TITANIC SURVIVAL PREDICTION

TECH-A-INTERN intership project (DataScience) submitted by Tels Mariya Thomas (October 2023)

Titanic ship sink happened in the year of 1912. In this project iam trying to analyse and predict the survival of passengers of Titanic through models.

https://www.kaggle.com/datasets/yasserh/titanic-dataset/data

In [163…

```
# import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

LOAD AND READ THE DATA

In [2]:

```
df=pd.read_csv(r"D:\datasets kaggle\Titanic-Dataset.csv")
df.head(5)
```

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

detials of features:

```
PassengerId: Passenger ID
Survived: Weather Survived or not: 0 = No, 1 = Yes
Pclass - Ticket class: 1 = 1st, 2 = 2nd, 3 = 3rd (1st ~ Upper; 2nd ~ Middle; 3rd
~ Lower)
```

```
Name:Name of the Passenger
Sex: Gender
Age:Age in Years
SibSp: No. of siblings / spouses aboard
    the Titanic  Sibling: Brother, Sister, Stepbrother, or Stepsister of
Passenger Aboard Titanic
    Spouse: Husband or Wife of Passenger Aboard Titanic (Mistresses and Fiances
Ignored)

Parch:No. of parents / children aboard the Titanic
    Parent: Mother or Father of Passenger Aboard Titanic
    Child: Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Titanic
    Some children travelled only with a nanny, therefore parch=0 for them.

Ticket:Ticket number
Fare:Passenger fare
cabin:cabin
embarked Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
```

In [3]:

```
#taking inforamation about the data
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 [4]:

```
#description of the data
df.describe()
```

Out[4]:

	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

age has missing values p class, Sibsp, parch, fare are skewed.

In [3]:
```
df.describe(include=object)
```

Out[3]:

	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

cabbin has missing values embarked has missing values

DATA PREPROCESSING

***Dropping the passenger id columns after extracting the title of the person and name as all are unique.**

In [4]:
```
# checking for titles in name column
df['Name'].str.split(" ", expand=True)[1].unique()
```

Out[4]:
```
array(['Mr.', 'Mrs.', 'Miss.', 'Master.', 'Planke,', 'Don.', 'Rev.',
       'Billiard,', 'der', 'Walle,', 'Dr.', 'Pelsmaeker,', 'Mulder,', 'y',
       'Steen,', 'Carlo,', 'Mme.', 'Impe,', 'Ms.', 'Major.', 'Gordon,',
       'Messemaeker,', 'Mlle.', 'Col.', 'Capt.', 'Velde,', 'the',
       'Shawah,', 'Jonkheer.', 'Melkebeke,', 'Cruyssen,'], dtype=object)
```

```
    1."Mlle." is an abbreviation for "Mademoiselle," a French courtesy title
used to address young unmarried women or girls.
    2.The title "Countess" is a noble title in various European countries and is
used to address the wife or widow of a Count, an Earl, or an equivalent noble
rank.
    3.In Dutch and Belgian nobility, "Jonkheer" (pronounced yon-kuhr) is an
honorific that translates to "young lord" or "young gentleman" in English. I
    4."Mme." is an abbreviation for "Madame," a French courtesy title that is
the equivalent of "Mrs." or "Ma'am" in English.
    5.Miss: "Miss" is used as a title before the name of an unmarried woman or
girl.
    6.Master: "Master" is used as a title before the name of a young boy. It is
used for boys who have not yet reached adulthood.
    7.Mr.: Used before a man's name to address him as "Mister.
    8.Mrs.: Used before a married woman's name.
```

In [7]:
```
import re
```

In [8]:
```
# getting the title pattern extracted from the name column
title_pattern = r'(Mr\.|Mrs\.|Miss\.|Master\.|Don\.|Rev\.|Dr\.|Ms\.|Major\.|Col\.|Capt\.

# Extract titles using regular expression and create a new 'Title' column
df['Title'] = df['Name'].str.extract(title_pattern, expand=False)
```

In [9]:
```
# checking for null values in title column
df[df['Title'].isnull()]
```

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title
759	760	1	1	Rothes, the Countess. of (Lucy Noel Martha Dye...	female	33.0	0	0	110152	86.5	B77	S	NaN

In [10]:

```
# the one column has null value bcz the title falls in third column after split. so fil
df['Title']=df['Title'].fillna('Countess.')
```

In [11]:

```
# dropping passenger id and name of the passenger
df1=df.drop(['PassengerId','Name'], axis=1)
```

In [ ]:

```

```

*checking for missing values

In [12]:

```
# checking for missing values in the columns
((df1.isnull().sum())/df1.shape[0])*100
```

Out[12]:

```
Survived      0.000000
Pclass        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
Title         0.000000
dtype: float64
```

There are 19 percentage missing values in the Age column. and 77 percentage missing values in the Cabin column.

* treating Age column based on the title

In [13]:

```
# checking for unique values in age
df1['Age'].unique()
```

Out[13]:

```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

In [14]:

```
# taking average of age according to title as each title points out to a particular grou
df1.groupby('Title')['Age'].median()
```

Out[14]:
```
Title
 Lady.        48.0
Capt.        70.0
Col.         58.0
Countess.    33.0
Don.         40.0
Dr.          46.5
Jonkheer.    38.0
Major.       48.5
Master.       3.5
Miss.        21.0
Mlle.        24.0
Mme.         24.0
Mr.          30.0
Mrs.         35.0
Ms.          28.0
Rev.         46.5
Sir.         49.0
Name: Age, dtype: float64
```

In [15]:
```
# imputing the median age of each titke to the missing values on age column
df1['Age']=df1['Age'].fillna(df1.groupby('Title')['Age'].transform('median'))
```

In [ ]:

*Treating Cabin column

In [19]:
```
# checking for columns with cabin values.
# Cabin may not be available to all passengers.
df1[~(df1['Cabin'].isnull())]
```

Out[19]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title
1	1	1	female	38.0	1	0	PC 17599	71.2833	C85	C	Mrs.
3	1	1	female	35.0	1	0	113803	53.1000	C123	S	Mrs.
6	0	1	male	54.0	0	0	17463	51.8625	E46	S	Mr.
10	1	3	female	4.0	1	1	PP 9549	16.7000	G6	S	Miss.
11	1	1	female	58.0	0	0	113783	26.5500	C103	S	Miss.
...	...	...	...	...	...	...	...	...	...	...	...
871	1	1	female	47.0	1	1	11751	52.5542	D35	S	Mrs.
872	0	1	male	33.0	0	0	695	5.0000	B51 B53 B55	S	Mr.
879	1	1	female	56.0	0	1	11767	83.1583	C50	C	Mrs.
887	1	1	female	19.0	0	0	112053	30.0000	B42	S	Miss.
889	1	1	male	26.0	0	0	111369	30.0000	C148	C	Mr.

204 rows × 11 columns

In [17]:
```
# cabin has an alphabet which represent the deck and a number followed by which is the 
# slicing the cabin and storing the deck as a new column deck.
df1['Deck']=df1['Cabin'].str.slice(0,1)
```

In [ ]:

In [18]:
```
# filling the null values in the cabin column
df1['Cabin']=df1['Cabin'].fillna('not available')
```

In [19]:
```
# filling null values in deck with not available
df1['Deck']=df1['Deck'].fillna('not available')
```

In [ ]:
```
```

In [20]:
```
# dropping cabin column
df2=df1.drop('Cabin', axis=1)
```

*** creating a new column named all possible decks based on passenger class.**

In [ ]:
```
A Deck ---- exclusively for First Class
B Deck ---- First Class cabins
C Deck ---- First Class cabins
D Deck ---- First, Second and Third Class passengers had cabins
E Deck ---- passenger accommodation for all three classes
F Deck ---- Second and Third Class passengers
G Deck ---- orlop (partial) decks
```

https://en.wikipedia.org/wiki/Titanic#Background The boat deck, on which the lifeboats were housed. It was from here during the early hours of 15 April 1912 that Titanic's lifeboats were lowered into the North Atlantic. The bridge and wheelhouse were at the forward end, in front of the captain's and officers' quarters. The bridge stood 8 feet (2.4 m) above the deck, extending out to either side so that the ship could be controlled while docking. The wheelhouse stood within the bridge. The entrance to the First Class Grand Staircase and gymnasium were located midships along with the raised roof of the First Class lounge, while at the rear of the deck were the roof of the First Class smoke room and the relatively modest Second Class entrance. The wood-covered deck was divided into four segregated promenades: for officers, First Class passengers, engineers, and Second Class passengers respectively. Lifeboats lined the side of the deck except in the First Class area, where there was a gap so that the view would not be spoiled.[18][19]

```
   A Deck, also called the promenade deck, extended along the entire 546 feet
(166 m) length of the superstructure. It was reserved exclusively for First
Class passengers and contained First Class cabins, the First Class lounge, smoke
room, reading and writing rooms, and Palm Court.[18]

  B Deck, the bridge deck, was the top weight-bearing deck and the uppermost
level of the hull. More First Class passenger accommodations were located here
with six palatial staterooms (cabins) featuring their own private promenades. On
Titanic, the à la carte restaurant and the Café Parisien provided luxury dining
facilities to First Class passengers. Both were run by subcontracted chefs and
their staff; all were lost in the disaster. The Second Class smoking room and
entrance hall were both located on this deck. The raised forecastle of the ship
was forward of the bridge deck, accommodating Number 1 hatch (the main hatch
through to the cargo holds), numerous pieces of machinery and the anchor
housings.[c] Aft of the bridge deck was the raised poop deck, 106 feet (32 m)
long, used as a promenade by Third Class passengers. It was where many of
Titanic's passengers and crew made their last stand as the ship sank. The
forecastle and poop deck were separated from the bridge deck by well decks.[20]
[21]
```

C Deck, the shelter deck, was the highest deck to run uninterrupted from stem to stern. It included both well decks; the aft one served as part of the Third Class promenade. Crew cabins were housed below the forecastle and Third Class public rooms were housed below the poop deck. In between were the majority of First Class cabins and the Second Class library.[20][22]

D Deck, the saloon deck, was dominated by three large public rooms—the First Class reception room, the First Class dining saloon and the Second Class dining saloon. An open space was provided for Third Class passengers. First, Second and Third Class passengers had cabins on this deck, with berths for firemen located in the bow. It was the highest level reached by the ship's watertight bulkheads (though only by eight of the fifteen bulkheads).[20][23]

E Deck, the upper deck, was predominantly used for passenger accommodation for all three classes plus berths for cooks, seamen, stewards and trimmers. Along its length ran a long passageway nicknamed 'Scotland Road', in reference to a famous street in Liverpool. Scotland Road was used by Third Class passengers and crew members.[20][24]

F Deck, the middle deck, was the last complete deck, and mainly accommodated Second and Third Class passengers and several departments of the crew. The Third Class dining saloon was located here, as were the swimming pool, Turkish bath and kennels.[20][24][25]

G Deck, the lower deck, was the lowest complete deck that carried passengers, and had the lowest portholes, just above the waterline. The squash court was located here along with the travelling post office where letters and parcels were sorted ready for delivery when the ship docked. Food was also stored here. The deck was interrupted at several points by orlop (partial) decks over the boiler, engine and turbine rooms.[20][26]
The orlop decks, and the tank top below that, were on the lowest level of the ship, below the waterline. The orlop decks were used as cargo spaces, while the tank top—the inner bottom of the ship's hull—provided the platform on which the ship's boilers, engines, turbines and electrical generators were housed. This area of the ship was occupied by the engine and boiler rooms, areas which passengers would have been prohibited from seeing. They were connected with higher levels of the ship by flights of stairs; twin spiral stairways near the bow provided access up to D Deck.[20][26]

In [22]:

```
# creating the new column called all possible decks.

conditions = [
    (df2['Pclass'] == 1) & (df2['Deck'] == 'not available'),
    ((df2['Deck'] == 'A') | (df2['Deck'] == 'B') | (df2['Deck'] == 'C') | (df2['Deck'] =
    (df2['Pclass'] == 2) & (df2['Deck'] == 'not available'),
    ((df2['Deck'] == 'D') | (df2['Deck'] == 'E') | (df2['Deck'] == 'F')),
    (df2['Pclass'] == 3) & (df2['Deck'] == 'not available'),
    ((df2['Deck'] == 'D') | (df2['Deck'] == 'E') | (df2['Deck'] == 'F') | (df2['Deck'] =

choices = ['A/B/C/D', 'A/B/C/D', 'D/E/F', 'D/E/F', 'D/E/F/G', 'D/E/F/G']

df2['all possible Decks'] = np.select(conditions, choices, default=df2['Deck'])
```

In [ ]:

*getting a new column with no of persons in that specified ticket and fare per person

In [24]:

```
# getting a new column with no of persons in that specified ticket
value=df2['Ticket'].value_counts()
df2['persons_in_ticket']=df2['Ticket'].map(value)
```

In [25]:

```
value
```

Out[25]:

```
347082      7
CA. 2343    7
1601        7
3101295     6
CA 2144     6
           ..
9234        1
19988       1
2693        1
PC 17612    1
370376      1
Name: Ticket, Length: 681, dtype: int64
```

In [26]:

```
# creating fare per person using persond per ticket
df2['fare per person']=df2['Fare']/df2['persons_in_ticket']
```

In [28]:

```
# checking unique values in the column
df2['persons_in_ticket'].unique()
```

Out[28]:

```
array([1, 2, 4, 3, 7, 5, 6], dtype=int64)
```

*create a travel category column

In [ ]:

```
# creating a new column called traveler actegory -- big family(5), small family(5), nuc
```

In [31]:

```
# creating a new user defined function to apply on persond in ticket
def travel(x):
    if x==1:
        return 'solo'
    elif x==2:
        return 'bi'
    elif (x==3) | (x==4):
        return 'small family'
    elif (x==5) | (x==6) |(x==7):
        return 'large family'


df2['travelercategory']=df2['persons_in_ticket'].apply(travel)
```

In [ ]:

In [ ]:

* CREATING A NEW AGE GROUP COLUMN

In [36]:

```
## Age group columns
```

In [33]:

```
df2['age group']=df1['Age'].apply(lambda x: 'infant' if 0<x<=2 else 'child' if 2<x<=16 \
                                  else 'youngadults' if 16<x<=30 else \
                                 'middleaged' if 30<x<=45 else 'oldadults' if 45<x<=60
                                  else 'seniorcitizens' if x>60  else 'notavailable' )
```

In [ ]:

VISUALIZATION OF THE DATA

In [35]:

```
df2_num=df2.select_dtypes(exclude=object)
df2_cat=df2.select_dtypes(include=object)
```

In [36]:

