In [ ]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input dir
ectory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

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

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier

from scipy.stats import boxcox
from sklearn.model_selection import StratifiedKFold
```

Exploring and Cleaning the dataframe

In [2]:
```
# this is not to skip some columns when showing the dataframe
pd.set_option('display.max_columns', None)

train = pd.read_csv('/kaggle/input/titanic/train.csv')
train.head(2)
```

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

In [3]:
```
test = pd.read_csv('/kaggle/input/titanic/test.csv')
test.head(2)
```

Out[3]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S

PassengerId: An unique index for passenger rows. It starts from 1 for first row and increments by 1 for every new rows.

Survived: Shows if the passenger survived or not. 1 stands for survived and 0 stands for not survived.

Pclass: Ticket class. 1 stands for First class ticket. 2 stands for Second class ticket. 3 stands for Third class ticket.

Name: Passenger's name. Name also contain title. "Mr" for man. "Mrs" for woman. "Miss" for girl. "Master" for boy.

Sex: Passenger's sex. It's either Male or Female.

Age: Passenger's age. "NaN" values in this column indicates that the age of that particular passenger has not been recorded.

SibSp: Number of siblings or spouses travelling with each passenger.

Parch: Number of parents of children travelling with each passenger.

Ticket: Ticket number.

Fare: How much money the passenger has paid for the travel journey.

Cabin: Cabin number of the passenger. "NaN" values in this column indicates that the cabin number of that particular passenger has not been recorded.

Embarked: Port from where the particular passenger was embarked/boarded.

```
In [4]: train.shape
```

```
Out[4]: (891, 12)
```

```
In [5]: test.shape
```

```
Out[5]: (418, 11)
```

In [6]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [7]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            332 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           417 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Checking out how many null values in both datasets

In [8]:
```
train.isnull().sum()
```

Out[8]:
```
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 [9]:
```
test.isnull().sum()
```

Out[9]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

It seems that Cabin is the most missing feature we have

The percentage of survivors in Train

In [10]:
```
# This code allocates the passengers who survived and those who didn't into 2 different variables.
survived = train[train['Survived'] == 1]
not_survived = train[train['Survived'] == 0]
# This code prints the number of survived and dead passengers along with the percentage.
print ("Survived: %i (%.1f%%)"%(len(survived), float(len(survived))/len(train)*100.0))
print ("Not Survived: %i (%.1f%%)"%(len(not_survived), float(len(not_survived))/len(train)*100.0))
print ("Total: %i"%len(train))
```

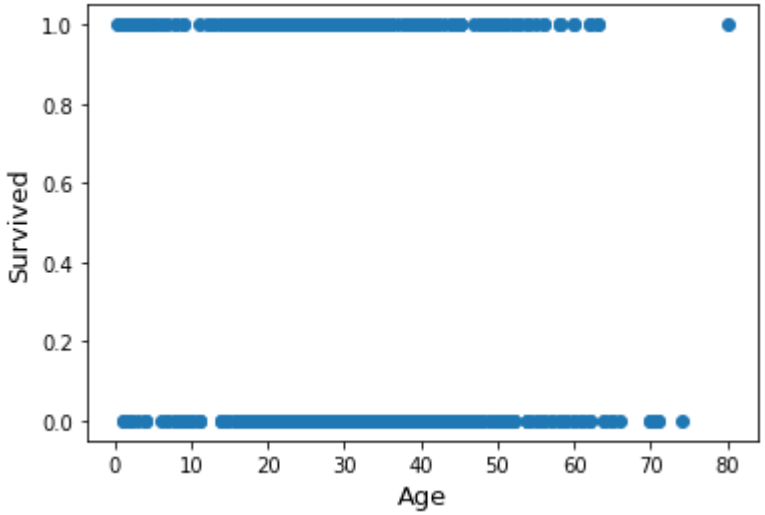
```
Survived: 342 (38.4%)
Not Survived: 549 (61.6%)
Total: 891
```

We have more Not Survived samples than Survived samples. This will affect how we sample data for training the model.

Detecting and removing outliers

Removing Age outlier

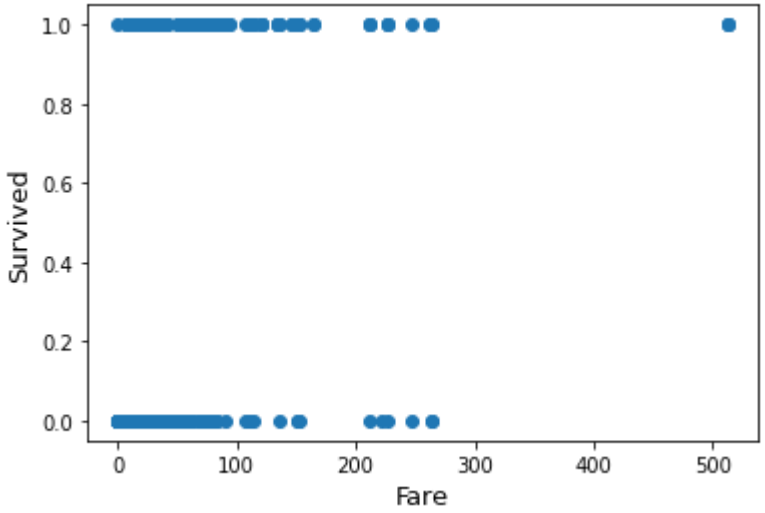
In [11]:
```
#we will use scater plot to see outlier ..
fig, ax = plt.subplots()
ax.scatter(x = train['Age'], y = train['Survived'])
plt.ylabel('Survived', fontsize=13)
plt.xlabel('Age', fontsize=13)
plt.show()
```



In [12]:
```
# dropping that one old grandma from the boat
train = train.drop(train[(train['Age']>79) & (train['Survived']>0.8)].index)
```

Removing Fare outlier

In [13]:
```
fig, ax = plt.subplots()
ax.scatter(x = train['Fare'], y = train['Survived'])
plt.ylabel('Survived', fontsize=13)
plt.xlabel('Fare', fontsize=13)
plt.show()
```



In [14]:
```
train = train.drop(train[(train['Fare']>400) & (train['Survived']>0.8)].index)
```

Merging dataframes

After we are done with cleaning the train data, now we merge both train and test to fill missing features and do feature engineering.

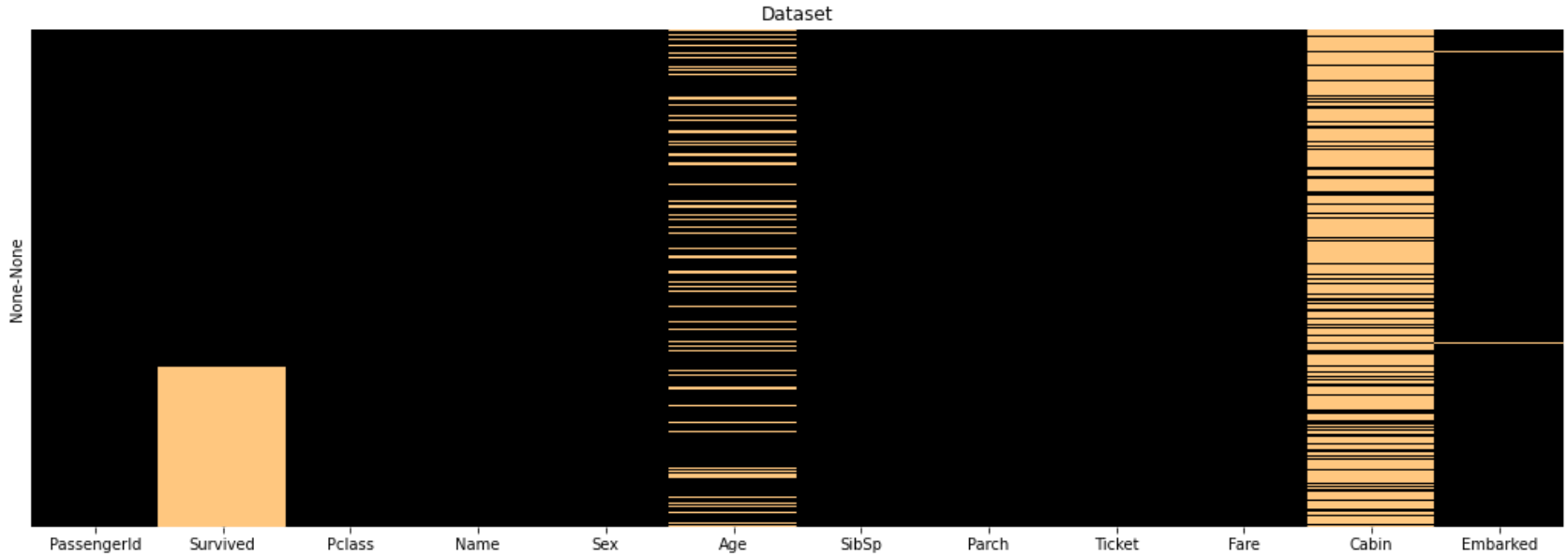
In [15]:
```
target = train['Survived']
#concatinating the two dataframes
whole_df = pd.concat([train,test],keys=[0,1], sort = False)
# Sanity check to ensure the total length matches
whole_df.shape[0] == train.shape[0]+test.shape[0]
```

Out[15]: True

Attempting to fill missing values

In [16]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (18, 6))

