588116, 0.61411884],
[0.32786604, 0.67213396],
[0.69598802, 0.30401198],
[0.93002365, 0.06997635],
[0.83632069, 0.16367931],
[0.47785329, 0.52214671],
[0.89316678, 0.10683322],
[0.39819454, 0.60180546],
[0.89294829, 0.10705171],
[0.9305139 , 0.0694861 ],
[0.95187078, 0.04812922],
[0.89997915, 0.10002085],
```

```
[0.43016626, 0.56983374],
[0.90588574, 0.09411426],
[0.9146019 , 0.0853981 ],
[0.11235038, 0.88764962],
[0.72776156, 0.27223844],
[0.97419364, 0.02580636],
[0.12804759, 0.87195241],
[0.93430825, 0.06569175],
[0.88254513, 0.11745487],
[0.04432996, 0.95567004],
[0.90592932, 0.09407068],
[0.45372052, 0.54627948],
[0.79575333, 0.20424667],
[0.60210556, 0.39789444],
[0.25378242, 0.74621758],
[0.8826708 , 0.1173292 ],
[0.68806419, 0.31193581],
[0.88265186, 0.11734814],
[0.70004488, 0.29995512],
[0.20716191, 0.79283809],
[0.88806253, 0.11193747],
[0.27558421, 0.72441579],
[0.35745578, 0.64254422],
[0.76086619, 0.23913381],
[0.76666556, 0.23333444],
[0.89997915, 0.10002085],
[0.85337984, 0.14662016],
[0.38572465, 0.61427535],
[0.19051619, 0.80948381],
[0.70391693, 0.29608307],
[0.19484729, 0.80515271],
[0.27212605, 0.72787395],
[0.8839621 , 0.1160379 ],
[0.74459357, 0.25540643],
[0.23518855, 0.76481145],
[0.81501916, 0.18498084],
[0.38572465, 0.61427535],
[0.90536516, 0.09463484],
[0.4106211 , 0.5893789 ],
[0.9242273 , 0.0757727 ],
[0.92791828, 0.07208172],
[0.3104241 , 0.6895759 ],
[0.40193776, 0.59806224],
[0.68895298, 0.31104702],
[0.91466345, 0.08533655],
[0.07147442, 0.92852558],
[0.87694862, 0.12305138],
```

```
[0.38572465, 0.61427535],
[0.6518783 , 0.3481217 ],
[0.74004035, 0.25995965],
[0.12833996, 0.87166004],
[0.65742258, 0.34257742],
[0.12726699, 0.87273301],
[0.59804896, 0.40195104],
[0.09822878, 0.90177122],
[0.1860191 , 0.8139809 ],
[0.31348936, 0.68651064],
[0.91906142, 0.08093858],
[0.94989269, 0.05010731],
[0.26444959, 0.73555041],
[0.88593971, 0.11406029],
[0.96157317, 0.03842683],
[0.89670547, 0.10329453],
[0.0781294 , 0.9218706 ],
[0.65254715, 0.34745285],
[0.91617925, 0.08382075],
[0.96329379, 0.03670621],
[0.80610433, 0.19389567],
[0.94661103, 0.05338897],
[0.44038964, 0.55961036],
[0.18662757, 0.81337243],
[0.34643517, 0.65356483],
[0.9174091 , 0.0825909 ],
[0.19775673, 0.80224327],
[0.89997915, 0.10002085],
[0.71882255, 0.28117745],
[0.94442549, 0.05557451],
[0.27787712, 0.72212288],
[0.92019239, 0.07980761],
[0.28919633, 0.71080367],
[0.50825781, 0.49174219],
[0.09986118, 0.90013882],
[0.91199107, 0.08800893],
[0.79035999, 0.20964001],
[0.66815148, 0.33184852],
[0.83504206, 0.16495794],
[0.95169134, 0.04830866],
[0.8895021 , 0.1104979 ],
[0.72377226, 0.27622774],
[0.70821257, 0.29178743],
[0.03599589, 0.96400411],
[0.70495827, 0.29504173],
[0.5906016 , 0.4093984 ],
[0.73428404, 0.26571596],
```

```
        [0.43514057, 0.56485943],
        [0.93660846, 0.06339154],
        [0.92241535, 0.07758465],
        [0.1338508 , 0.8661492 ],
        [0.10301272, 0.89698728],
        [0.95516284, 0.04483716],
        [0.81402474, 0.18597526],
        [0.87695094, 0.12304906],
        [0.38541183, 0.61458817],
        [0.18340836, 0.81659164]])
```

[146]:
```
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
 ↪f1_score
```

[147]:
```
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Recall: ", recall_score(y_test, y_pred))
print("Precision: ", precision_score(y_test, y_pred))
print("F1Score: ", f1_score(y_test, y_pred))
```