```
df2_num
```

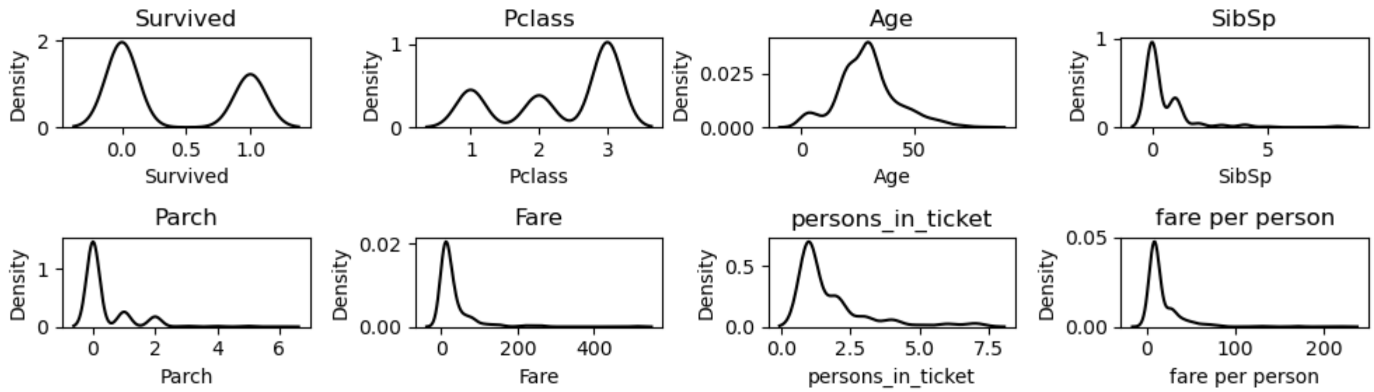
Out[36]:

	Survived	Pclass	Age	SibSp	Parch	Fare	persons_in_ticket	fare per person
0	0	3	22.0	1	0	7.2500	1	7.2500
1	1	1	38.0	1	0	71.2833	1	71.2833
2	1	3	26.0	0	0	7.9250	1	7.9250
3	1	1	35.0	1	0	53.1000	2	26.5500
4	0	3	35.0	0	0	8.0500	1	8.0500
...	...	...	...	...	...	...	...	...
886	0	2	27.0	0	0	13.0000	1	13.0000
887	1	1	19.0	0	0	30.0000	1	30.0000
888	0	3	21.0	1	2	23.4500	2	11.7250
889	1	1	26.0	0	0	30.0000	1	30.0000
890	0	3	32.0	0	0	7.7500	1	7.7500

891 rows × 8 columns

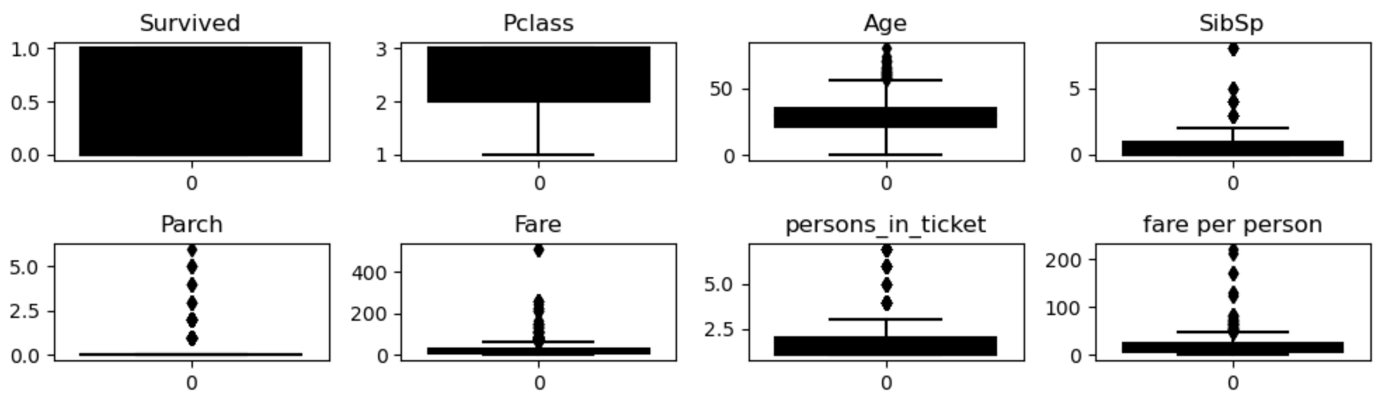
In [37]:

```
plt.figure(figsize=(10,3))
re=1
for i in df2_num.columns:
    plt.subplot(2,4,re)
    sns.kdeplot(df2[i], color='black')
    plt.title(i)
    re+=1

plt.tight_layout()
plt.show()
```



In [38]:

```
plt.figure(figsize=(10,3))
re=1
for i in df2_num.columns:
    plt.subplot(2,4,re)
    sns.boxplot(df2[i], color='black')
    plt.title(i)
    re+=1

plt.tight_layout()
plt.show()
```

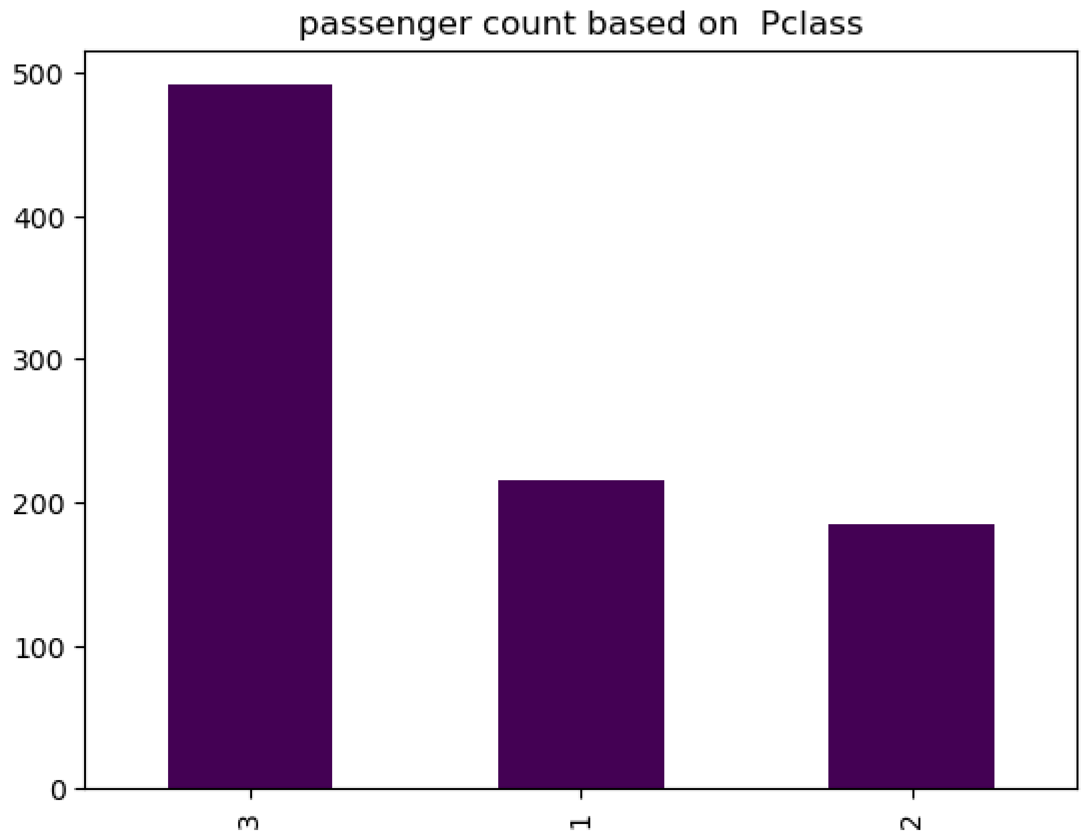


In [ ]:

```
The age, fare, persons per ticket, fare per person all has outliers.
```

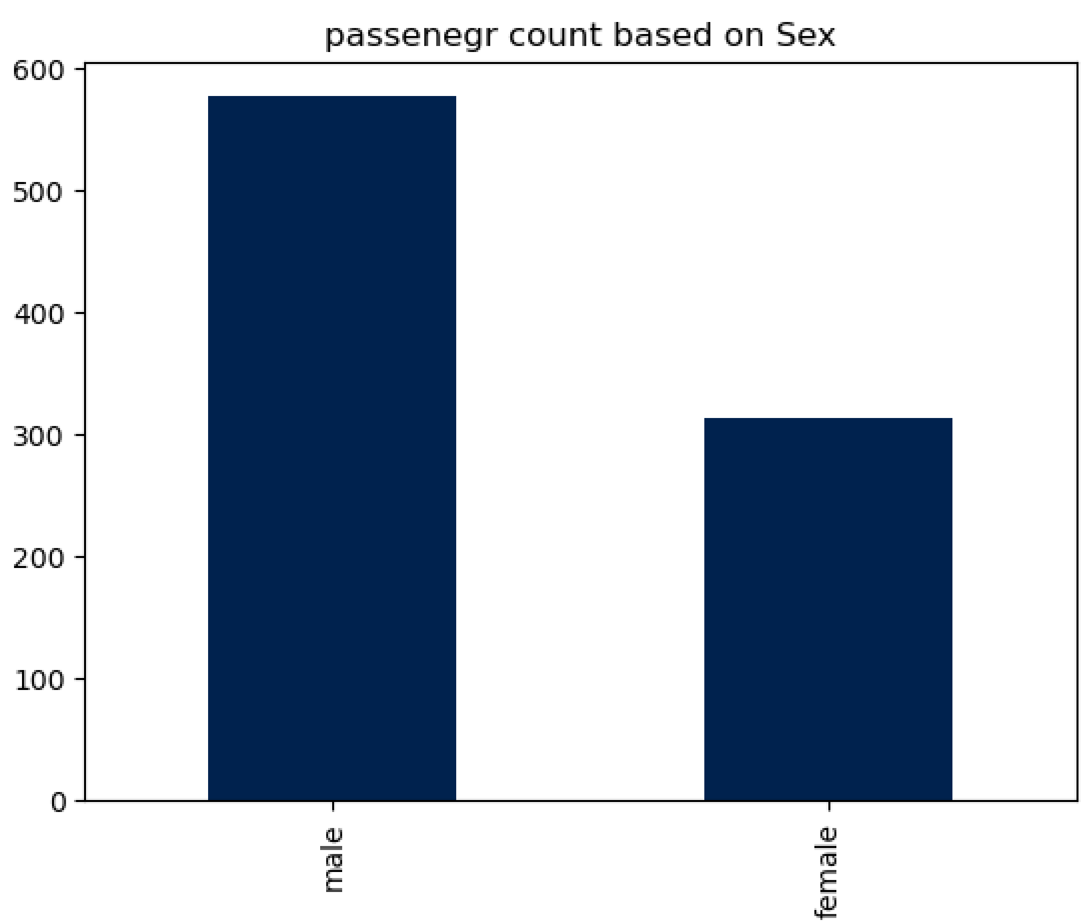
In [79]:

```
df2['Pclass'].value_counts().plot(kind='bar', cmap='viridis')
plt.title('passenger count based on  Pclass')
plt.show()
```



third class has the highest number of passengers compare to other 2 classes

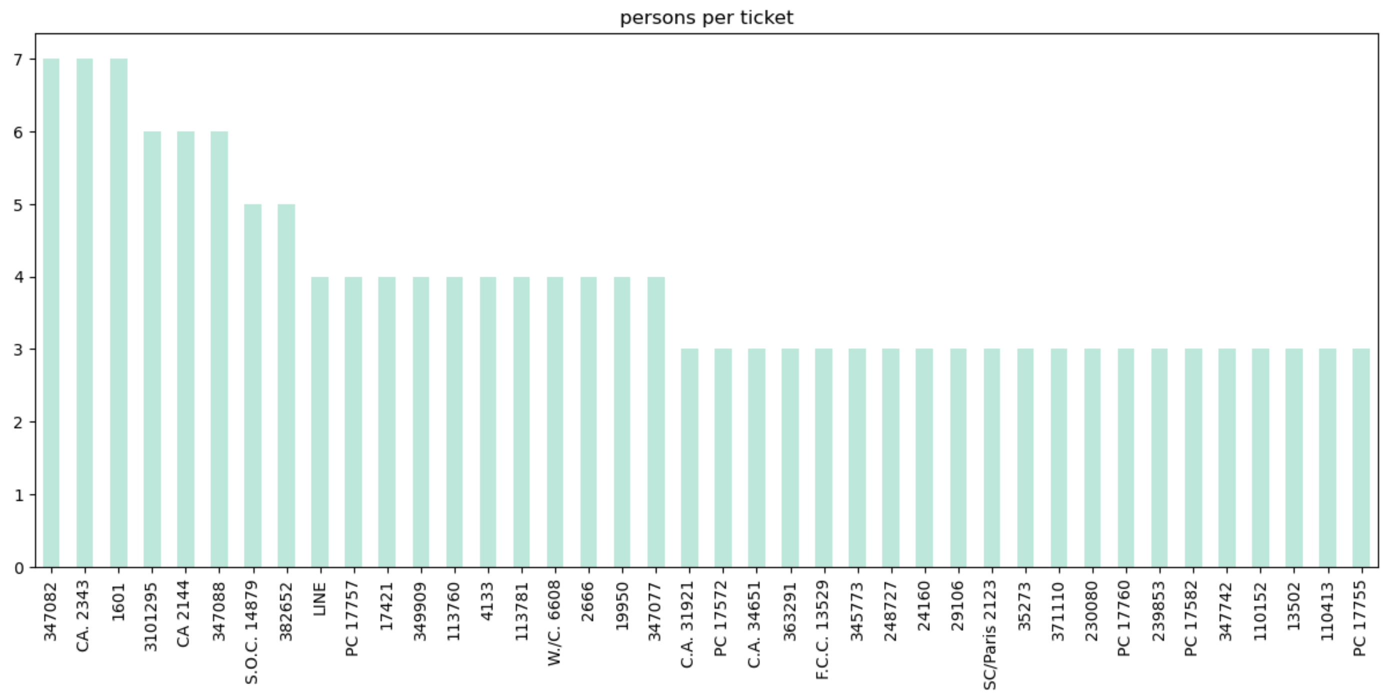
In [82]:
```
df2['Sex'].value_counts().plot(kind='bar', cmap='cividis')
plt.title('passenegr count based on Sex')
plt.show()
```



The number of male passengers is high compared to the female passengers.

In [86]:

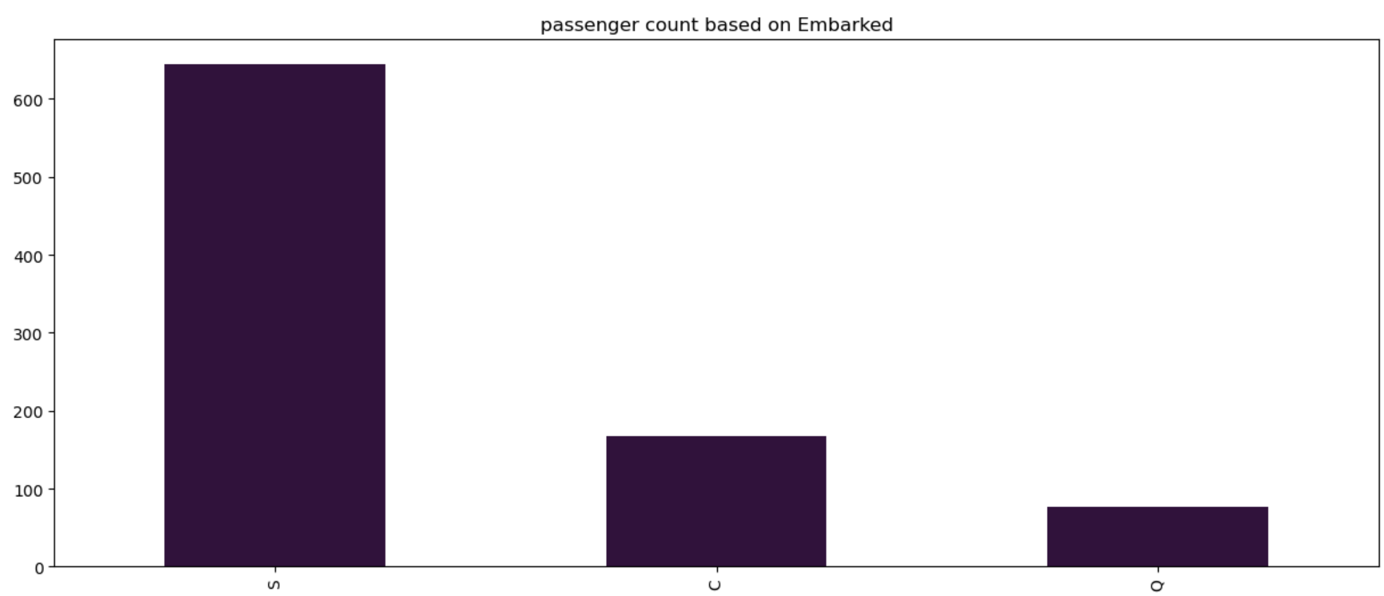
```
plt.figure(figsize=(15,6))
a=df2['Ticket'].value_counts()
a[a>2].plot(kind='bar', cmap='icefire')
plt.title('persons per ticket')
plt.show()
```



different ticket has different counts, highest being 7 and lowest being 1.

In [87]:

```
plt.figure(figsize=(15,6))
a=df2['Embarked'].value_counts()
a[a>2].plot(kind='bar', cmap='turbo')
plt.title('passenger count based on Embarked')
plt.show()
```

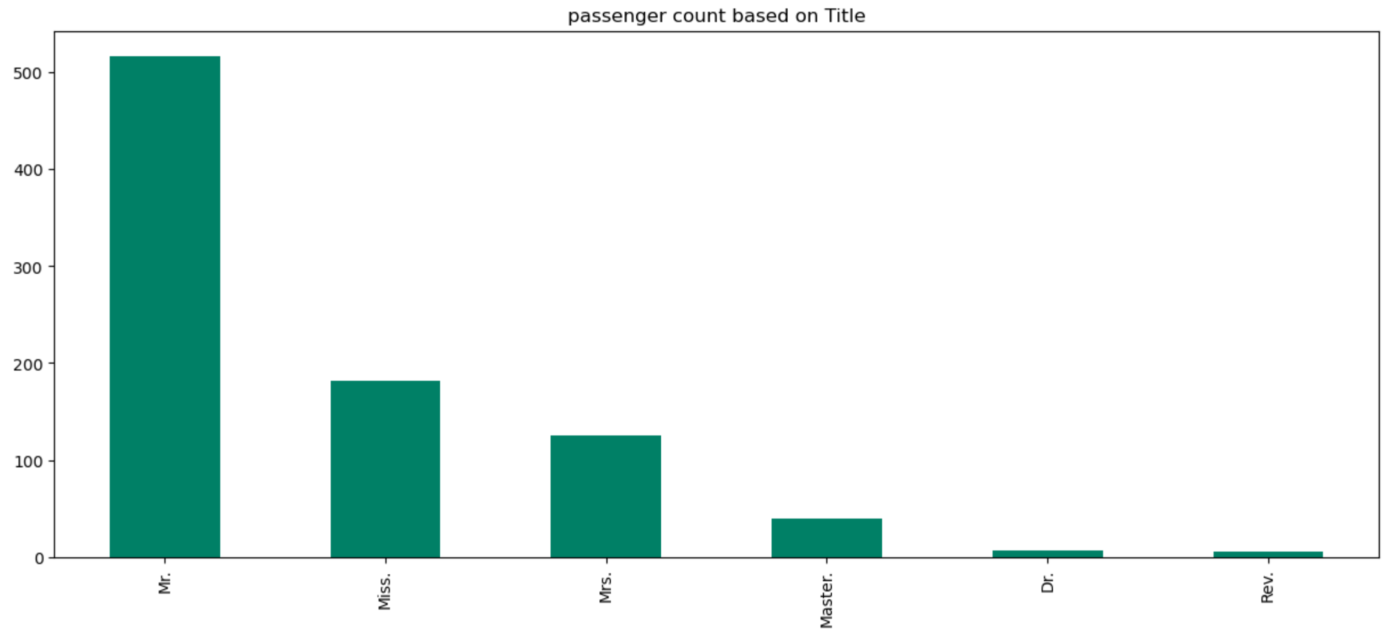


the passenger count is high for southamptin.

In [88]:

```
plt.figure(figsize=(15,6))
a=df2['Title'].value_counts()
a[a>2].plot(kind='bar', cmap='summer')
```

```
plt.title('passenger count based on Title')
plt.show()
```



passenger with title mr. is higher in count followed by miss.

In [91]:

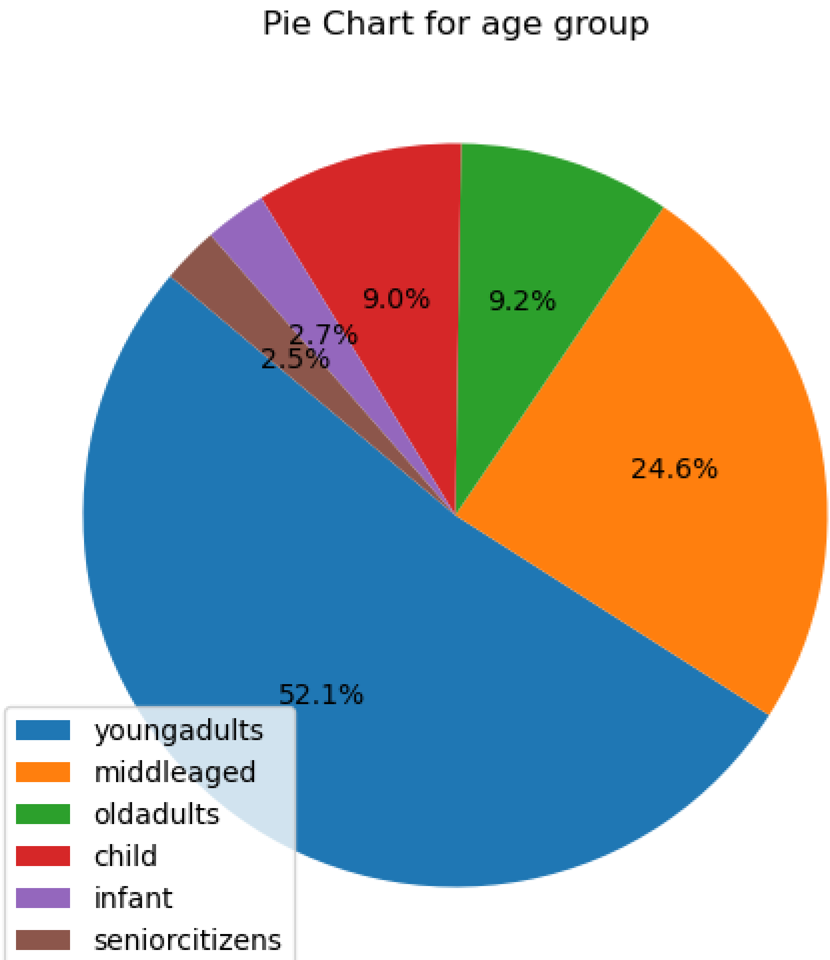
```
a=df2['age group'].value_counts()
a.index
```

Out[91]:

```
Index(['youngadults', 'middleaged', 'oldadults', 'child', 'infant',
       'seniorcitizens'],
      dtype='object')
```

In [92]:

```
# creating  apie diagram to knoe the percentage of age group
plt.figure(figsize=(6,8))
plt.pie(a, autopct='%1.1f%%',
        startangle=140)
plt.legend(a.index)
plt.title('Pie Chart for age group')
plt.show()
```

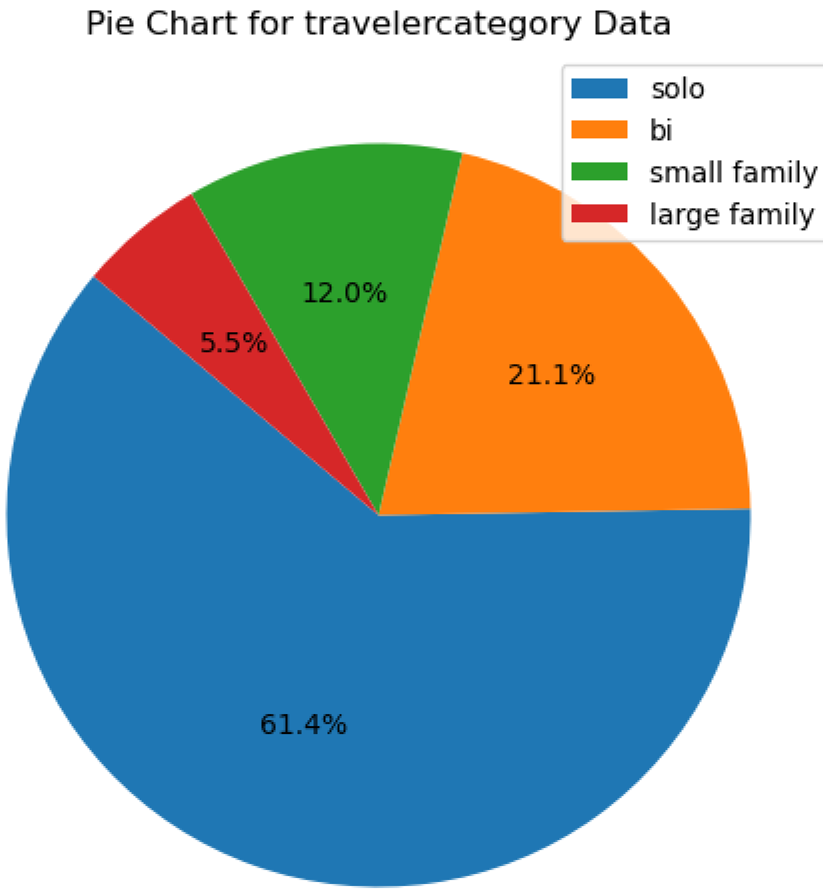


52 percentage of the passengers were young adults , 24 were middleaaged and senior citizens were the lowest with 2.5%.

In [ ]:

In [94]:

```
# pie chat to understand the percentage of people in each travel catgeory
b=df2['travelercategory'].value_counts()
plt.figure(figsize=(6,8))
plt.pie(b, autopct='%1.1f%%',
        startangle=140)
plt.legend(b.index)
plt.title('Pie Chart for travelercategory Data')
plt.show()
```



61 percentage of the total travelrs were solo travelers followed by bi-travelers.

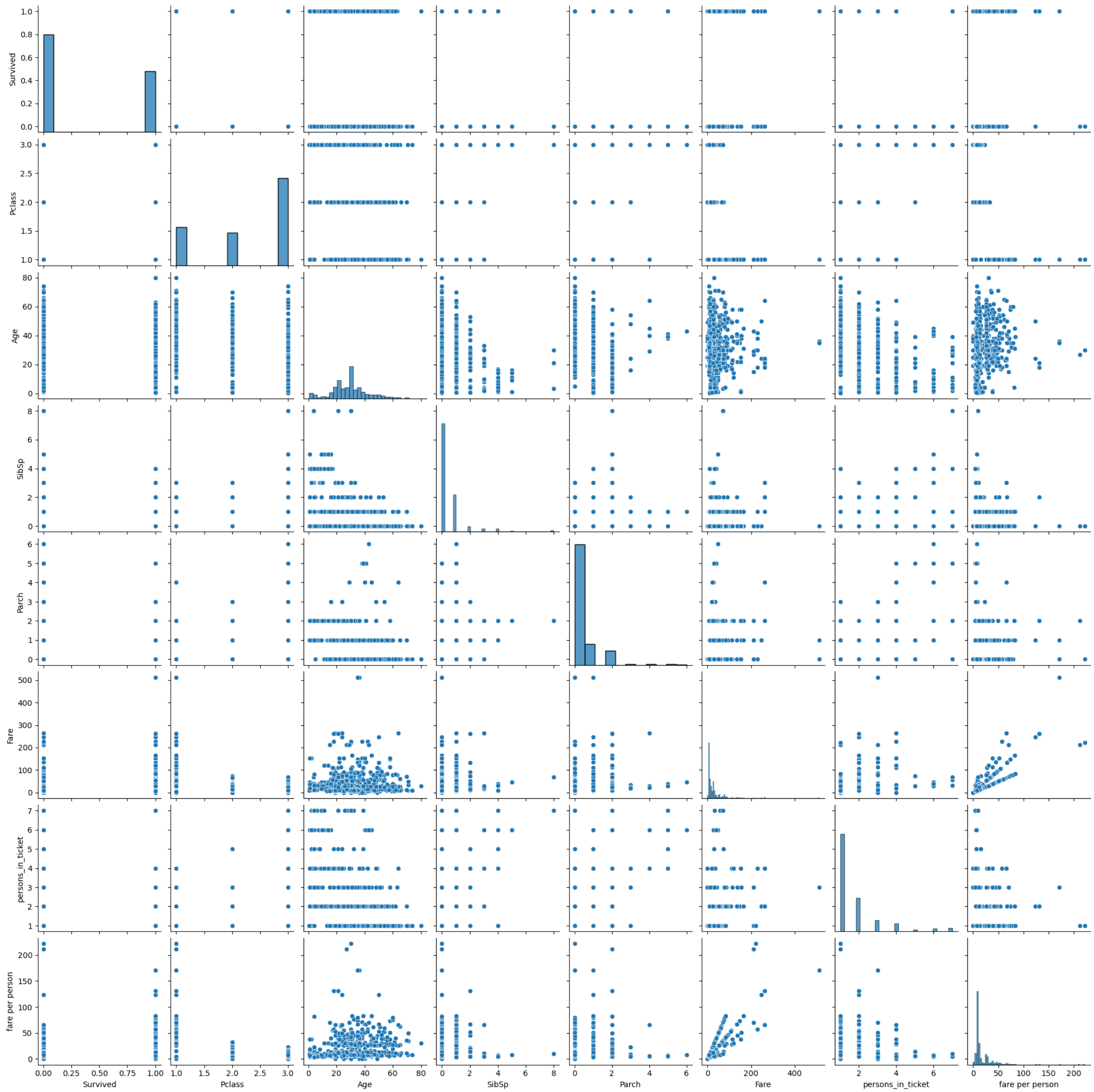
Bivariate analysis

In [95]:

```
sns.pairplot(df2)
```

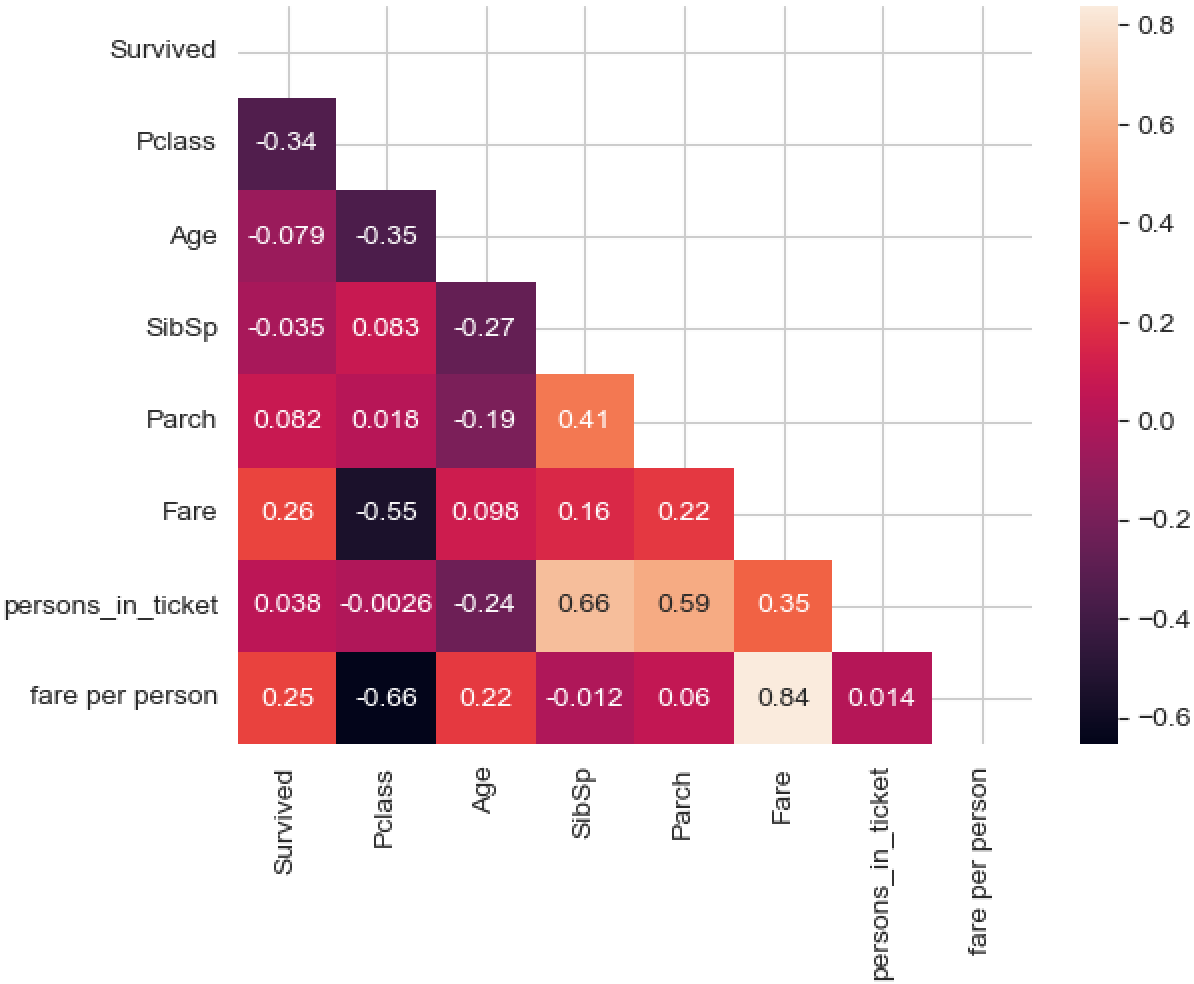
```
C:\Users\telsm\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The fig
ure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[95]: <seaborn.axisgrid.PairGrid at 0x21cd73e8090>



In [164…

```
sns.heatmap(df2.corr(), annot=True, mask=np.triu(df2.corr()))
plt.show()
```



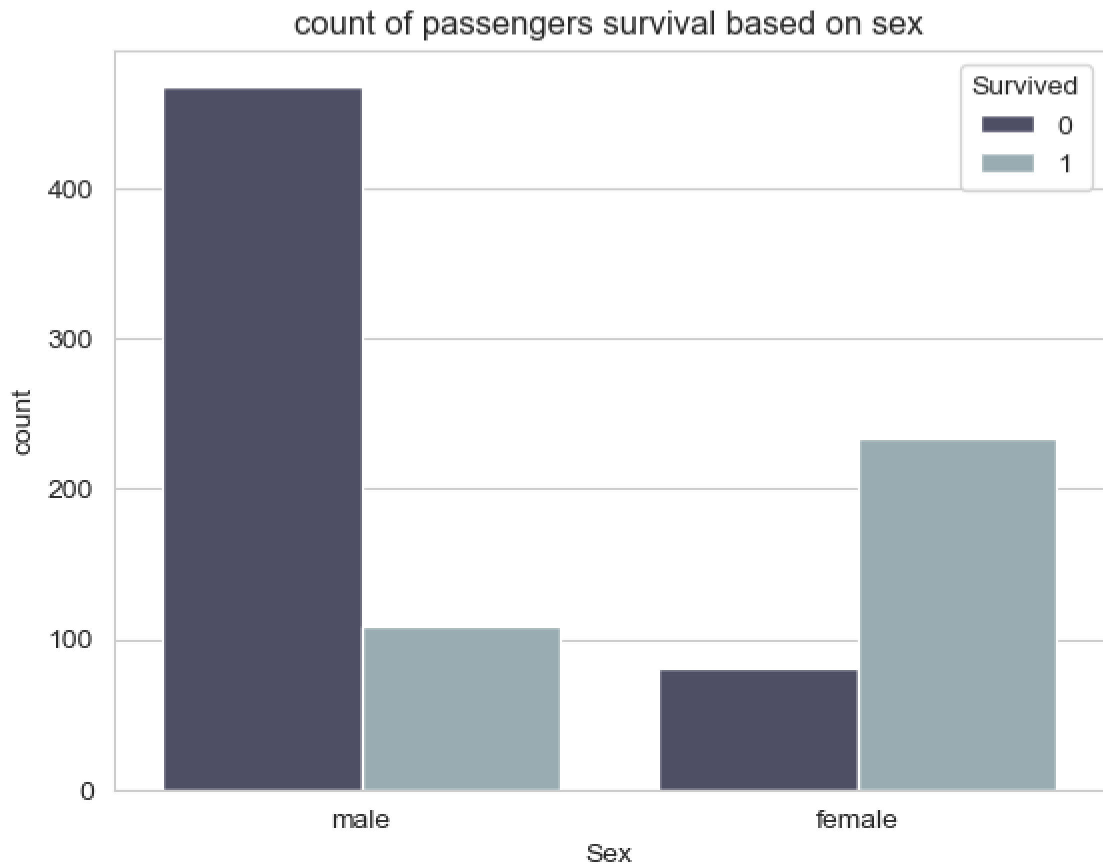
there is be a correlation between fare and passengerclass

visualization of variables with arget column. ('Survived')

sex and survived

In [104…

```
sns.set_style("whitegrid")

sns.countplot(hue=df2['Survived'], x=(df2['Sex']),palette='bone' )
plt.title('count of passengers survival based on sex')
plt.show()
```



In [122…
```
df2[['Sex','Survived']].value_counts()
```

Out[122…
```
Sex     Survived
male    0           468
female  1           233
male    1           109
female  0            81
dtype: int64
```

In [115…
```
df2[['Sex']].value_counts()
```

Out[115…
```
Sex
male      577
female    314
dtype: int64
```

In [130…
```
df2['Sex'].count()
```

Out[130…
```
891
```

In [140…
```
# % of male survived = survived male passengers /total male passenger
t_s=((233+109)/891)*100
t_m_s=(109/891)*100
t_f_s=(233/891)*100
p_m=(109/577)*100
f_m=(233/314)*100
print('total survival percentage of passengers',t_s )
print('total male survival percentage in survived',t_m_s)
print('total female survival percentage in survived',t_f_s)
print('percentage of male survived', p_m)
print('percentage of females survived', f_m)
```

```
total survival percentage of passengers 38.38383838383838
```

```
total male survival percentage in survived 12.2334455667789
total female survival percentage in survived 26.15039281705948
percentage of male survived 18.890814558058924
percentage of females survived 74.20382165605095
```

the survival percentage of females is high

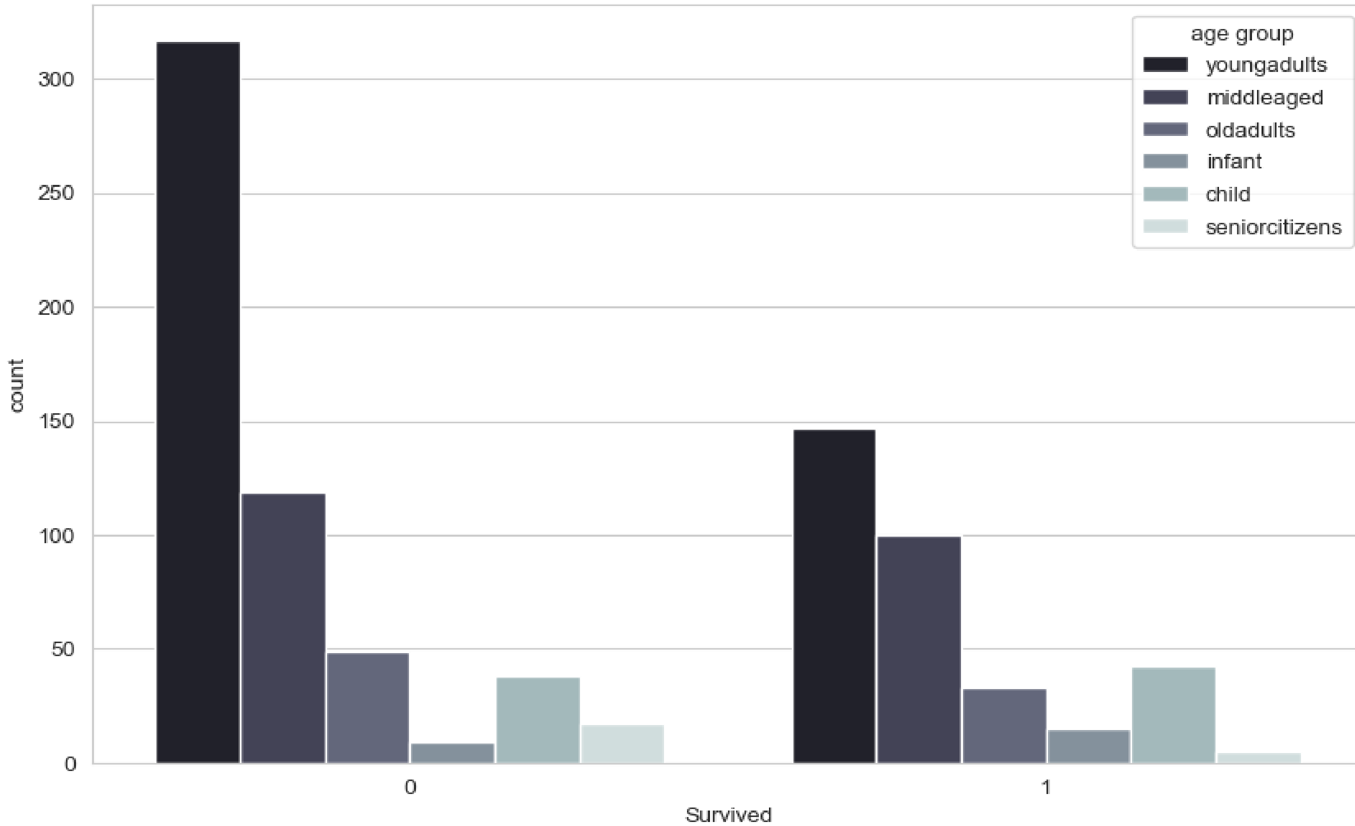
survived vs age group

In [141…

```
plt.figure(figsize=(10,6))
sns.countplot(x=df2['Survived'], hue=df2['age group'], palette='bone')
```

Out[141…

```
<Axes: xlabel='Survived', ylabel='count'>
```



In [ ]:

```
count of youngadults who died is more.
```

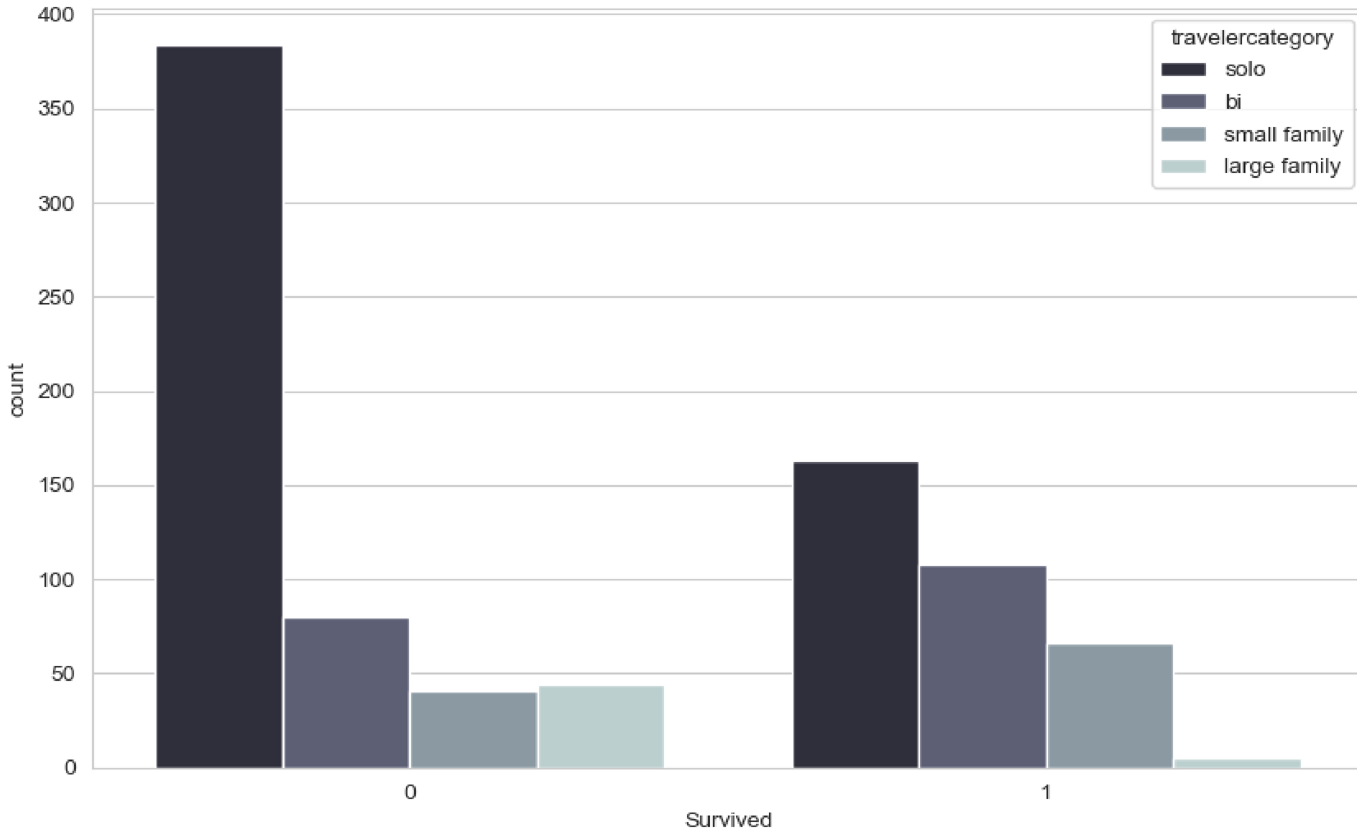
traveler category vs survived

In [142…

```
plt.figure(figsize=(10,6))
sns.countplot(x=df2['Survived'], hue=df2['travelercategory'], palette='bone')
```