# This is a plot to show the missing values in the merged dataset
sns.heatmap(whole_df.isnull(), yticklabels=False, ax = ax, cbar=False, cmap='copper')
ax.set_title('Dataset')
```

Out[16]:
```
Text(0.5, 1.0, 'Dataset')
```



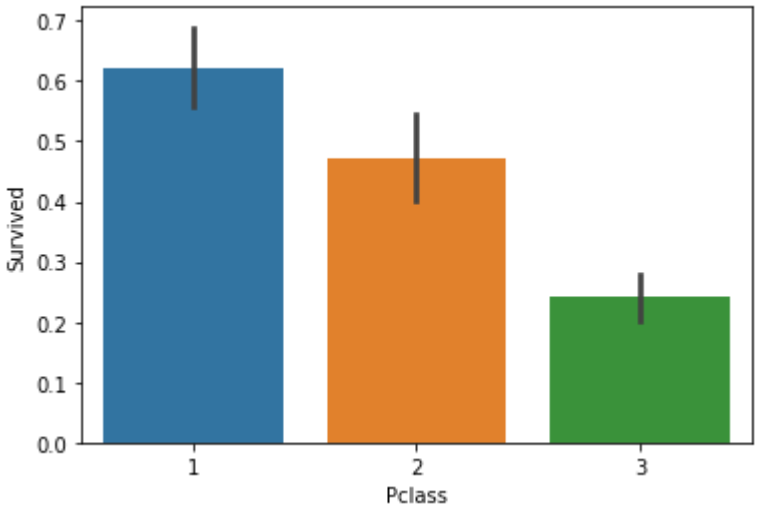
NOTE: missing values in Survived belongs to the test data as it doesnt have a label

Feature Exploration and Filling

Survival based on Class

In [17]:
```
sns.barplot(x='Pclass', y='Survived', data=whole_df)
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9920be4e0>
```

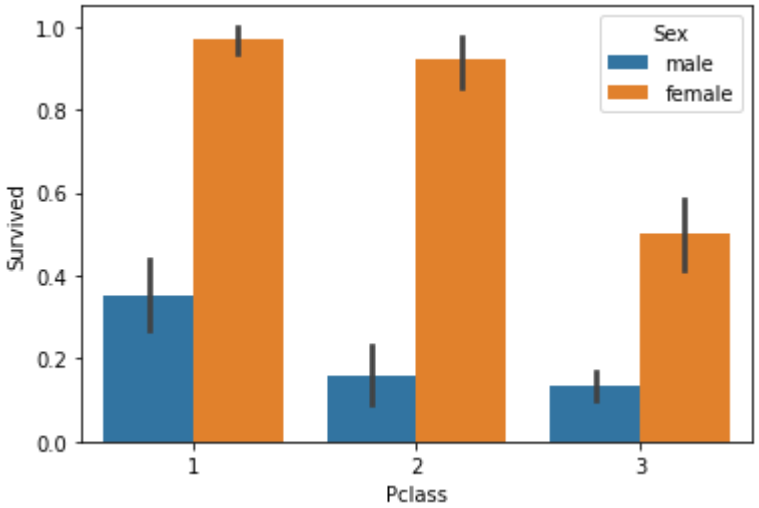


1. From the plot above, people who are in the first class have the highest survival probability compared to 2 and 3.

We can see a negative correlation between the label and Pclass

In [18]:
```
# This code shows the number of survived and dead passengers based on their class.
sns.barplot(x='Pclass',y='Survived',hue='Sex',data=whole_df,order=None,hue_order=None)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd99af96c50>



Above 80% of women in the first two classes (1,2) and overall, women have a higher chance to survive than men.

Since we have few missing enteries of the Fare price in test data, we can fill it with the average price depending on the class.

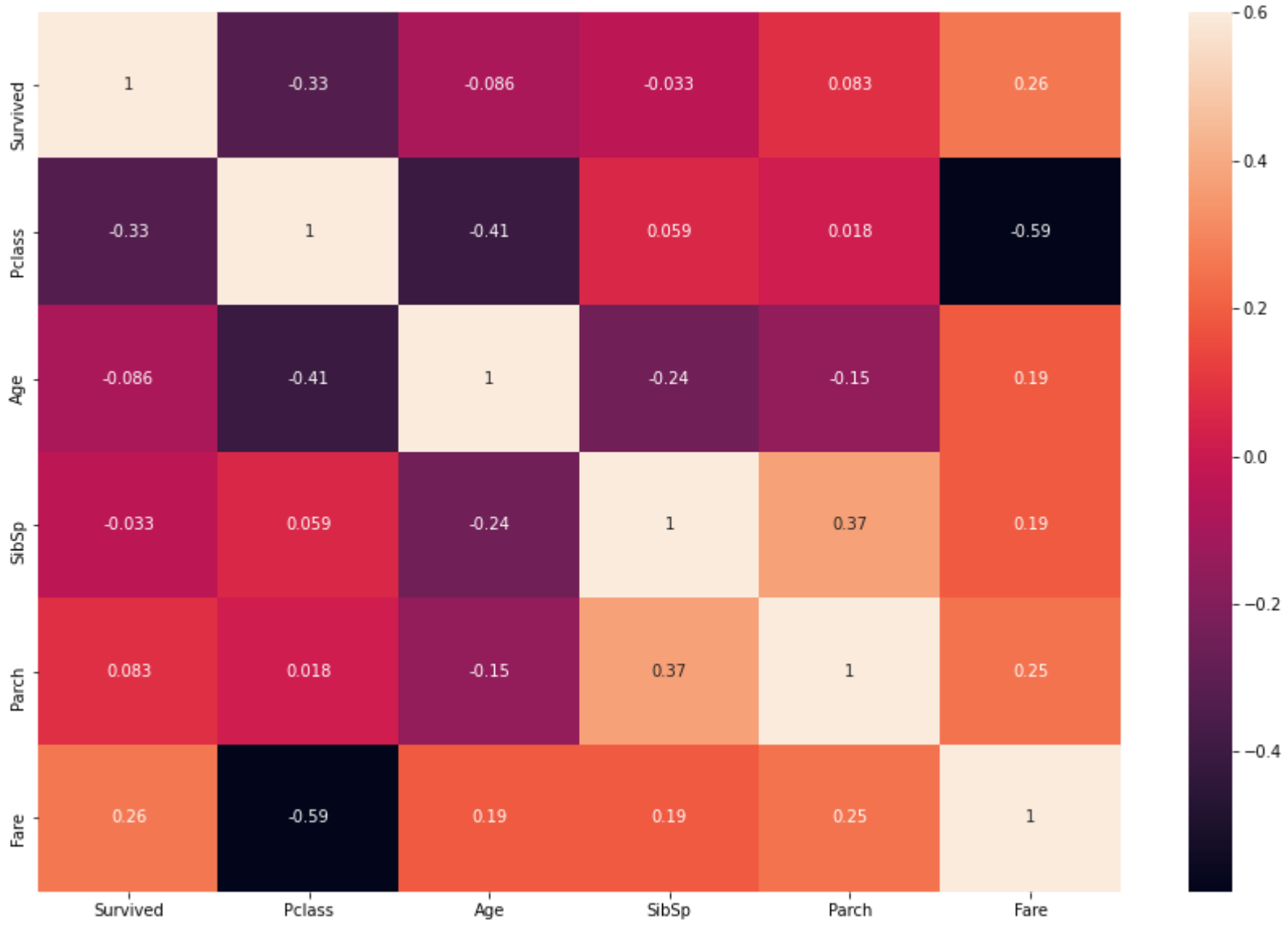
In [19]:
```
whole_df['Fare'].fillna(whole_df[whole_df['Pclass'] == 3]['Fare'].mean(), inplace=True)
```

Filling Age

To fill age, we are looking for features with correlations with Age, to use

In [20]:
```
plt.figure(figsize=(15,10))
sns.heatmap(whole_df.drop('PassengerId',axis=1).corr(), vmax=0.6, annot=True)
```

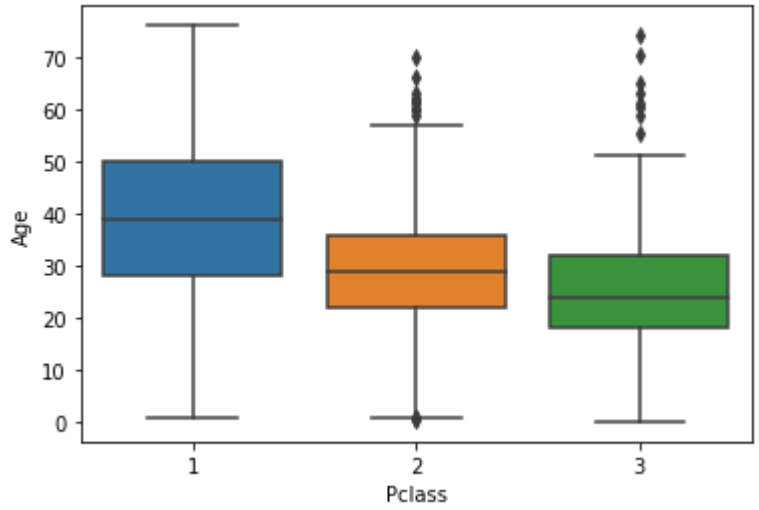
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9920770f0>



Here it seems the most correlated features are Pclass, SibSp, Parch and Fare

In [22]:
```
sns.boxplot(data=whole_df, y='Age', x='Pclass')
```

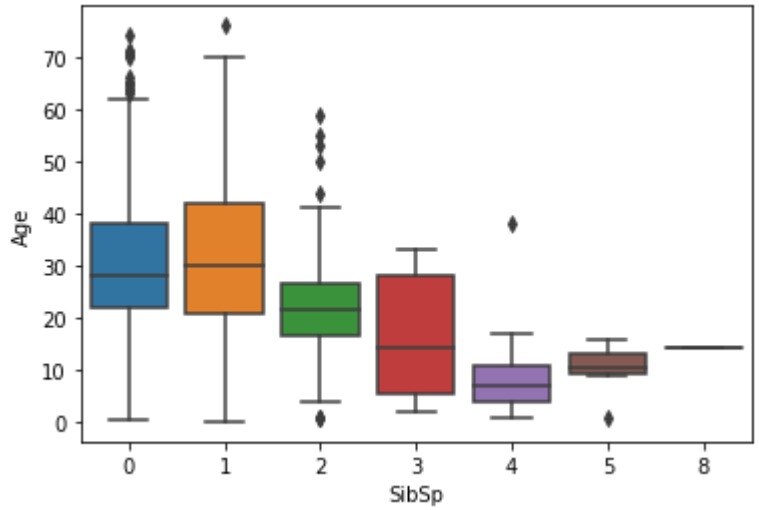
Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd991fa6828>
```



For Pclass and Age, the first class tend to have older people when compared to second class. and second class has older people when compared to third classs

In [23]:
```
sns.boxplot(data=whole_df, y='Age', x='SibSp')
```

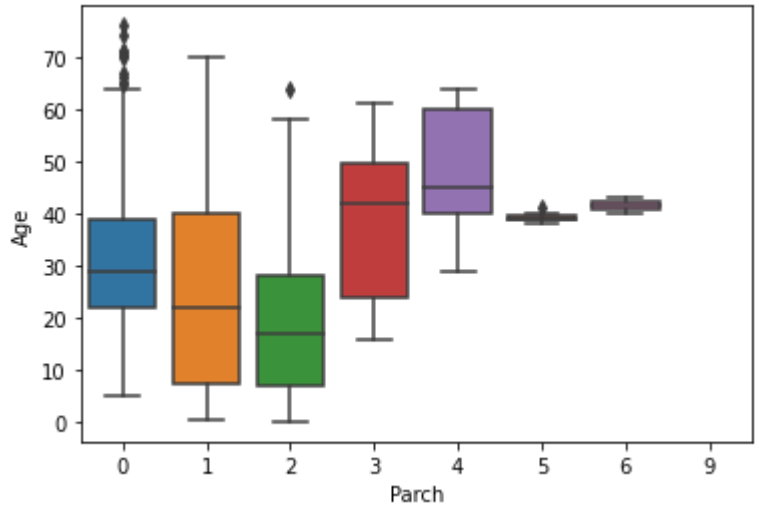
Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd991eeb5c0>
```



People onboard who has less siblings/spouse onboard tend to be older.