```
Accuracy:  0.7847533632286996
Recall:  0.6666666666666666
Precision:  0.8
F1Score:  0.7272727272727272
```

[148]:
```
# tuning probability cutoff
prob = pd.DataFrame()
```

[150]:
```
prob["y_actual"] = y_train
```

[152]:
```
prob['p(y=1|x)'] = lr.predict_proba(x_train)[:,1]
```

[153]:
```
prob.head()
```

[153]:

	y_actual	p(y=1|x)
225	0	0.111281
856	1	0.867430
620	0	0.108423
450	0	0.137570
423	0	0.481504

[155]:
```
cut = [float(x)/10 for x in range(0,11)]
cut
```

[155]:
```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
```

[156]:
```
for i in cut:
    prob[i] = prob['p(y=1|x)'].map(lambda x: 1 if x>i else 0)
```

[157]:
```
prob.head()
```

[157]:

	y_actual	p(y=1|x)	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
225	0	0.111281	1	1	0	0	0	0	0	0	0	0	0
856	1	0.867430	1	1	1	1	1	1	1	1	1	0	0
620	0	0.108423	1	1	0	0	0	0	0	0	0	0	0
450	0	0.137570	1	1	0	0	0	0	0	0	0	0	0
423	0	0.481504	1	1	1	1	1	0	0	0	0	0	0

[158]:
```
cutoff_df = pd.DataFrame(columns = ['prob', 'accuracy', 'recall', 'precision'])

for i in cut:
    a = accuracy_score(prob['y_actual'], prob[i])
    r = recall_score(prob['y_actual'], prob[i])
    p = precision_score(prob['y_actual'], prob[i])

    cutoff_df.loc[i] = [i,a,r,p]
```

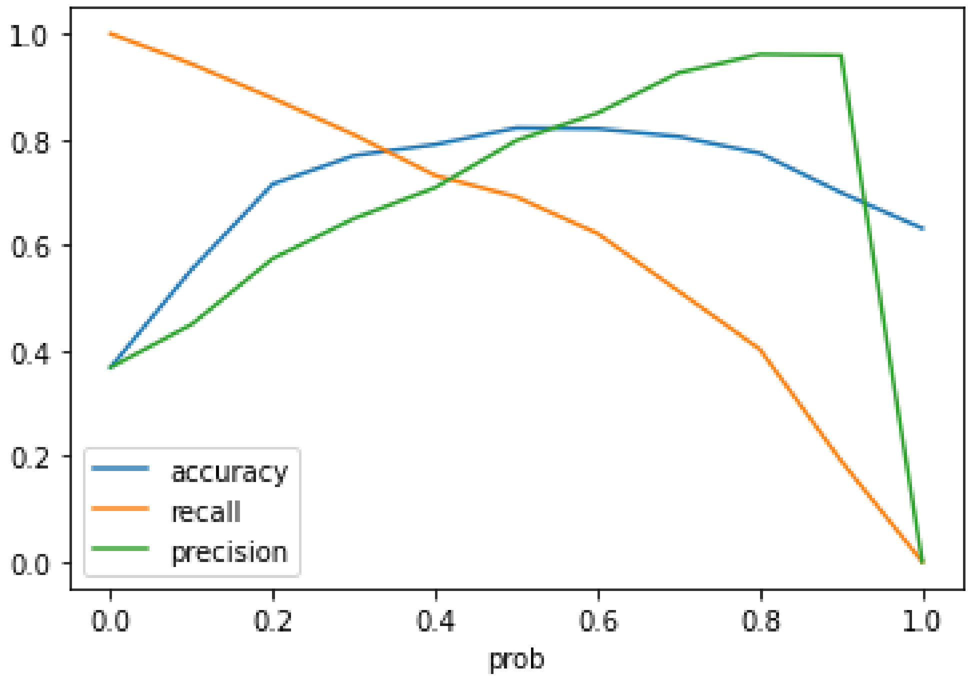
[159]:
```
cutoff_df
```

[159]:

	prob	accuracy	recall	precision
0.0	0.0	0.368263	1.000000	0.368263
0.1	0.1	0.553892	0.943089	0.449612
0.2	0.2	0.715569	0.878049	0.574468
0.3	0.3	0.769461	0.808943	0.650327
0.4	0.4	0.790419	0.731707	0.708661
0.5	0.5	0.821856	0.691057	0.798122
0.6	0.6	0.820359	0.621951	0.850000
0.7	0.7	0.805389	0.512195	0.926471
0.8	0.8	0.773952	0.402439	0.961165
0.9	0.9	0.699102	0.191057	0.959184
1.0	1.0	0.631737	0.000000	0.000000

[160]:
```
cutoff_df.plot.line(x = 'prob', y = ['accuracy', 'recall', 'precision'])
```

[160]: <AxesSubplot:xlabel='prob'>



[161]:
```
y_test_pred_prob = lr.predict_proba(x_test)[:,1]
```

[168]:
```
y_test_01 = list(map(lambda x:1 if x>0.45 else 0, y_test_pred_prob))
```

[169]:
```
print("Accuracy: ", accuracy_score(y_test, y_test_01))
print("Recall: ", recall_score(y_test, y_test_01))
print("Precision: ", precision_score(y_test, y_test_01))
print("F1Score: ", f1_score(y_test, y_test_01))
```

```
Accuracy:  0.7892376681614349
Recall:  0.6770833333333334
Precision:  0.8024691358024691
F1Score:  0.7344632768361582
```