Out[142…

```
<Axes: xlabel='Survived', ylabel='count'>
```



In [ ]:

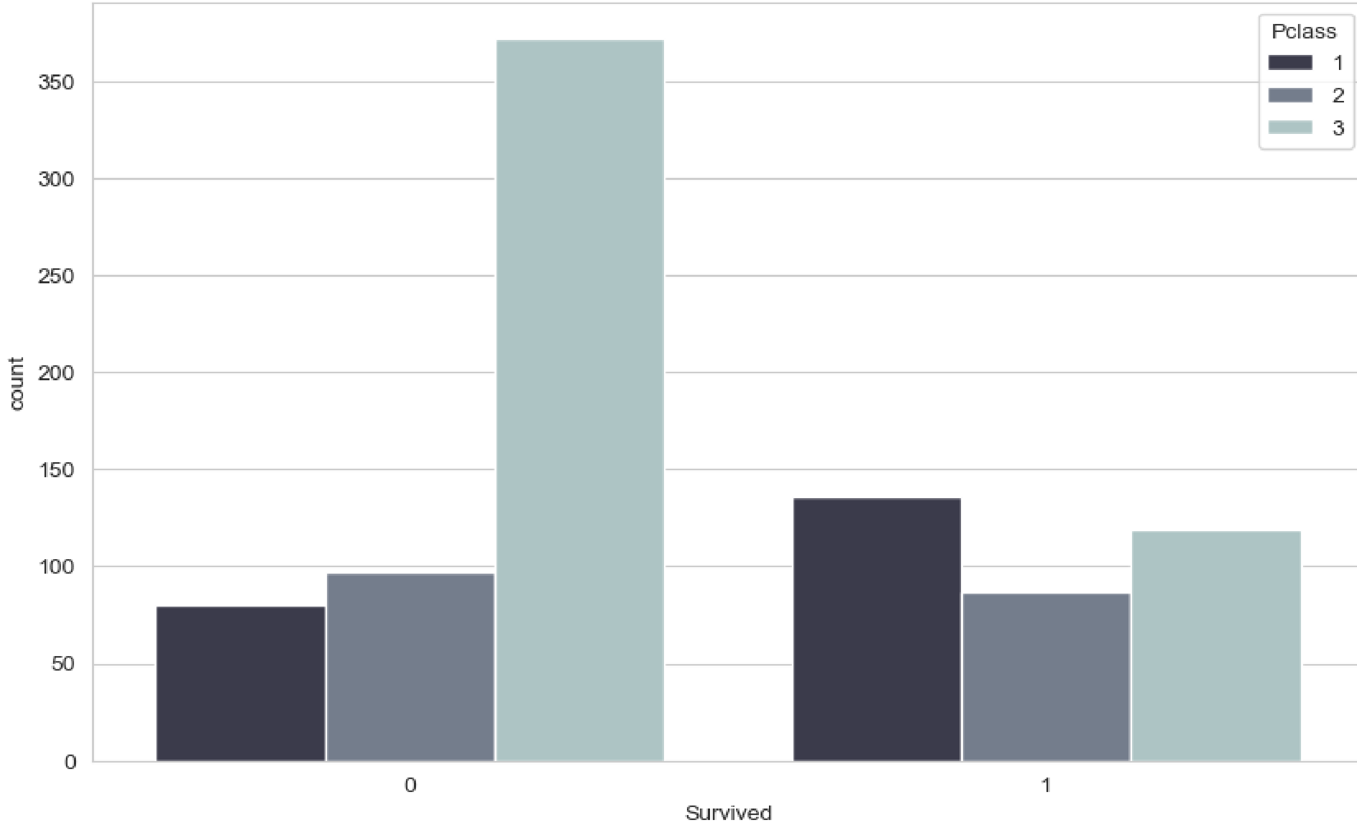
pclass vs survived

In [145…

```
plt.figure(figsize=(10,6))
sns.countplot(x=df2['Survived'], hue=df2['Pclass'], palette='bone')
```

Out[145…

```
<Axes: xlabel='Survived', ylabel='count'>
```

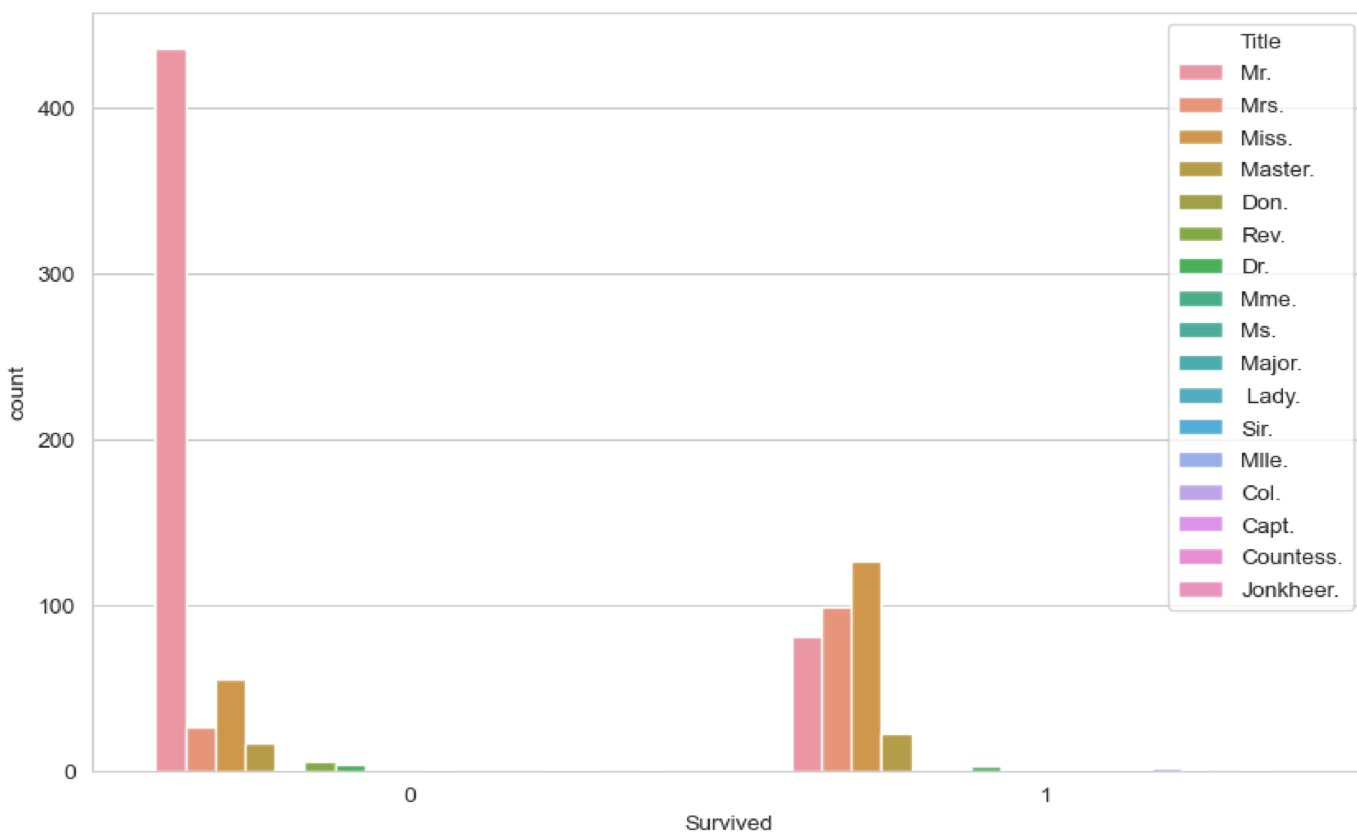


the third class passengers survival count is low

title vs survived

In [65]:
```
plt.figure(figsize=(10,6))
sns.countplot(x=df2['Survived'], hue=df2['Title'])
```

Out[65]: <Axes: xlabel='Survived', ylabel='count'>

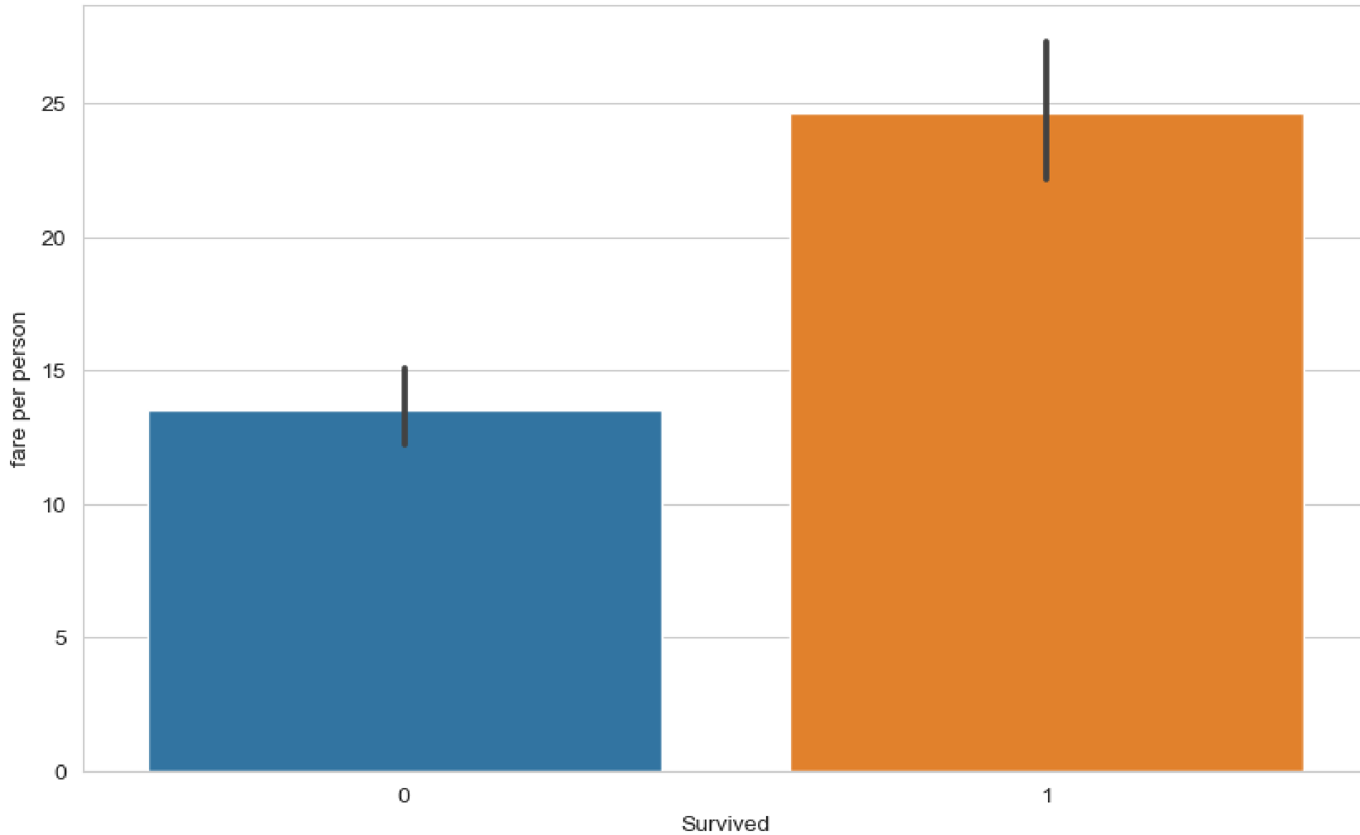


In [ ]:
```
mr. title people died more
```

fare per person vs survival category

In [66]:
```
plt.figure(figsize=(10,6))
sns.barplot(x=df2['Survived'], y=df2['fare per person'])
```

Out[66]: <Axes: xlabel='Survived', ylabel='fare per person'>

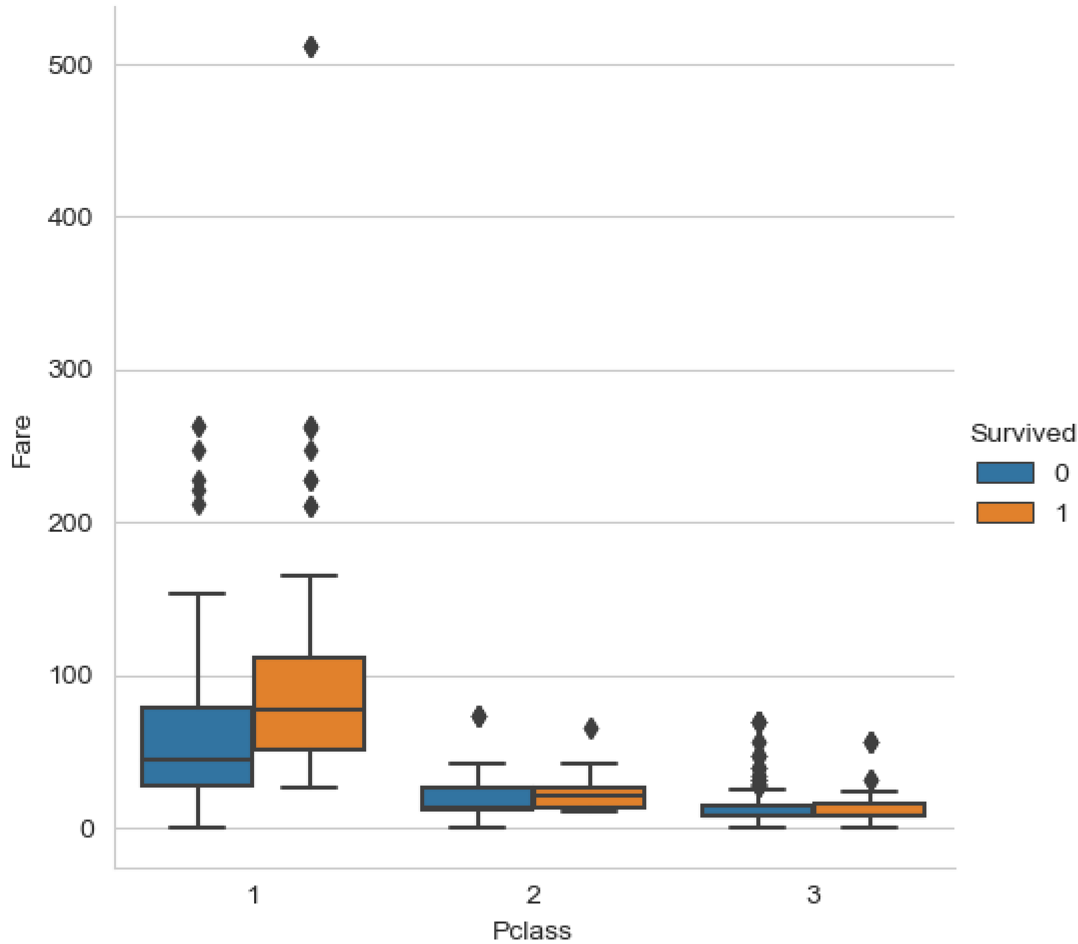


In [ ]:
```
survived category has comparably high fare.
```

In [68]:
```
from gapminder import gapminder
```

In [165…
```
sns.catplot(data=df2, x="Pclass", y="Fare", hue="Survived", kind="box")
```

Out[165… <seaborn.axisgrid.FacetGrid at 0x21ce163f050>



fare is high for survived passengers.