In [24]:
```
sns.boxplot(data=whole_df, y='Age', x='Parch')
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd99aed3278>
```



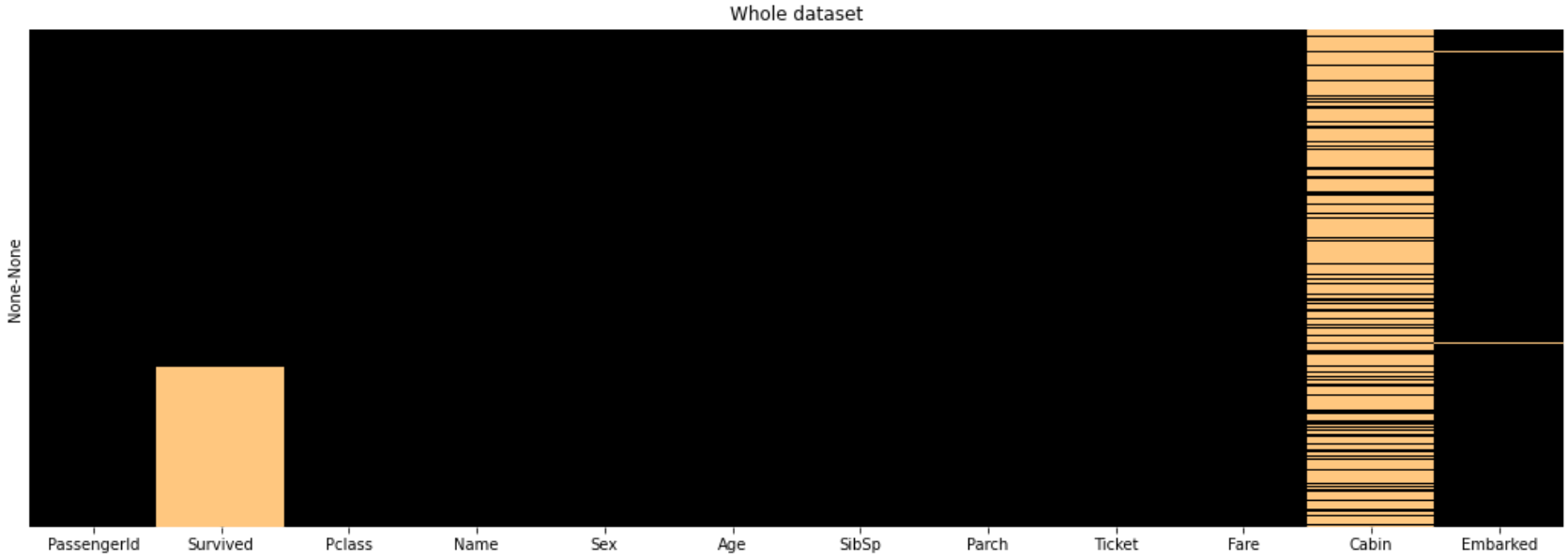
The older the traveller, the more probable that they will have more Parents/children onboard.

We will fill age by looking at the average age of people in the same class and have the same number of Siblings/Spouse and Parentch\children

In [25]:
```
def adjust_age(df,age_s):
    #convert age feature to a numpy array
    age_array = age_s.to_numpy()
    result = []

    # in case cannot find similar group we use default to fill
    default = int(df["Age"].median())
    # iterate every row
    for i,val in enumerate(age_array):
        # if empty
        if np.isnan(val):
            # get the median age of the group with similar attributes
            median_age = df[(df['Pclass']==df.iloc[i]['Pclass'])&(df['SibSp']==df.iloc[i]['SibSp'])&(df['Parc
h']==df.iloc[i]['Parch'])]['Age'].median()

            try:
                # age will be nan if no similar group found
                result.append(int(median_age))
            except:
                # append the default if no median_age found
                result.append(default)
        else:
            result.append(int(val))

    return result

# filling age using our function
whole_df['Age'] = adjust_age(whole_df,whole_df['Age'])
```

In [26]:
```
# Rechecking the condition of our dataset
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (18, 6))
sns.heatmap(whole_df.isnull(), yticklabels=False, ax = ax, cbar=False, cmap='copper')
ax.set_title('Whole dataset')
```

Out[26]: Text(0.5, 1.0, 'Whole dataset')



Filling Cabin

For This feature, we wanted to remove it due to the large amount of missing data in both train and test. However, we think people in the first class cabins might have a higher priority compared to second and third. Hence, we are filling empty values with 'NA'

In [27]:
```
whole_df['Cabin'].fillna(value="NA",inplace=True)
```

In [28]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (18, 6))

# train data
sns.heatmap(whole_df.isnull(), yticklabels=False, ax = ax, cbar=False, cmap='copper')
ax.set_title('Train data')
```

Out[28]: Text(0.5, 1.0, 'Train data')

Filling Embarked

In [29]:
```
train['Embarked'].fillna(train['Embarked'].mode()[0], inplace=True)
```

Feature engineering

here we make other features and turn existing ones in a way which we think it will help the model in making predictions

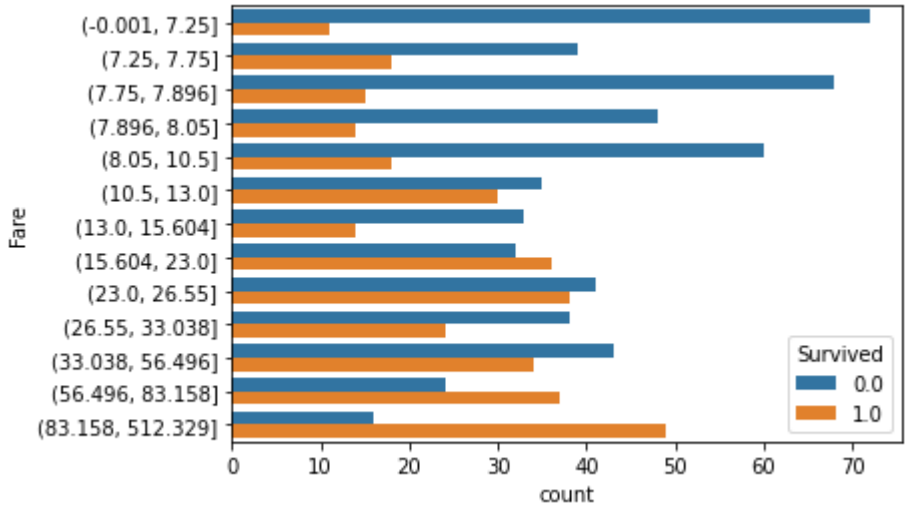
Fare

Converting the continuos range to categorical thru binning

In [30]:
```
whole_df['Fare'] = pd.qcut(whole_df['Fare'], 13)
```

In [31]:
```
sns.countplot(data=whole_df, y='Fare',hue='Survived')
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd991a2dc18>
```



important thing is to make the feature into limited unique values if it is categorical

It appears that the more the fare, the higher the survival probability as seen from the orange bar. On the other hand, the cheaper the fare, the more probable that a person will not survive.

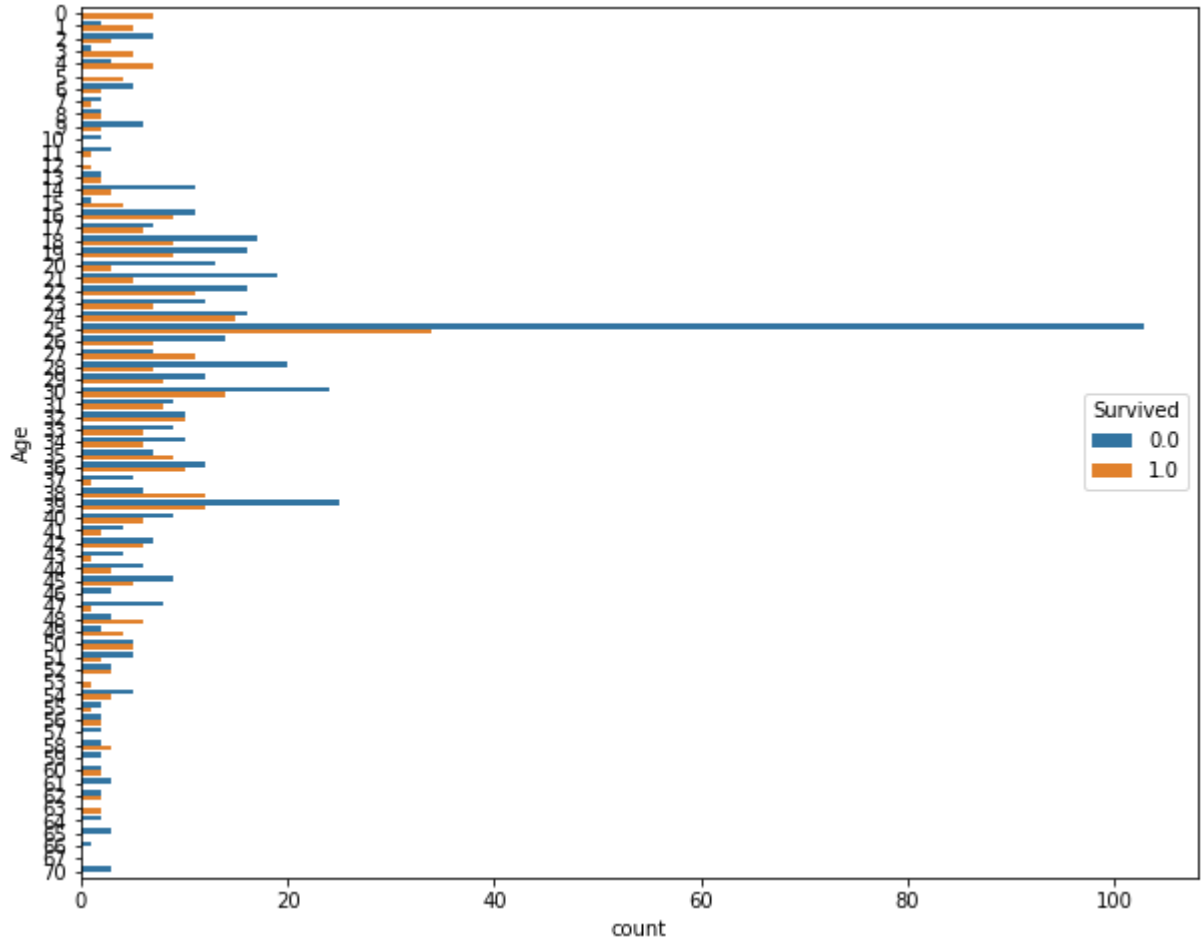
Age

For Age, it will be categorized as below

- infant: < 1 year
- Child: 1 - 14 years
- youth: 15 - 24 years
- adult: 25 - 63 years
- senior 64 and above

In [32]:
```
# This plots the count of people from different ages who survived and died
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10, 8))
sns.countplot(data=whole_df[(whole_df['Age'] >= 0)&(whole_df['Age'] <= 70)], y='Age', hue='Survived',ax=ax)
```

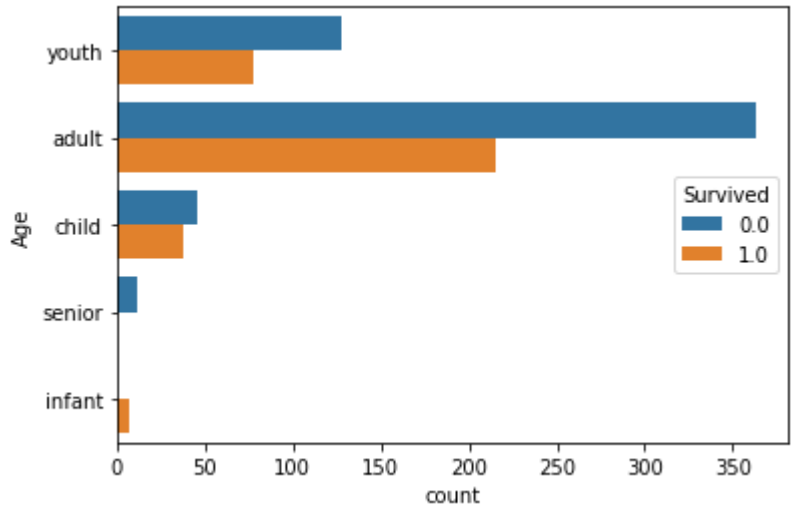
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9919d7828>



In [33]:
```
def categorize_age(val):
    if val < 1:
        return "infant"
    # Child case
    if val >= 1 and val <= 14:
        # classify as child
        return "child"
    #youth case
    elif val >= 15 and val <= 24:
        return "youth"
    # limited to 63 to keep seniors pure
    elif val >= 25 and val <= 63:
        return "adult"
    # seniors case (all dead)
    elif val >= 64:
        return "senior"
    print("Something is wrong!")
```

In [34]:
```
# applying the new function to age
whole_df['Age'] = whole_df['Age'].apply(categorize_age)
# plotting the survival of each age category
sns.countplot(data=whole_df, y='Age',hue='Survived')
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9919a6d30>