In [ ]:

```

```

MODEL BUILDING

In [148…

```
# import the libraries
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble  import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [149…

```
from sklearn.model_selection import cross_val_score,KFold
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split,GridSearchCV
from scipy import stats
from sklearn.metrics import classification_report,roc_curve,confusion_matrix,\
accuracy_score,recall_score,precision_score,f1_score,cohen_kappa_score
from sklearn.metrics import log_loss
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```python
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score, confusion_matrix,acc
```

In [38]:

```python

```

In [150…

```python
# drop ticket and title
df3=df2.drop(['Ticket','Title'], axis=1)
```

In [151…

```python
df3
```

Out[151…

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck	all possible Decks	persons_in_ticket	fa p pers
0	0	3	male	22.0	1	0	7.2500	S	not available	D/E/F/G	1	7.25
1	1	1	female	38.0	1	0	71.2833	C	C	A/B/C/D	1	71.28
2	1	3	female	26.0	0	0	7.9250	S	not available	D/E/F/G	1	7.92
3	1	1	female	35.0	1	0	53.1000	S	C	A/B/C/D	2	26.55
4	0	3	male	35.0	0	0	8.0500	S	not available	D/E/F/G	1	8.05
...	...	...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	0	0	13.0000	S	not available	D/E/F	1	13.00
887	1	1	female	19.0	0	0	30.0000	S	B	A/B/C/D	1	30.00
888	0	3	female	21.0	1	2	23.4500	S	not available	D/E/F/G	2	11.72
889	1	1	male	26.0	0	0	30.0000	C	C	A/B/C/D	1	30.00
890	0	3	male	32.0	0	0	7.7500	Q	not available	D/E/F/G	1	7.75

891 rows × 14 columns

In [152…

```python
# get dummies
df4=pd.get_dummies(df3, drop_first=True)
```

In [153…

```python
df4
```

Out[153…

	Survived	Pclass	Age	SibSp	Parch	Fare	persons_in_ticket	fare per person	Sex_male	Embarked_Q	...	all Decks_
0	0	3	22.0	1	0	7.2500	1	7.2500	1	0	...	
1	1	1	38.0	1	0	71.2833	1	71.2833	0	0	...	
2	1	3	26.0	0	0	7.9250	1	7.9250	0	0	...	
3	1	1	35.0	1	0	53.1000	2	26.5500	0	0	...	

4	0	3	35.0	0	0	8.0500	1	8.0500	1	0	...
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	27.0	0	0	13.0000	1	13.0000	1	0	...
887	1	1	19.0	0	0	30.0000	1	30.0000	0	0	...
888	0	3	21.0	1	2	23.4500	2	11.7250	0	0	...
889	1	1	26.0	0	0	30.0000	1	30.0000	1	0	...
890	0	3	32.0	0	0	7.7500	1	7.7500	1	1	...

891 rows × 30 columns

In [154…

```
# create an empty dataframe to store the scores for various algorithms
perf_score = pd.DataFrame(columns=["Model", "Accuracy","Recall",
                                   "Precision","F1 Score",'reliability'] )
```

In [155…

```
def per_measures(model,test,pred):
    accuracy    = accuracy_score(test,pred)
    f1score     = f1_score(test,pred)
    recall      = recall_score(test,pred)
    precision   = precision_score(test,pred)
    reliability = cohen_kappa_score(test,pred)
    return (accuracy,recall,precision,f1score,reliability)
```

In [156…

```
def update_performance (name, model,test,pred):
    # assign 'comp_perf' as global variable
    global perf_score


    # append the results to the dataframe 'score_card'
    # 'ignore_index = True' do not consider the index labels
    perf_score = perf_score.append({'Model'       : name,
                                    'Accuracy'    : per_measures(model,test,pred)[0],
                                    'Recall'      : per_measures(model,test,pred)[1],
                                    'Precision'   : per_measures(model,test,pred)[2],
                                    'F1 Score'    : per_measures(model,test,pred)[3],
                                    'reliability' : per_measures(model,test,pred)[4]

                                     },
                                   ignore_index = True)
```

In [ ]:

In [ ]:

SPLITTING THE DATA

In [157…

```
x=df4.drop('Survived', axis=1)
y=df4['Survived']
```

In [158…

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2, random_state=10)
```

In [159…
```
xtrain.shape,xtest.shape,ytrain.shape,ytest.shape
```
Out[159…
```
((712, 29), (179, 29), (712,), (179,))
```
In [ ]:

Logistic regression -- base model

In [166…
```
# call the model
lgt=LogisticRegression(random_state=10)
```
In [167…
```
# fit the model
lgt.fit(xtrain,ytrain)
```
Out[167…
```
▾ LogisticRegression
LogisticRegression(random_state=10)
```
In [168…
```
# get the predictions
y_pred=lgt.predict(xtest)
```
In [169…
```
# get the predicted probabilities
y_pred_prop = lgt.predict_proba(xtest)[:,1]
```
In [170…
```
# update performance
update_performance (name='logistic_base', model=lgt,test=ytest,pred=y_pred)
```
In [171…
```
perf_score
```
Out[171…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081

KNN model - base model

In [172…
```
knn = KNeighborsClassifier()
```
In [173…
```
knn.fit(xtrain,ytrain)
```
Out[173…
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```
In [174…
```
xtest_array = xtest.values if isinstance(xtest, pd.DataFrame) else xtest
y_pred = knn.predict(xtest_array)
```

In [175…

```
update_performance(name = 'KNN-Base', model = knn,test=ytest,pred=y_pred)
perf_score
```

Out[175…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314

Decision Tree

In [176…

```
dt = DecisionTreeClassifier(random_state=10)
dt.fit(xtrain,ytrain)
ypred_dt = dt.predict(xtest)
```

In [177…

```
update_performance(name = 'Decision Tree-Gini', model = dt,test=ytest,
                   pred=ypred_dt)
perf_score
```

Out[177…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956

In [178…

```
dt_e = DecisionTreeClassifier(criterion='entropy',random_state=10)
dt_e.fit(xtrain,ytrain)
ypred_dt_e = dt_e.predict(xtest)
```

In [179…

```
update_performance(name = 'Decision Tree-Entropy', model = dt_e,test=ytest,pred=ypred_dt
perf_score
```

Out[179…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333

In [ ]:

```

```

DecisionTree-TUNED

In [180…

```
tuned_paramaters = [{'criterion': ['entropy','gini'],
                     'max_depth': [3,5,7,9,11],
                     'max_features': ["sqrt", "log2"],
```

```
                    'min_samples_split': [2,5,8],
                    'min_samples_leaf': [1,5,9],
                    'max_leaf_nodes': [5,8]}]
```

In [181…

```
dt = DecisionTreeClassifier(random_state = 10)
tree_grid = GridSearchCV(estimator = dt,
                         param_grid = tuned_paramaters,
                         cv = 5)
```

In [185…

```
tree_grid_model = tree_grid.fit(xtrain, ytrain)
tree_grid_model.best_params_
```

Out[185…

```
{'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 'log2',
 'max_leaf_nodes': 8,
 'min_samples_leaf': 9,
 'min_samples_split': 2}
```

In [186…

```
dt_grid_model = DecisionTreeClassifier(criterion = 'entropy',
                                  max_depth = 5,
                                  max_features = 'log2',
                                  max_leaf_nodes = 8,
                                  min_samples_leaf = 9,
                                  min_samples_split = 2)
```

In [187…

```
dt_grid_model= dt_grid_model.fit(xtrain,ytrain)
```

In [188…

```
ypred_dt_tp = dt_grid_model.predict(xtest)
```

In [189…

```
update_performance(name = 'Decision Tree-tuned_entropy', model = dt_grid_model,test=ytes
perf_score
```

Out[189…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003

In [ ]:

In [ ]:

Gaussian Naive Bayes

In [190…

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
```

In [191…

```
gnb = GaussianNB()
gnb.fit(xtrain,ytrain)
ypred_gnb = gnb.predict(xtest)
```

In [192…

```
update_performance(name = 'Gaussian NB',
                   model = gnb,
                   test=ytest,
                   pred=ypred_gnb)
perf_score
```

Out[192…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657

In [193…

```
bnb = BernoulliNB()
bnb.fit(xtrain,ytrain)

ypred_bnb = bnb.predict(xtest)
```

In [194…

```
update_performance(name = 'Bernoulli NB', model = bnb,test=ytest,pred=ypred_bnb)

# print the dataframe
perf_score
```

Out[194…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159

In [195…

```
mnb = MultinomialNB()

mnb.fit(xtrain,ytrain)


ypred_mnb = mnb.predict(xtest)
```

In [196…

```
update_performance(name = 'Multinomial NB', model = mnb,test=ytest,pred=ypred_mnb)
```

```
# print the dataframe
perf_score
```

Out[196…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878

Random Forest

In [197…

```
rf = RandomForestClassifier(random_state=10)
rf.fit(xtrain,ytrain)
ypred_rf = rf.predict(xtest)
```

In [198…

```
update_performance(name = 'Random Forest',
                   model = rf,
                   test=ytest,
                   pred=ypred_rf)
perf_score
```

Out[198…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025

OOB SAMPLE -PERFORMANCE

In [199…

```
rf = RandomForestClassifier(oob_score=True,random_state=10)

rf.fit(xtrain,ytrain)

ypred_rf = rf.predict(xtest)
```

In [200…

```
params =                 [{'criterion': ['entropy', 'gini'],
                          'n_estimators': [100],
                          'max_depth': [10,20],
                          'max_features': ['sqrt', 'log2'],
                         'min_samples_split': [2, 8],
                          'min_samples_leaf': [5, 9],
                          'max_leaf_nodes': [8, 11]}]
```

In [87]:

```
rf =RandomForestClassifier(random_state=10)

rf_cv = GridSearchCV(rf,params,cv=5,scoring='accuracy')

rf_cv.fit(xtrain,ytrain)

rf_cv.best_params_
```

Out[87]:

```
{'criterion': 'gini',
 'max_depth': 10,
 'max_features': 'sqrt',
 'max_leaf_nodes': 11,
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'n_estimators': 100}
```

In [201…

```
rf_model = RandomForestClassifier(criterion ='gini' ,
                                  n_estimators =100,
                                  max_depth = 10 ,
                                  max_features = 'sqrt',
                                  max_leaf_nodes =11,
                                  min_samples_leaf =5 ,
                                  min_samples_split = 2,
                                  )


rf_model.fit(xtrain,ytrain)

ypred_rf_tp = rf_model.predict(xtest)
```

In [202…

```
update_performance(name = 'Random Forest-Tunned', model = rf_model,test=ytest,pred=ypred

# print the dataframe
perf_score
```

Out[202…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095

Bagging classifier

In [203…

```
dt=DecisionTreeClassifier(random_state=10)
bc=BaggingClassifier(dt)
bc.fit(xtrain,ytrain)
ypred_bc = bc.predict(xtest)
```

In [204…

```
update_performance(name = 'Bagging Classifier-dt',
                   model = bc,
                   test=ytest,
                   pred=ypred_bc)
perf_score
```

Out[204…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383

In [205…

```
knn=KNeighborsClassifier()

bag_knn=BaggingClassifier(knn)
bag_knn.fit(xtrain,ytrain)

ypred_bag_knn = bag_knn.predict(xtest)
```

In [206…

```
update_performance(name = 'Bagging Classifier-knn', model = bag_knn,test=ytest,pred=ypre

# print the dataframe
perf_score
```

Out[206…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003

5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522

In [207…

```
logr=LogisticRegression()

bag_logr=BaggingClassifier(logr)
bag_logr.fit(xtrain,ytrain)

ypred_bag_logr = bag_logr.predict(xtest)
```

In [208…

```
update_performance(name = 'Bagging Classifier-Log', model = bag_logr,test=ytest,pred=ypr

# print the dataframe
perf_score
```

Out[208…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066

AdaBoost

In [209…

```
abcl = AdaBoostClassifier(random_state=10)
abcl.fit(xtrain,ytrain)

ypred_abcl = abcl.predict(xtest)
```

In [210…

```
update_performance(name = 'AdaBoost-dt',
```

```
                        model = abcl,
                        test=ytest,
                        pred=ypred_abcl)
perf_score
```

Out[210…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066
13	AdaBoost-dt	0.810056	0.790323	0.700000	0.742424	0.592855

Gradientboost

In [211…

```
gbcl = GradientBoostingClassifier(random_state=10)
gbcl.fit(xtrain,ytrain)
ypred_gbcl = gbcl.predict(xtest)
```

In [212…

```
update_performance(name = 'Gradient Boosting',
                   model = gbcl,
                   test=ytest,
                   pred=ypred_gbcl)

# print the dataframe
perf_score
```

Out[212…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159

7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066
13	AdaBoost-dt	0.810056	0.790323	0.700000	0.742424	0.592855
14	Gradient Boosting	0.843575	0.790323	0.765625	0.777778	0.657135

XGBOOST

In [213…

```
xgb= XGBClassifier(random_state=10)
xgb.fit(xtrain,ytrain)

ypred_xgb= xgb.predict(xtest)
```

In [214…

```
update_performance(name = 'XGB', model = xgb,test=ytest,pred=ypred_xgb)

# print the dataframe
perf_score
```

Out[214…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066
13	AdaBoost-dt	0.810056	0.790323	0.700000	0.742424	0.592855
14	Gradient Boosting	0.843575	0.790323	0.765625	0.777778	0.657135
15	XGB	0.837989	0.774194	0.761905	0.768000	0.643549

STACKING

In [215…
```
lgr=LogisticRegression()
RF=RandomForestClassifier()
dt=DecisionTreeClassifier()
```

In [212…
```

```

In [216…
```
base_learners=[('lr_model',lgr),('DT_model',dt)]

stack =StackingClassifier(estimators=base_learners)
stack.fit(xtrain,ytrain)
ypred_stack = stack.predict(xtest_array )
```

In [217…
```
update_performance(name = 'Stacking', model = stack,test=ytest,pred=ypred_stack)

# print the dataframe
perf_score
```

Out[217…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066
13	AdaBoost-dt	0.810056	0.790323	0.700000	0.742424	0.592855
14	Gradient Boosting	0.843575	0.790323	0.765625	0.777778	0.657135
15	XGB	0.837989	0.774194	0.761905	0.768000	0.643549
16	Stacking	0.810056	0.725806	0.725806	0.725806	0.580507

voting

In [218…
```
vote_soft =VotingClassifier(estimators=base_learners ,voting='soft')

vote_soft.fit(xtrain,ytrain)
ypred_vote = vote_soft.predict(xtest)
```

In [219…

```
update_performance(name = 'Voting', model = vote_soft,test=ytest,pred=ypred_vote)