we can see that all infants survived while all seniors died. also, aduls have the worst chance of survival

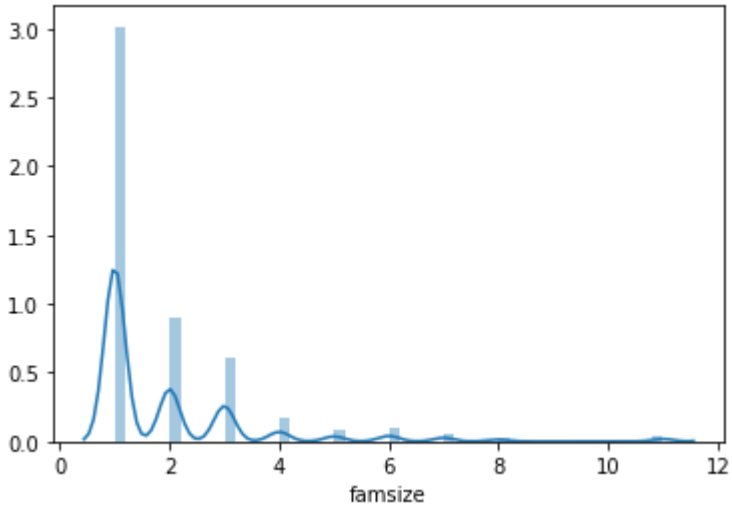
Parch + SibSp

We adding these two give us an estimate of the family size

In [35]:
```
# +1 is because im adding the person as well. 1 means alone
whole_df['famsize'] = whole_df['Parch']+whole_df['SibSp']+1
```

In [36]:
```
#whole_df['famsize'] = np.log1p(whole_df['famsize'])
sns.distplot(whole_df['famsize'])
whole_df['famsize'].skew()
```

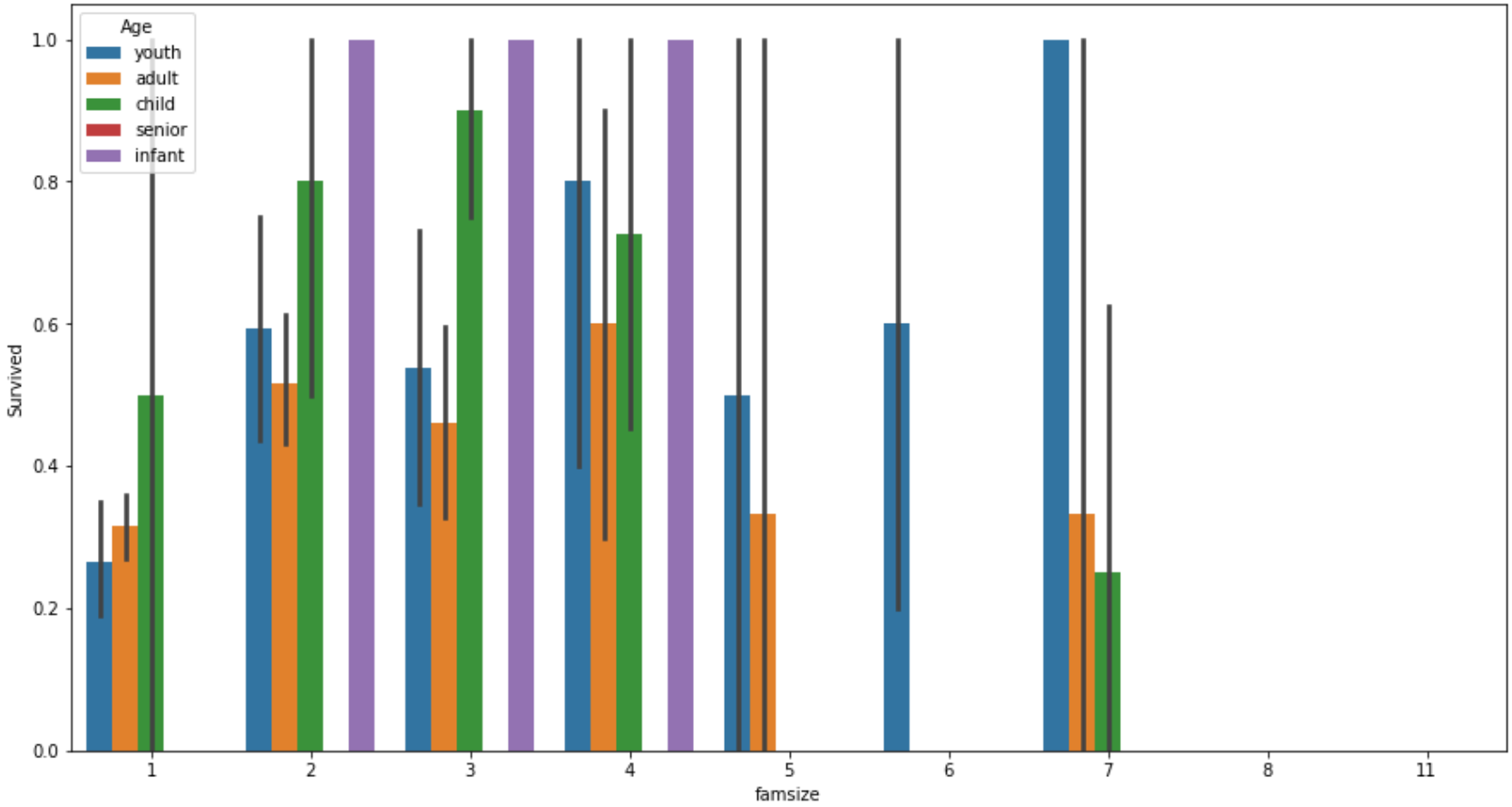
Out[36]: 2.8485226731871713



There are many people who dont have a close family member onboard

In [37]:
```
#sns.countplot(data=train, y='famsize',hue='Survived')
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (15, 8))
sns.barplot(data=whole_df, y='Survived',x='famsize',hue='Age',ax=ax)
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9916ef160>



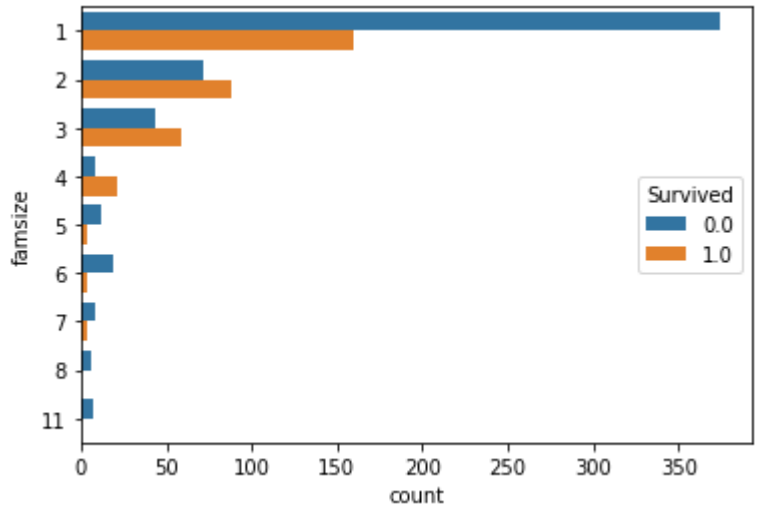
- The best chance of survival for children is for those who have a fammily size of 2 - 4
- infants and seniors survival rate are unaffected by family size
- youth survival rate tends to have a slight positive correlation with the increase of family size

Changing famsize feature to be categorical

Based on the distribution below, sizes with similar distributions will be grouped together

In [38]:
```
sns.countplot(data=whole_df, y='famsize',hue='Survived')
```

Out[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd991435518>
```



grouping sizes which have similar survival rates together

In [39]:
```
def classify_fam_size(famsize):
    # highest count and high death rate
    if famsize == 1:
        return "solo"
    # higher survival rate than death rate
    elif famsize in [2,3,4]:
        return "small-family"
    # noticably higher death rate compared to the last groups
    elif famsize in [5,6]:
        return "mid-family"
    elif famsize > 6:
        return "large-family"

whole_df['famsize'] = whole_df['famsize'].apply(classify_fam_size)
```

Cabin

Listing all unique values in Cabin

We can group them by the first letter

In [40]:
```
whole_df['Cabin'].unique()
```

Out[40]:
```
array(['NA', 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'B50', 'A26', 'D48', 'E58',
       'C126', 'B71', 'D49', 'B5', 'B20', 'F G63', 'C62 C64', 'E24',
       'C90', 'C45', 'E8', 'D45', 'C46', 'D30', 'E121', 'D11', 'E77',
       'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36', 'B102', 'B69', 'E49',
       'C47', 'D28', 'E17', 'A24', 'B51 B53 B55', 'C50', 'B42', 'C148',
       'B45', 'B36', 'A21', 'D34', 'A9', 'C31', 'B61', 'C53', 'D43',
       'C130', 'C132', 'C55 C57', 'C116', 'F', 'A29', 'C6', 'C28', 'C51',
       'C97', 'D22', 'B10', 'E45', 'E52', 'A11', 'B11', 'C80', 'C89',
       'F E46', 'B26', 'F E57', 'A18', 'E60', 'E39 E41', 'B52 B54 B56',
       'C39', 'B24', 'D40', 'D38', 'C105'], dtype=object)
```

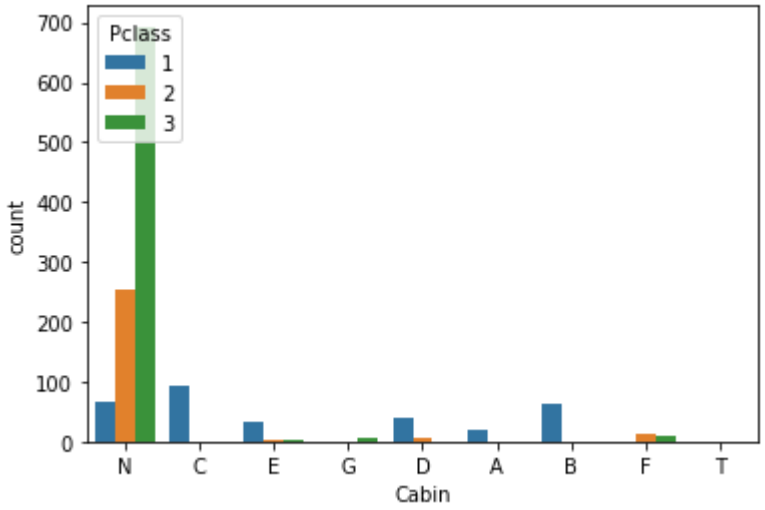
In [41]:
```
def take_section(code):
        return code[0]
whole_df['Cabin'] = whole_df['Cabin'].apply(take_section)
```

In [42]:
```
whole_df['Cabin'].unique()
```

Out[42]:
```
array(['N', 'C', 'E', 'G', 'D', 'A', 'B', 'F', 'T'], dtype=object)
```

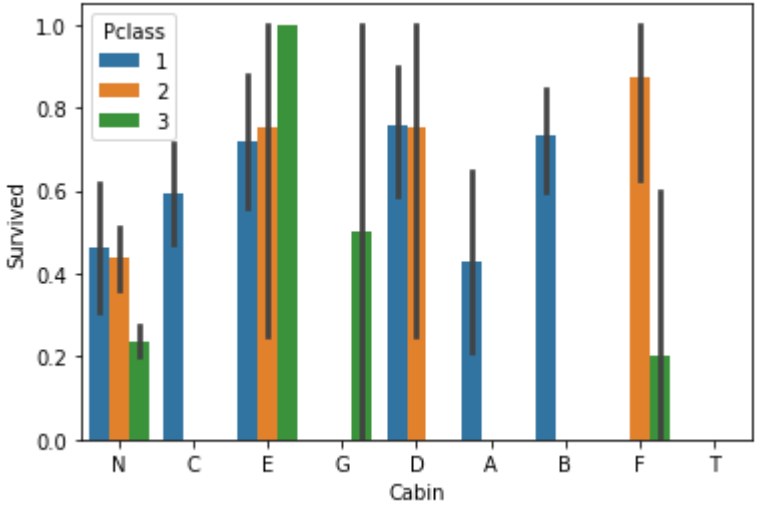
In [43]:
```
sns.countplot(data=whole_df, x='Cabin',hue='Pclass')
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9913bac18>



In [44]:
```
sns.barplot(data=whole_df, x='Cabin',y='Survived',hue='Pclass')
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9912fb9e8>

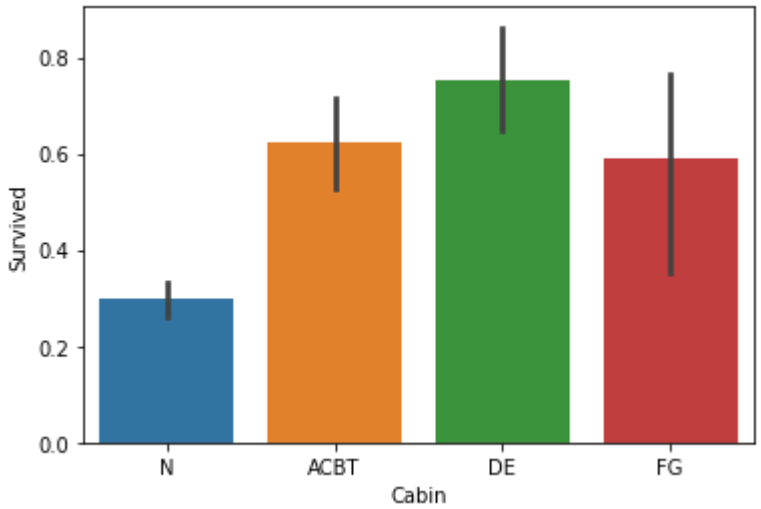


Note that Cabins with starting letter of C,A and B are purely contain first class travellers. Since E and D have similar survived/dead distributions and both counts are very small, we are grouping them together

In [45]:
```
#grouping cabins based on class
def group_cabin(code):
    # adding T because there is 1 person in T and has a class same as ppl
    # in A,C and B
    if code in ['A','C','B','T']:
        return "ACBT"
    elif code in ["F","G"]:
        return "FG"
    elif code in ['E','D']:
        return 'DE'
    else:
        return code

whole_df['Cabin'] = whole_df['Cabin'].apply(group_cabin)
```

In [46]:
```
sns.barplot(data=whole_df, x='Cabin',y='Survived')
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd991278470>
```



Names

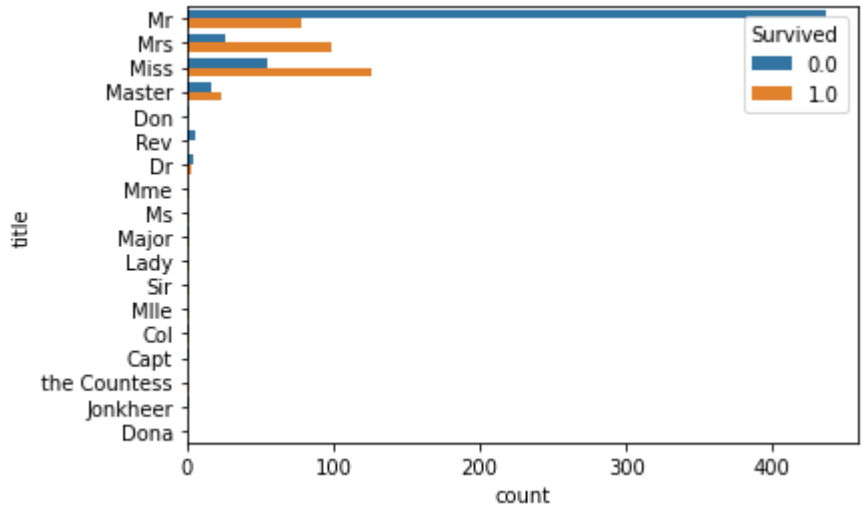
In [47]:
```
# Getting the title of each person
whole_df['title'] = whole_df['Name'].apply(lambda name: name.split(',')[1].split('.')[0].strip())
```

In [48]:
```
whole_df['title'].unique()
```

Out[48]:
```
array(['Mr', 'Mrs', 'Miss', 'Master', 'Don', 'Rev', 'Dr', 'Mme', 'Ms',
       'Major', 'Lady', 'Sir', 'Mlle', 'Col', 'Capt', 'the Countess',
       'Jonkheer', 'Dona'], dtype=object)
```