# print the dataframe
perf_score
```

Out[219…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066
13	AdaBoost-dt	0.810056	0.790323	0.700000	0.742424	0.592855
14	Gradient Boosting	0.843575	0.790323	0.765625	0.777778	0.657135
15	XGB	0.837989	0.774194	0.761905	0.768000	0.643549
16	Stacking	0.810056	0.725806	0.725806	0.725806	0.580507
17	Voting	0.798883	0.774194	0.685714	0.727273	0.568906

In [ ]:

SUMMARY OF MODELS

In [220…

```
perf_score.sort_values(by='F1 Score', ascending=False)
```

Out[220…

	Model	Accuracy	Recall	Precision	F1 Score	reliability
4	Decision Tree-tuned_entropy	0.854749	0.822581	0.772727	0.796875	0.684003
14	Gradient Boosting	0.843575	0.790323	0.765625	0.777778	0.657135
15	XGB	0.837989	0.774194	0.761905	0.768000	0.643549
0	logistic_base	0.837989	0.741935	0.779661	0.760331	0.638081
12	Bagging Classifier-Log	0.826816	0.741935	0.754098	0.747967	0.616066
13	AdaBoost-dt	0.810056	0.790323	0.700000	0.742424	0.592855
10	Bagging Classifier-dt	0.815642	0.741935	0.730159	0.736000	0.594383
9	Random Forest-Tunned	0.826816	0.693548	0.781818	0.735043	0.607095

3	Decision Tree-Entropy	0.804469	0.774194	0.695652	0.732824	0.579333
17	Voting	0.798883	0.774194	0.685714	0.727273	0.568906
16	Stacking	0.810056	0.725806	0.725806	0.725806	0.580507
8	Random Forest	0.804469	0.741935	0.707692	0.724409	0.573025
2	Decision Tree-Gini	0.787709	0.758065	0.671429	0.712121	0.544956
6	Bernoulli NB	0.787709	0.725806	0.681818	0.703125	0.538159
7	Multinomial NB	0.782123	0.596774	0.725490	0.654867	0.497878
5	Gaussian NB	0.625698	0.887097	0.478261	0.621469	0.311657
11	Bagging Classifier-knn	0.715084	0.661290	0.577465	0.616541	0.391522
1	KNN-Base	0.720670	0.629032	0.590909	0.609375	0.392314

from the models that i build gradient boost is the best performing model.

Opted model is Decision Tree-tuned_entropy

In [221…
```
dt_grid_model
```

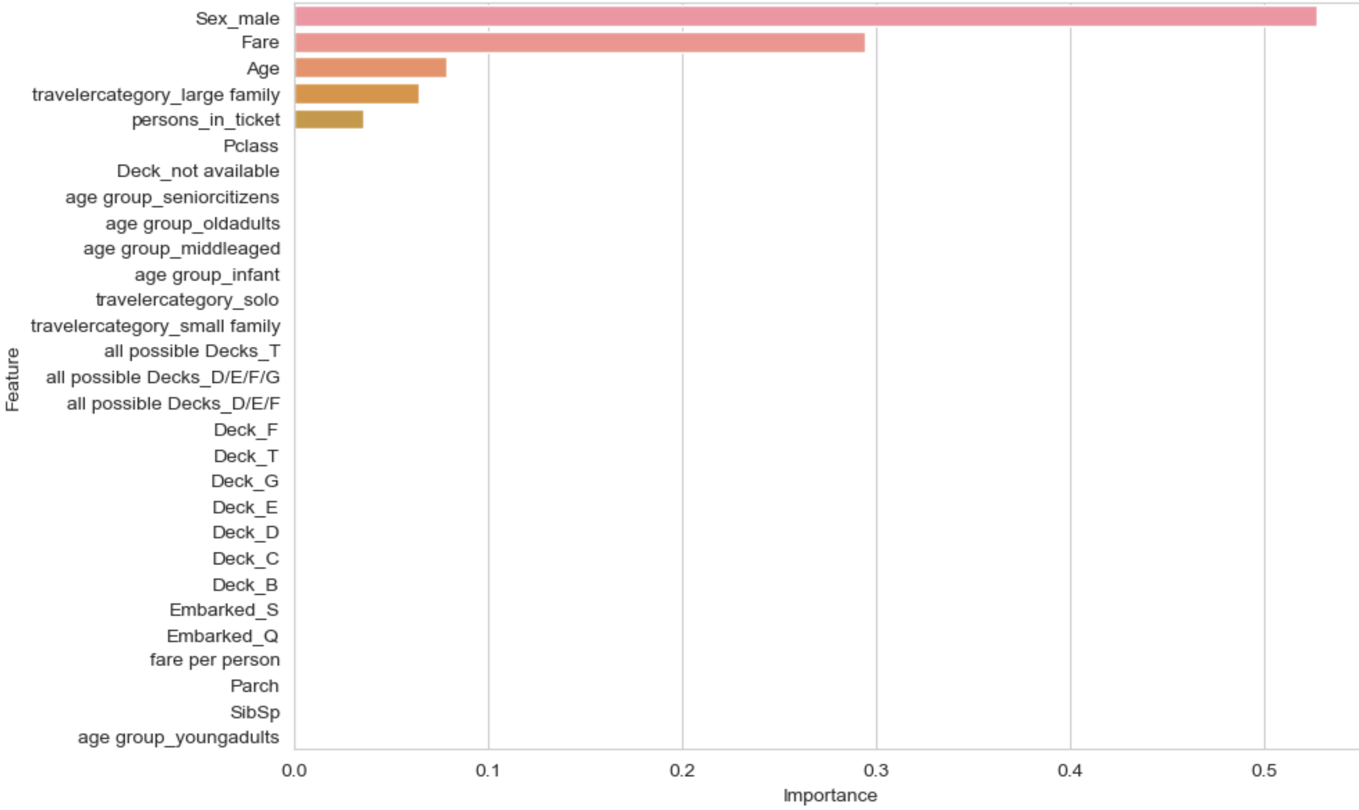
Out[221…
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, max_features='log2',
                       max_leaf_nodes=8, min_samples_leaf=9)
```

In [222…
```
ypred_dt_tp =dt_grid_model.predict(xtest)
```

In [ ]:
```

```

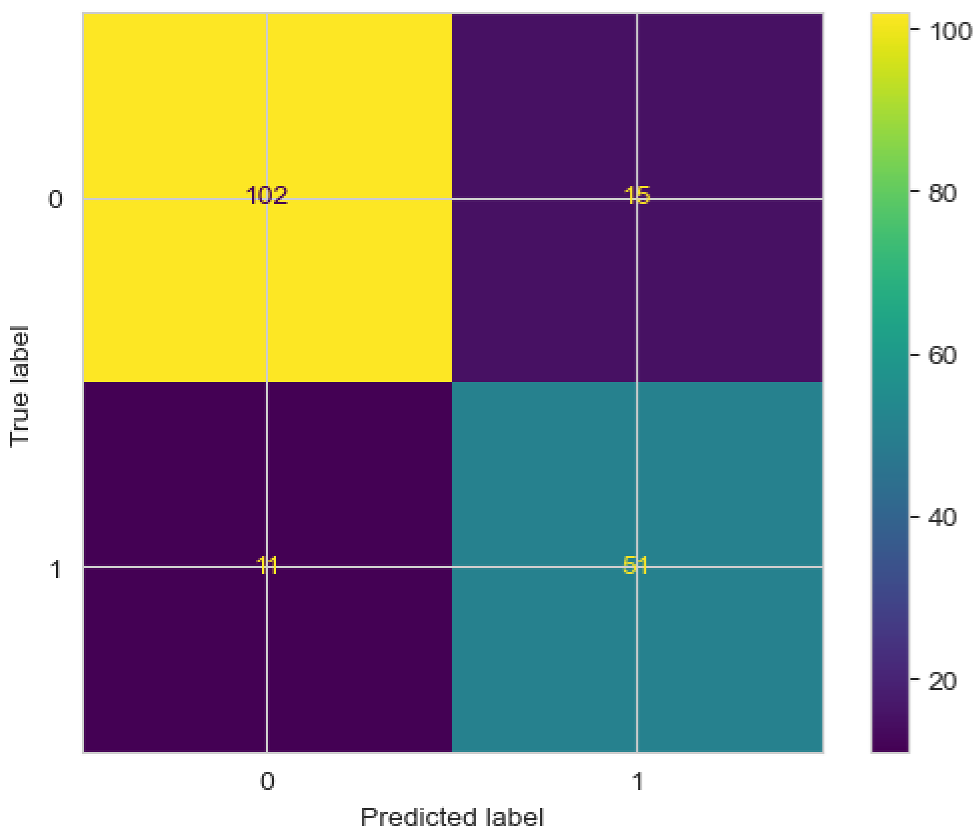
In [223…
```
feature_imp = pd.DataFrame()
feature_imp['Feature']=xtrain.columns
feature_imp['Importance']=dt_grid_model.feature_importances_
```

In [224…
```
plt.figure(figsize=(10,7))
feature_imp = feature_imp.sort_values('Importance',ascending=False)
sns.barplot(x='Importance', y='Feature', data=feature_imp)
plt.show()
```



In [226…

```
ConfusionMatrixDisplay.from_predictions(ytest,ypred_dt_tp)
plt.show()
```



In [ ]:

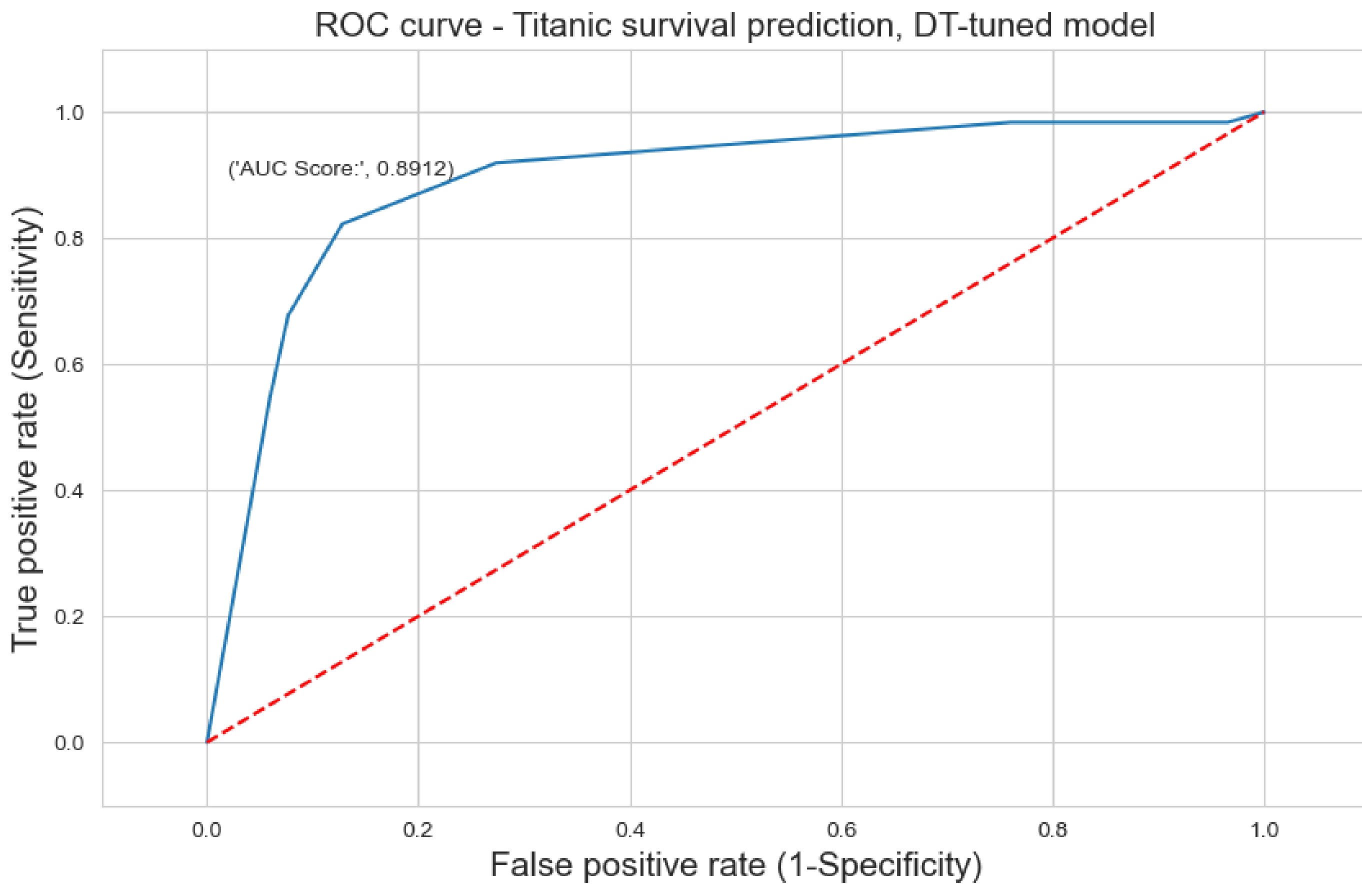
```

```

In [227…

```
y_pred_prob = dt_grid_model.predict_proba(xtest)[:,1]
y_pred_prob
```

Out [227…
```
array([0.11792453, 0.11792453, 0.11792453, 0.87878788, 0.87878788,
       0.11792453, 0.11792453, 0.10891089, 0.11792453, 0.11792453,
       0.10891089, 0.87878788, 0.87878788, 0.10891089, 0.10891089,
       0.11792453, 0.875     , 0.09090909, 0.57692308, 0.11792453,
       0.10891089, 0.10891089, 0.87878788, 0.11792453, 0.875     ,
       0.11792453, 0.11792453, 0.875     , 0.10891089, 0.87878788,
       0.87878788, 0.11792453, 0.87878788, 0.11792453, 0.57692308,
       0.11792453, 0.57692308, 0.375     , 0.11792453, 0.11792453,
       0.10891089, 0.10891089, 0.11792453, 0.57692308, 0.375     ,
       0.375     , 0.11792453, 0.11792453, 0.87878788, 0.375     ,
       0.375     , 0.375     , 0.11792453, 0.375     , 0.875     ,
       0.87878788, 0.09090909, 0.57692308, 0.87878788, 0.10891089,
       0.11792453, 0.87878788, 0.57692308, 0.87878788, 0.11792453,
       0.11792453, 0.375     , 0.11792453, 0.11792453, 0.87878788,
       0.10891089, 0.11792453, 0.11792453, 0.11792453, 0.87878788,
       0.11792453, 0.11792453, 0.875     , 0.375     , 0.11792453,
       0.10891089, 0.11792453, 0.09090909, 0.875     , 0.87878788,
       0.10891089, 0.57692308, 0.87878788, 0.10891089, 0.87878788,
       0.11792453, 0.11792453, 0.375     , 0.11792453, 0.87878788,
       0.57692308, 0.57692308, 0.10891089, 0.87878788, 0.10891089,
       0.875     , 0.375     , 0.11792453, 0.375     , 0.87878788,
       0.87878788, 0.11792453, 0.875     , 0.87878788, 0.375     ,
       0.87878788, 0.10891089, 0.87878788, 0.11792453, 0.11792453,
       0.375     , 0.375     , 0.87878788, 0.57692308, 0.375     ,
       0.87878788, 0.09090909, 0.87878788, 0.57692308, 0.87878788,
       0.57692308, 0.87878788, 0.10891089, 0.11792453, 0.875     ,
       0.87878788, 0.11792453, 0.11792453, 0.87878788, 0.87878788,
       0.57692308, 0.10891089, 0.375     , 0.87878788, 0.11792453,
       0.57692308, 0.11792453, 0.375     , 0.375     , 0.11792453,
       0.10891089, 0.10891089, 0.375     , 0.11792453, 0.11792453,
       0.11792453, 0.375     , 0.87878788, 0.11792453, 0.11792453,
       0.87878788, 0.11792453, 0.10891089, 0.11792453, 0.11792453,
       0.10891089, 0.57692308, 0.87878788, 0.11792453, 0.11792453,
       0.87878788, 0.09090909, 0.11792453, 0.87878788, 0.11792453,
       0.87878788, 0.875     , 0.11792453, 0.87878788, 0.375     ,
       0.10891089, 0.11792453, 0.11792453, 0.375     ])
```