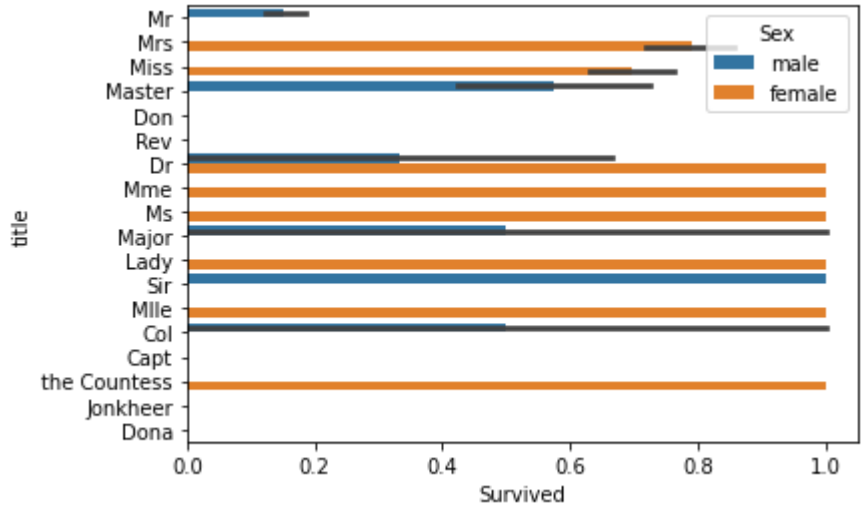
In [49]:
```
sns.countplot(data=whole_df, y='title',hue='Survived')
```

Out[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9911888d0>
```



In [50]:
```
sns.barplot(data=whole_df, y='title',x='Survived', hue='Sex')
```

Out[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd991ac5470>
```

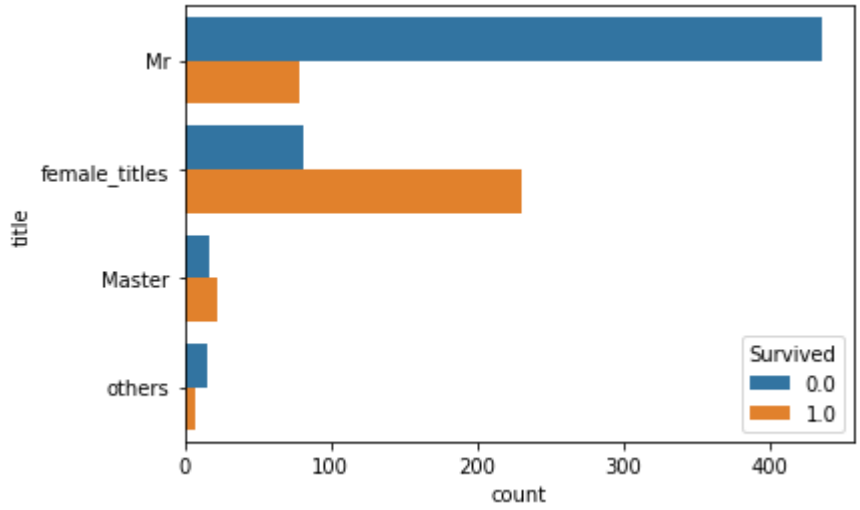


We will merge all Female titles together. However, for male titles we will keep Mr and Master separate as they have contradicting survival rates. we will group other titles in a group called other

In [51]:
```
def standardise_names(name):
    if name in ['Mrs','Miss','Ms', 'Lady','Mlle','the Countess','Dona']:
        return "female_titles"
    elif name in ['Mr','Master']:
        return name
    else:
        return 'others'
whole_df['title'] = whole_df['title'].apply(standardise_names)
```

In [52]:
```
sns.countplot(data=whole_df, y='title',hue='Survived')
```

Out[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9911787b8>
```



After grouping, Mr has the highest chance of death while female_titles have the highest chance of survival.

Using tickets to make ticket survival rate

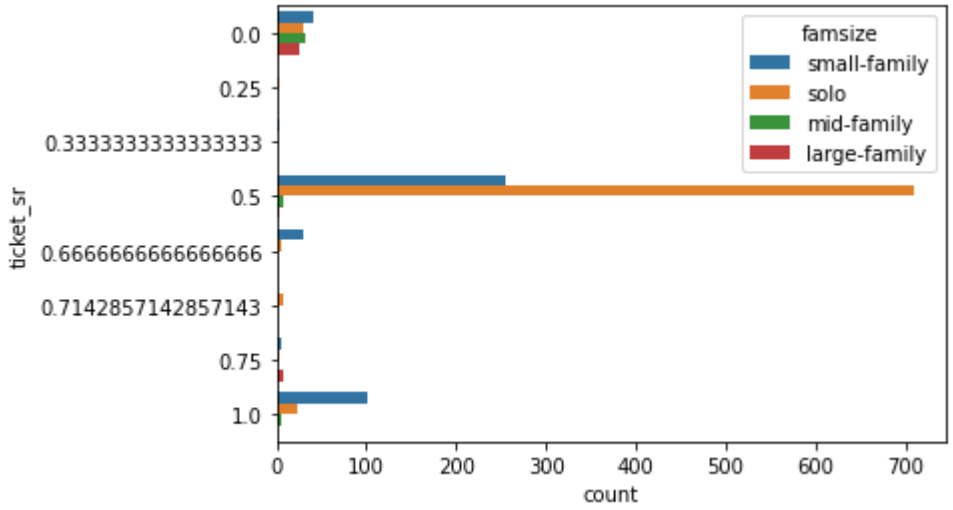
We are thinking that people who are close/friends/family will buy the ticket together. Therefore, in case of an accident they will try to help each other which results in either the majority of the group surviving or dying. Hence, if most of the group survived based on train dataset, we are assuming that the person in the test set (no label) will also survive given that they have the same ticket as the survivor group in training set.

In [53]:
```
tickets = {}
# grouping training set by ticket and finding the survival ratio
for ticket, df in whole_df.xs(0).groupby('Ticket'):
    # not putting high survival rate for solo ppl
    if df.shape[0]>1:
        tickets[ticket] = df['Survived'].sum()/ df.shape[0]
```

In [54]:
```
# setting the calculated survival rate based on the ticket
# if ticket not precalculated then set 0.5 (idk)
default = 0.5
whole_df['ticket_sr'] = whole_df['Ticket']
whole_df['ticket_sr'] = whole_df['ticket_sr'].apply(lambda x: tickets[x] if x in tickets.keys() else default)
```

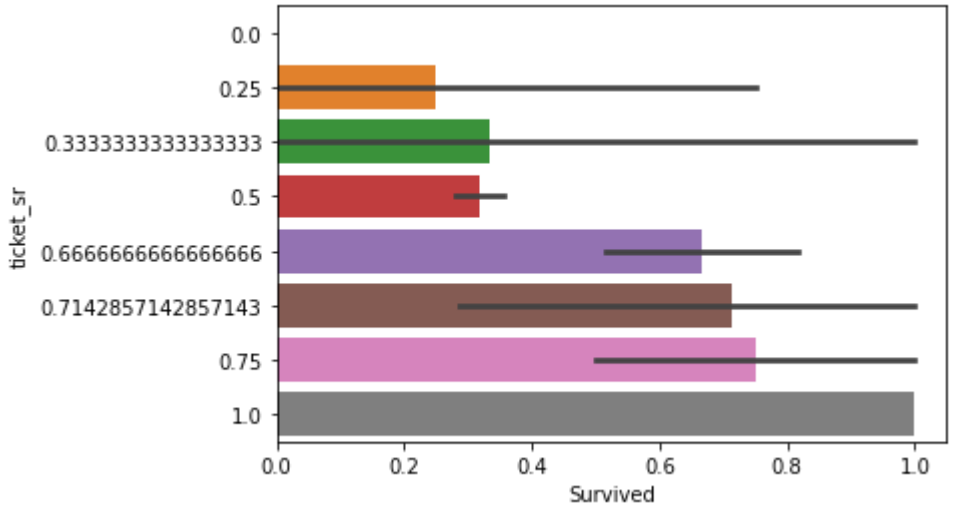
In [55]:
```
sns.countplot(data=whole_df, hue='famsize',y='ticket_sr',orient='h')
```

Out[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd99106a4e0>
```



In [56]:
```
sns.barplot(data=whole_df, y='ticket_sr',x='Survived',orient='h')
```

Out[56]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd990f39470>
```



We can see that the less the ticket survival rate is, the less chance of survival the ticket holder has

Removing uncessesary Features and Dummify

In [57]:
```
whole_df.drop(labels= ['Name','Ticket','SibSp','Parch'], axis=1,inplace=True)
whole_df = pd.get_dummies(whole_df,columns=['Pclass','Sex',"Age",'Fare','Cabin','Embarked','title','famsize'
],drop_first=True)
```

In [58]:
```
# global setting to show all the features
pd.set_option('display.max_columns', None)
whole_df.head(3)
```

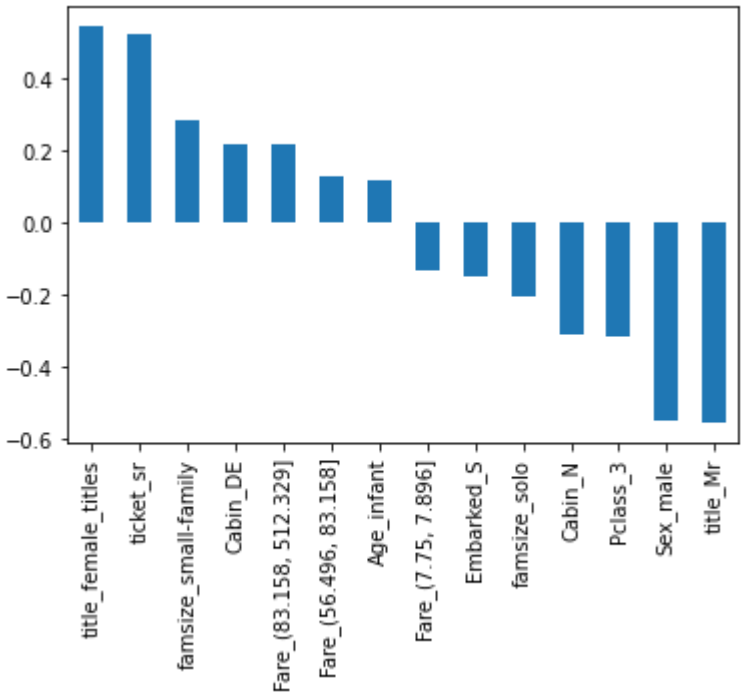
Out[58]:

		PassengerId	Survived	ticket_sr	Pclass_2	Pclass_3	Sex_male	Age_child	Age_infant	Age_senior	Age_youth	Fare_(7.25, 7.75]	Fare
0	**0**	1	0.0	0.5	0	1	1	0	0	0	1	0	
	1	2	1.0	0.5	0	0	0	0	0	0	0	0	
	2	3	1.0	0.5	0	1	0	0	0	0	0	0	

High correlation features in the training set

In [59]:
```
corr = whole_df.drop('PassengerId',axis=1).corr()
corr = corr[(corr['Survived'] > 0.1) | (corr['Survived'] < -0.1)]
corr['Survived'].sort_values(ascending=False).drop('Survived').plot(kind='bar')
```

Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd9911785c0>



now what you see here the benefit of all the above hard work. what we did is that we divided each feature into further subdivisions and then we divided them all using dummy variable and now we have the closest correlated specific sub divisions

These are the top features that correlate with the Survival rate. female titles, ticket survival rate, Mr title and sex male all play a drastic role in predicting the survival rate

Machine Learning prepping

This section is for

- separating the dataset into **X** (features) and **y** (label)
- split the data into training and testing set
  - research **optimal train to test ratio**
- Breaking down data into test and train
- feature selection

In [60]:
```
# separate the sets back
train,test = whole_df.xs(0),whole_df.xs(1)
```

In [61]:
```
# preparing X and target
y = train['Survived'].astype(int)
X = train.drop(labels=['Survived','PassengerId'], axis=1)
```

In [62]:
```
# This one was used during testing the different models but they were all removed to make the notebook cleane
r
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state=24)
```

In [63]:
```
# prepping the test set to submit to kaggle
X_final,IDs =  test.drop(labels=['PassengerId','Survived'], axis=1), test['PassengerId']
```

Machine Learning Experimentation

this section is for

- testing multiple machine learning algorithms on dataset
  - the goal is to **find best model**
  - value **test accuracy** over **train accuracy**
  - if test acc is way higher than train acc this means overfitting

In [64]:
```
# special libraries for ML
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score,StratifiedKFold, KFold
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier,BaggingClassifier,ExtraTreesClassifier,GradientBoostingClassi
fier,RandomForestClassifier
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
```

Getting features with high correlation

In [65]:
```
thresh = 0.1
corr = train.drop('PassengerId',axis=1).corr()
corr = corr[(corr['Survived'] > thresh) | (corr['Survived'] < -thresh)].drop('Survived')
selected_features = corr.sort_values('Survived').index
selected_features
```

Out[65]:
```
Index(['title_Mr', 'Sex_male', 'Pclass_3', 'Cabin_N', 'famsize_solo',
       'Embarked_S', 'Fare_(7.75, 7.896]', 'Age_infant',
       'Fare_(56.496, 83.158]', 'Fare_(83.158, 512.329]', 'Cabin_DE',
       'famsize_small-family', 'ticket_sr', 'title_female_titles'],
      dtype='object')
```

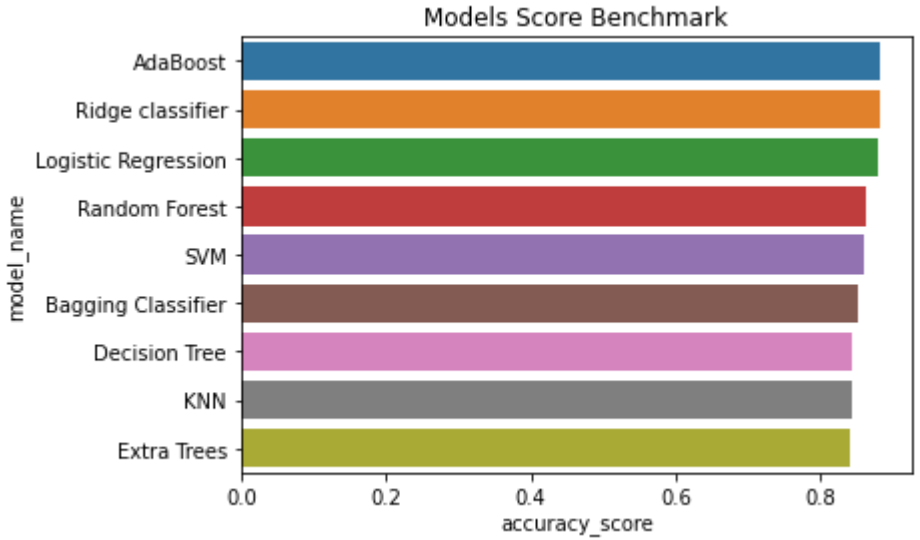
Models benchmark

In [66]:
```
# make visualization of the score of different models
def make_benchmark(X,y, scoring='accuracy', cv = 5):
    # using stratified because we have more deaths than survivors
    folds = StratifiedKFold(n_splits=cv)
    # different models for benchmark
    models = [SVC(gamma='auto'),
            DecisionTreeClassifier(),KNeighborsClassifier(),
            ExtraTreeClassifier(),LogisticRegression(max_iter=10000,solver='lbfgs'),
            RidgeClassifier(),AdaBoostClassifier(),
            BaggingClassifier(),RandomForestClassifier()]
    # names of the models
    names = ["SVM","Decision Tree","KNN","Extra Trees","Logistic Regression",
             "Ridge classifier","AdaBoost","Bagging Classifier","Random Forest"]
    # cross val mean score for each model
    scores = [np.mean(cross_val_score(model, X, y, cv=folds, scoring=scoring)) for model in models]


    return pd.DataFrame({"model_name":names,
                         "model": models,
                         scoring+"_score":scores})
```

In [67]:
```
# we will use this later on gridsearch
folds = StratifiedKFold(n_splits=5)
```

In [69]:
```
# make benchmark
bench = make_benchmark(X,y,cv=5)
#plot benchmark
sns.barplot(x='accuracy_score',y='model_name',data=bench.sort_values('accuracy_score',ascending=False)).set_t
itle("Models Score Benchmark")
```

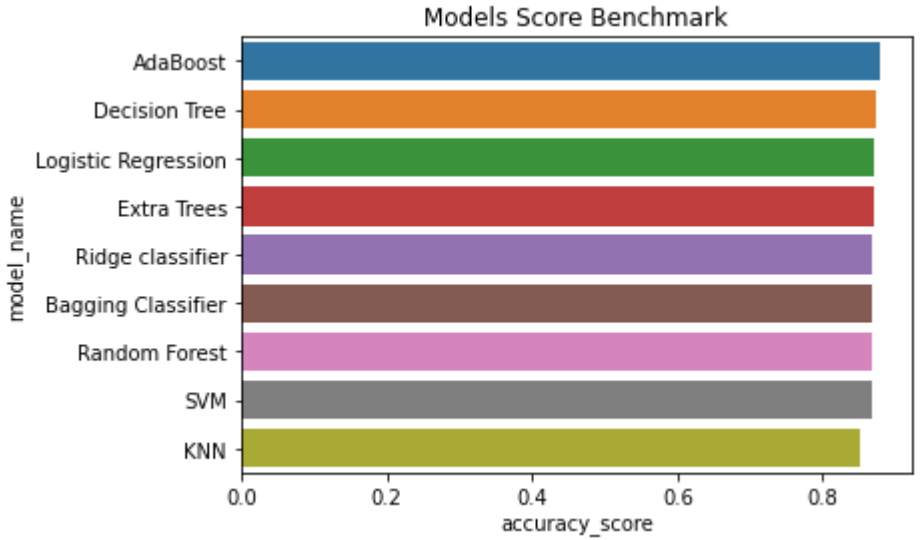
Out[69]: Text(0.5, 1.0, 'Models Score Benchmark')



do benchmark but using selected features

In [70]:
```
# do benchmark but using features selected previously (has corr > 0.1)

bench = make_benchmark(X[selected_features],y,cv=5)
sns.barplot(x='accuracy_score',y='model_name',data=bench.sort_values('accuracy_score',ascending=False)).set_t
itle("Models Score Benchmark")
```

Out[70]:
```
Text(0.5, 1.0, 'Models Score Benchmark')
```



These are the performance of different models out of the box (except logistic reg because it wasnt converging).

After a lot of trials of fitting diffirent models, we noticed that although many of them gives us a high cross validation score, they overfit the data as kaggle score is a lot lower than the CV score. The best model that acheived a decent score is Random Forest which has almost equal accuracy when fitted on the whole dataset or selected features

Model Optimization

this section is for

- experimenting with different hyperparameters of the model
  - the goal is to find a set of hyperparameters that yeild the best result

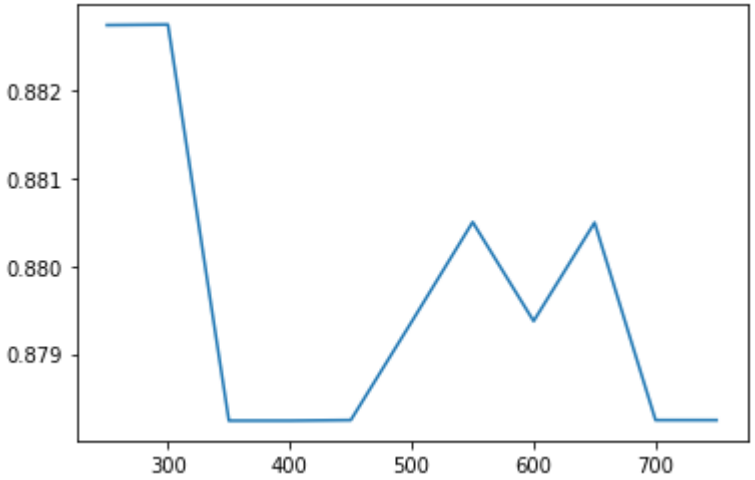
In [ ]:
```
## WARNING: this will take more than 10 minutes
## IF you still want to do so please uncomment the block

'''rf = RandomForestClassifier(random_state=24)
params = {'criterion':['gini','entropy'],
          'max_features':['auto'],
          'n_estimators':[150,200,300],
          'max_depth': [3,5,7,10,14],
          'class_weight':[ "balanced", "balanced_subsample"],
          'min_samples_split':[20,15,10]}

gsrf = GridSearchCV(rf , params,cv=KFold(5),verbose=2)
gsrf.fit(X,y)
gsrf.best_estimator_'''
```

Finding the effect of number of base estimators on accuracy

the model is the best model in gridsearch above

In [71]:
```
# different numbers of estimators effect on random forest
folds = StratifiedKFold(n_splits=5)
x_axis, y_axis = [],[]
for i in range(250,800,50):
    rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                                criterion='entropy', max_depth=7, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=15,
                                min_weight_fraction_leaf=0.0, n_estimators=i,
                                n_jobs=None, oob_score=False, random_state=24, verbose=0,
                                warm_start=False)

    y_axis.append(np.mean(cross_val_score(rf, X, y, cv=folds,scoring='accuracy')))
    x_axis.append(i)

sns.lineplot(x=x_axis, y=y_axis)
```

Out[71]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9919e4a58>
```



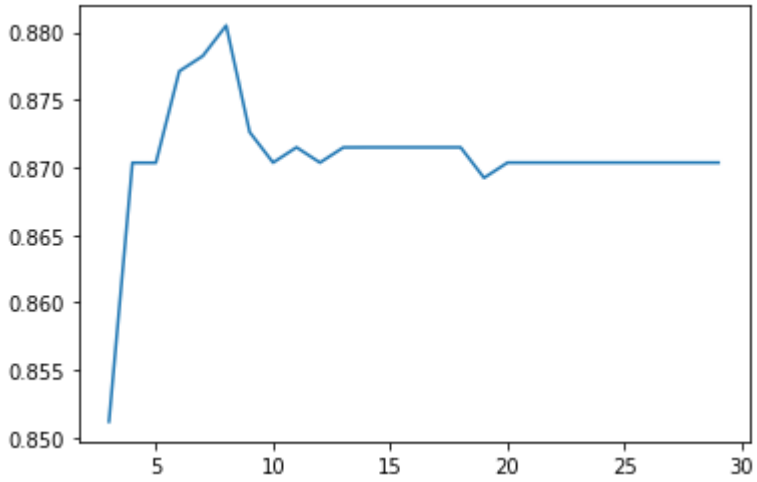
It seems like the lesser the depth the higher accuracy we get. However, we are affraid that this the RF is overfitting. Hence we went with 700

Finding the effect of RF depth on accuracy

Note: we stabalized the n_estimator of 700

In [72]:
```
#effect of  different depth on RandomForest

folds = StratifiedKFold(n_splits=5)
x_axis, y_axis = [],[]
for i in range(3,30,1):
    #our model
    rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                           criterion='entropy', max_depth=i, max_features='auto',
                           max_leaf_nodes=None, max_samples=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=15,
                           min_weight_fraction_leaf=0.0, n_estimators=700,
                           n_jobs=None, oob_score=False, random_state=24, verbose=0,
                           warm_start=False)
    # add score for plotting
    y_axis.append(np.mean(cross_val_score(rf, X, y, cv=folds,scoring='accuracy')))
    x_axis.append(i)

sns.lineplot(x=x_axis, y=y_axis)
```

Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd990ae1a90>



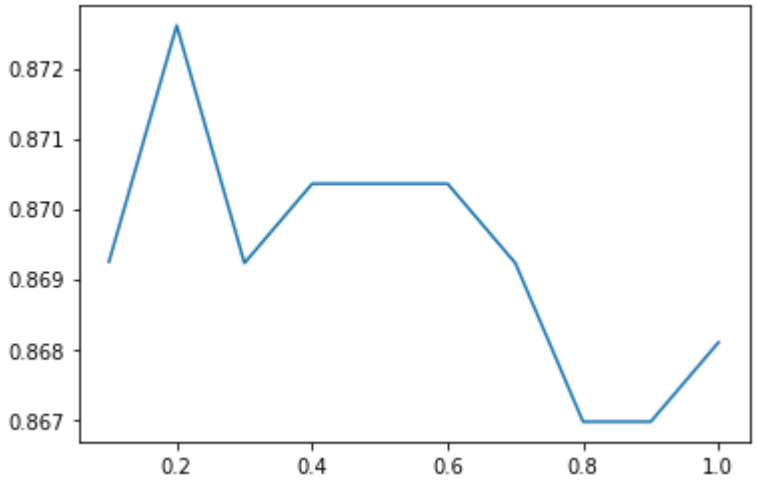
From the above graph we think trees in the forest wont expand more than depth of 20. The best depth they yeilds the best accuracy is 9

Finding the effect of max_features on accuracy

Note that we stabalized the n_estimator and depth

In [73]:
```
#effect of  different depth on RandomForest

folds = StratifiedKFold(n_splits=5)
x_axis, y_axis = [],[]
for i in np.linspace(0.1,1.,10):
    #our model
    rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                           criterion='entropy', max_depth=9, max_features=i,
                           max_leaf_nodes=None, max_samples=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=15,
                           min_weight_fraction_leaf=0.0, n_estimators=700,
                           n_jobs=None, oob_score=False, random_state=24, verbose=0,
                           warm_start=False)
    # add score for plotting
    y_axis.append(np.mean(cross_val_score(rf, X, y, cv=folds,scoring='accuracy')))
    x_axis.append(i)

sns.lineplot(x=x_axis, y=y_axis)
```

Out[73]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9909c4b00>
```



taking 20% of features to consider when looking for the best split yeilds the best accuracy while taking 80% yeilds us the lowest. we will use 80% as we are afraid of overfitting

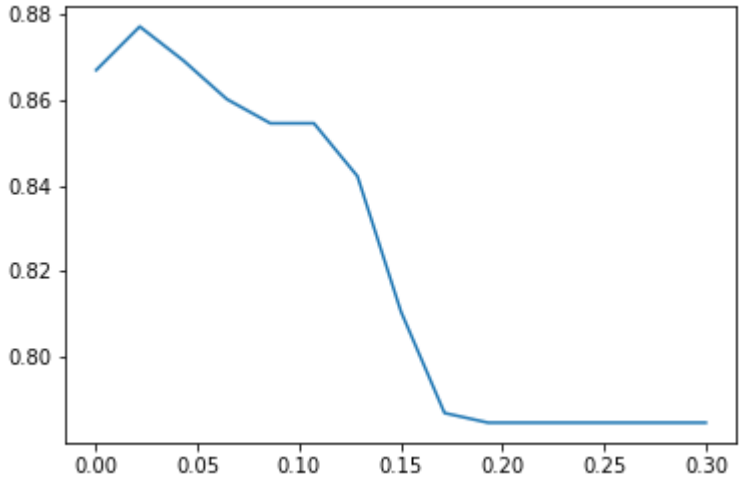
Finding the effect of CCPAlpha on accuracy (Pruning)

stabalized hyperparameters:

- n_estimators
- depth
- max features

In [74]:
```
folds = StratifiedKFold(n_splits=5)
x_axis, y_axis = [],[]
for i in np.linspace(0.0001,0.3,15):
    #our model
    rf = RandomForestClassifier(bootstrap=True, ccp_alpha=i, class_weight='balanced',
                            criterion='entropy', max_depth=9, max_features=.8,
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=15,
                            min_weight_fraction_leaf=0.0, n_estimators=700,
                            n_jobs=None, oob_score=False, random_state=24, verbose=0,
                            warm_start=False)
    # add score for plotting
    y_axis.append(np.mean(cross_val_score(rf, X, y, cv=folds,scoring='accuracy')))
    x_axis.append(i)

sns.lineplot(x=x_axis, y=y_axis)
```

Out[74]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9909bf908>
```



It seems like intruducing a litle bit of pruning helps increase accuracy, then the accuracy drops down with the increase of pruning

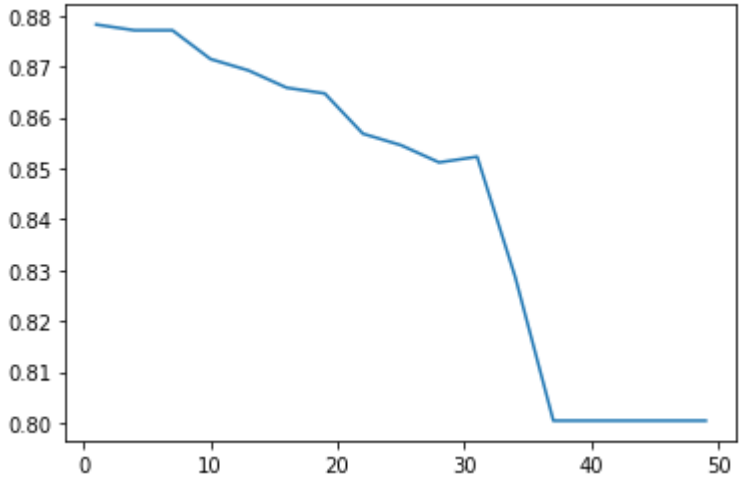
Finding the effect of number of minimum samples to make a leaf on accuracy

stabalized hyperparameters:

- n_estimators
- depth
- max features
- pruning (CCPAlpha)

In [75]:
```
folds = StratifiedKFold(n_splits=5)
x_axis, y_axis = [],[]
for i in range(1,50,3):
    #our model
    rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.02, class_weight='balanced',
                            criterion='entropy', max_depth=9, max_features=.8,
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=i, min_samples_split=15,
                            min_weight_fraction_leaf=0.0, n_estimators=700,
                            n_jobs=None, oob_score=False, random_state=24, verbose=0,
                            warm_start=False)
    # add score for plotting
    y_axis.append(np.mean(cross_val_score(rf, X, y, cv=folds,scoring='accuracy')))
    x_axis.append(i)

sns.lineplot(x=x_axis, y=y_axis)
```

Out[75]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9908ccc50>
```



It seems that the more we increased the number of samples required to make a leaf branch, the worse the accuracy gets. However, we cannot conclude the result as we think it might be solving the overfitting issue.

Finalizing random forest

In [76]:
```
rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.00,
                        class_weight='balanced_subsample', criterion='gini',
                        max_depth=9, max_features=0.2, min_samples_leaf=4,
                        min_samples_split=15,n_estimators=700,
                        random_state=24, verbose=0)
```

In [77]:
```
np.mean(cross_val_score(rf, X, y, cv=folds, scoring='accuracy'))
```

Out[77]:
```
0.8737319875579255
```

In [78]:
```
rf.fit(X,y)
```

Out[78]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                        class_weight='balanced_subsample', criterion='gini',
                        max_depth=9, max_features=0.2, max_leaf_nodes=None,
                        max_samples=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=4,
                        min_samples_split=15, min_weight_fraction_leaf=0.0,
                        n_estimators=700, n_jobs=None, oob_score=False,
                        random_state=24, verbose=0, warm_start=False)
```

In [79]:
```
y_hat = rf.predict(X_final)
```

In [80]:
```
submission_df = pd.DataFrame({"PassengerId":IDs,
                              "Survived": y_hat})
```

In [82]:
```
submission_df.to_csv('submit.csv',index=False)
```