In [238…
```
plt.figure(figsize=(10,6))
# the roc_curve() returns the values for false positive rate, true positive rate and thi
# pass the actual target values and predicted probabilities to the function
fpr, tpr, thresholds = roc_curve(ytest, y_pred_prob)

# plot the ROC curve
plt.plot(fpr, tpr)

# set limits for x and y axes
plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])

# plot the straight line showing worst prediction for the model
plt.plot([0, 1], [0, 1],'r--')

# add plot and axes labels
# set text size using 'fontsize'
plt.title('ROC curve - Titanic survival prediction, DT-tuned model', fontsize = 15)
plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)

# add the AUC score to the plot
# 'x' and 'y' gives position of the text
# 's' is the text
# use round() to round-off the AUC score upto 4 digits
plt.text(x = 0.02, y = 0.9, s = ('AUC Score:', round(roc_auc_score(ytest, y_pred_prob),4

# plot the grid
plt.grid(True)
```



In [229…
```
ypred_train_rf_tp = rf_model.predict(xtrain)
ypred_test_rf_tp = rf_model.predict(xtest)
```

In [230…
```
# getting the actual values and predicted values of train
a=pd.DataFrame({'ACTUAL':ytrain, 'PREDICTED': ypred_train_rf_tp})
```

In [231…
```
# getting the actual values and predicted values of test
b=pd.DataFrame({'ACTUAL':ytest, 'PREDICTED': ypred_test_rf_tp})
```

In [232…
```
#concating actual and predicted of test and train
c=pd.concat([a,b])
c
```

Out[232…

	ACTUAL	PREDICTED
57	0	0
717	1	1
431	1	0
633	0	0
163	0	0
...	...	...
456	0	0
191	0	0
603	0	0

94	0	0
766	0	0

891 rows × 2 columns

In [233…

```
# merging these values to the oroginal data frame.
final_titanic=df2.join(c)
```

In [234…

```
# created a new column in dataframe as correct prediction or incorrect prediction
def prediction(row):
    if row['ACTUAL'] == row['PREDICTED']:
        return 'correctprediction'
    else:
        return 'incorrectprediction'

final_titanic['prediction'] = final_titanic.apply(prediction, axis=1)
```

In [235…

```
# checking prediction count
final_titanic['prediction'].value_counts()
```

Out[235…

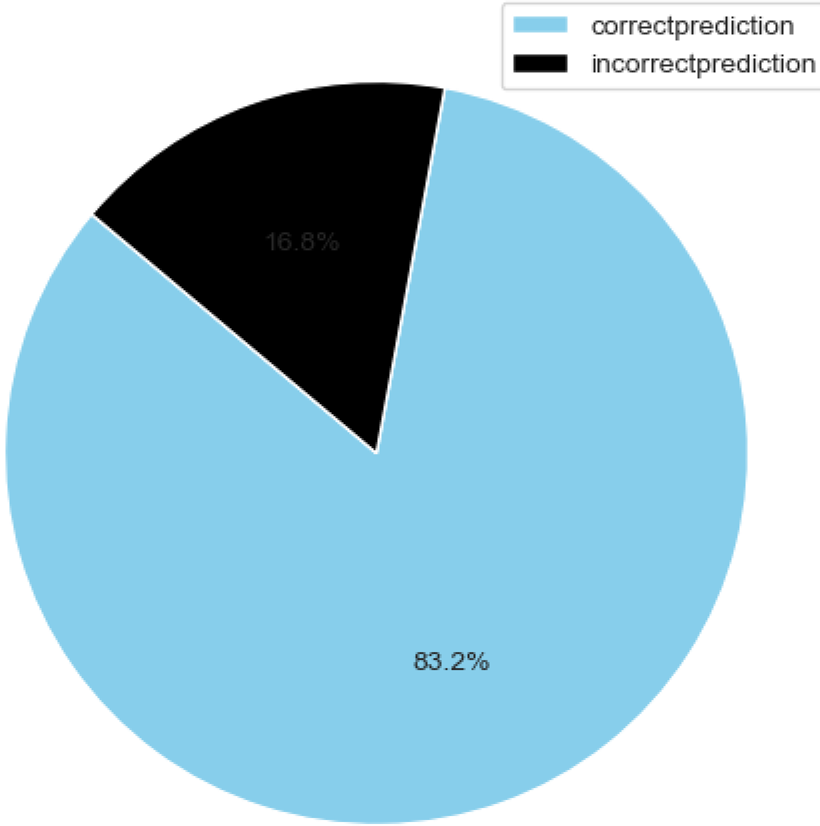
```
correctprediction      741
incorrectprediction    150
Name: prediction, dtype: int64
```

ANALYSIS OF THE PROJECT

In [244…

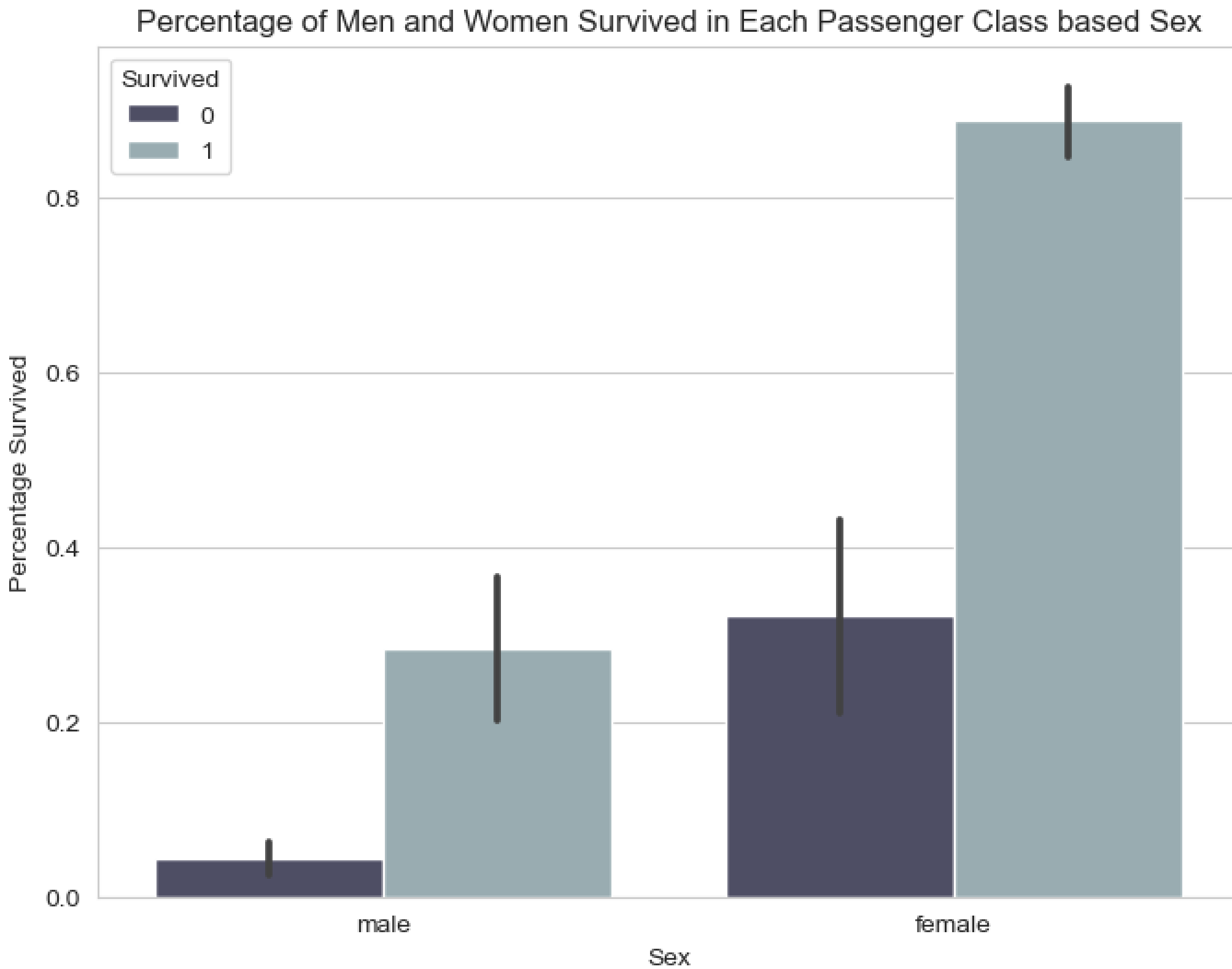
```
# checking the percentage of correct and incorrect prediction
b=final_titanic['prediction'].value_counts()
plt.figure(figsize=(6,8))
plt.pie(b, autopct='%1.1f%%',
        startangle=140,colors=['skyblue','black'])
plt.legend(b.index)
plt.title('Pie Chart for predictions from the model')
plt.show()
```

Pie Chart for predictions from the model



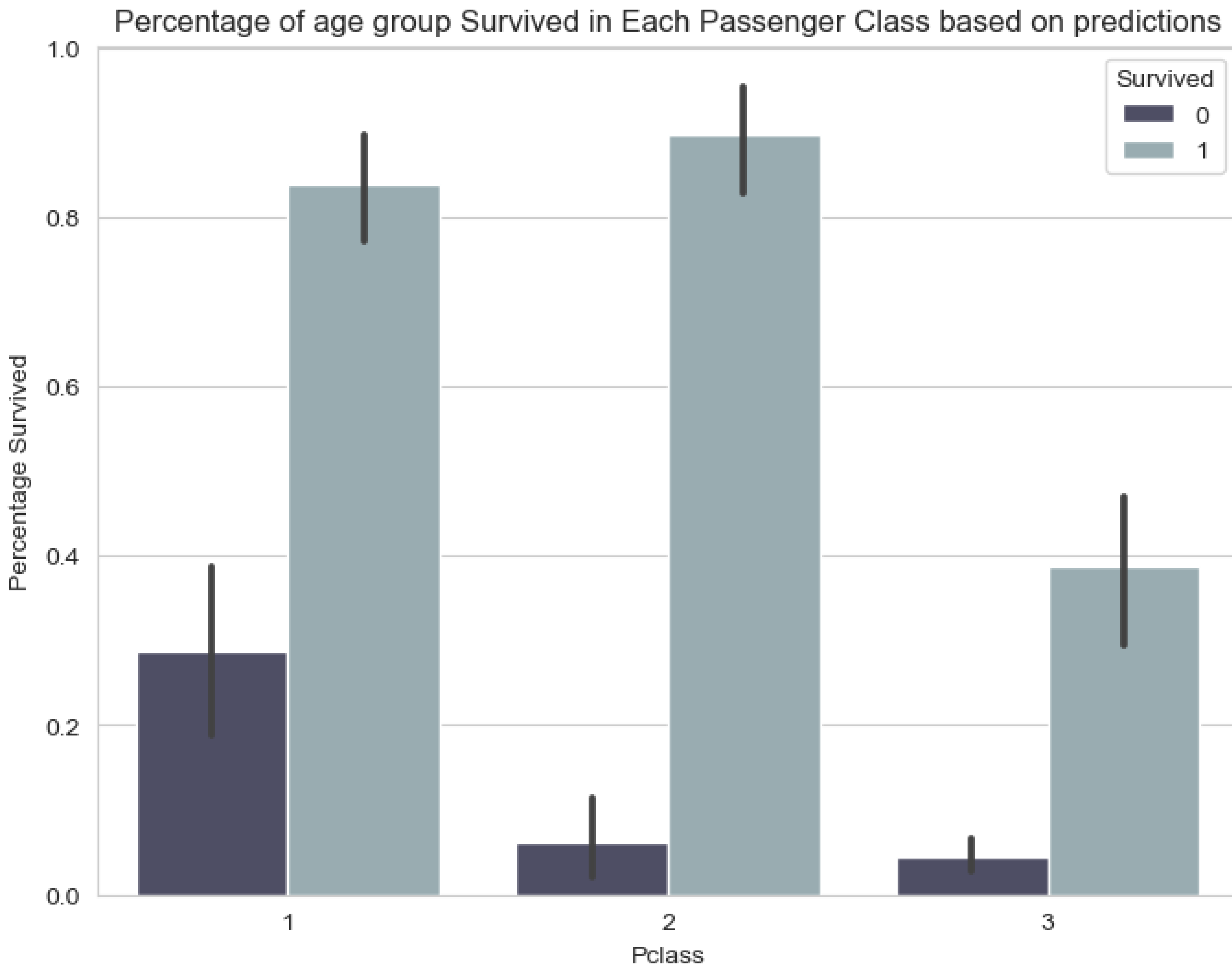
In [250…

```
# Assuming 'data' is your DataFrame containing passenger class, gender, and survival in
# Use the 'hue' parameter to group by gender within each passenger class
plt.figure(figsize=(8, 6))
sns.barplot(y='PREDICTED', x='Sex', hue='Survived', data=final_titanic, palette='bone')
plt.ylabel('Percentage Survived')
plt.title('Percentage of Men and Women Survived in Each Passenger Class based Sex')
plt.show()
```



In [251…

```
plt.figure(figsize=(8, 6))
sns.barplot(x='Pclass', y='PREDICTED', hue='Survived', data=final_titanic, palette='bone
plt.ylabel('Percentage Survived')
plt.title('Percentage of age group Survived in Each Passenger Class based on predictions
plt.show()
```



In [ ]:

conclusion

Only 38 pecentage of the passengers survived the accident, and among them 12 percentage were men and 26 percentage were females. The male passenger count was high as well as the death count.

The first class shows an appreciable survival comapred to other classes, but still it is not evident to conclude that the survival depends on pclass. But there is some relation between fare of the survived passengers(could be nobles) and class with the survival.

From the analysis, the females of any class in the titanic ship had a high survival rate compared to male passengers. It is difficult to predict exactly, if they followed a critierion to save the passengers.

thank you

In [ ]:

In [ ]: