In [272]:

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

In [273]:

```
train_data_raw = pd.read_csv('train.csv')
test_data_raw = pd.read_csv('test.csv')
```

In [274]:

```
train_data_raw.sample(5)
```

Out[274]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
118	119	0	1	Baxter, Mr. Quigg Edmond	male	24.0	0	1	PC 17558	247.5208	B58 B60	C
695	696	0	2	Chapman, Mr. Charles Henry	male	52.0	0	0	248731	13.5000	NaN	S
584	585	0	3	Paulner, Mr. Uscher	male	NaN	0	0	3411	8.7125	NaN	C
745	746	0	1	Crosby, Capt. Edward Gifford	male	70.0	1	1	WE/P 5735	71.0000	B22	S
347	348	1	3	Davison, Mrs. Thomas Henry (Mary E Finck)	female	NaN	1	0	386525	16.1000	NaN	S

In [275]:

```
columns_all = train_data_raw.columns
columns_all
```

Out[275]:

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

Exploratory Data Analysis

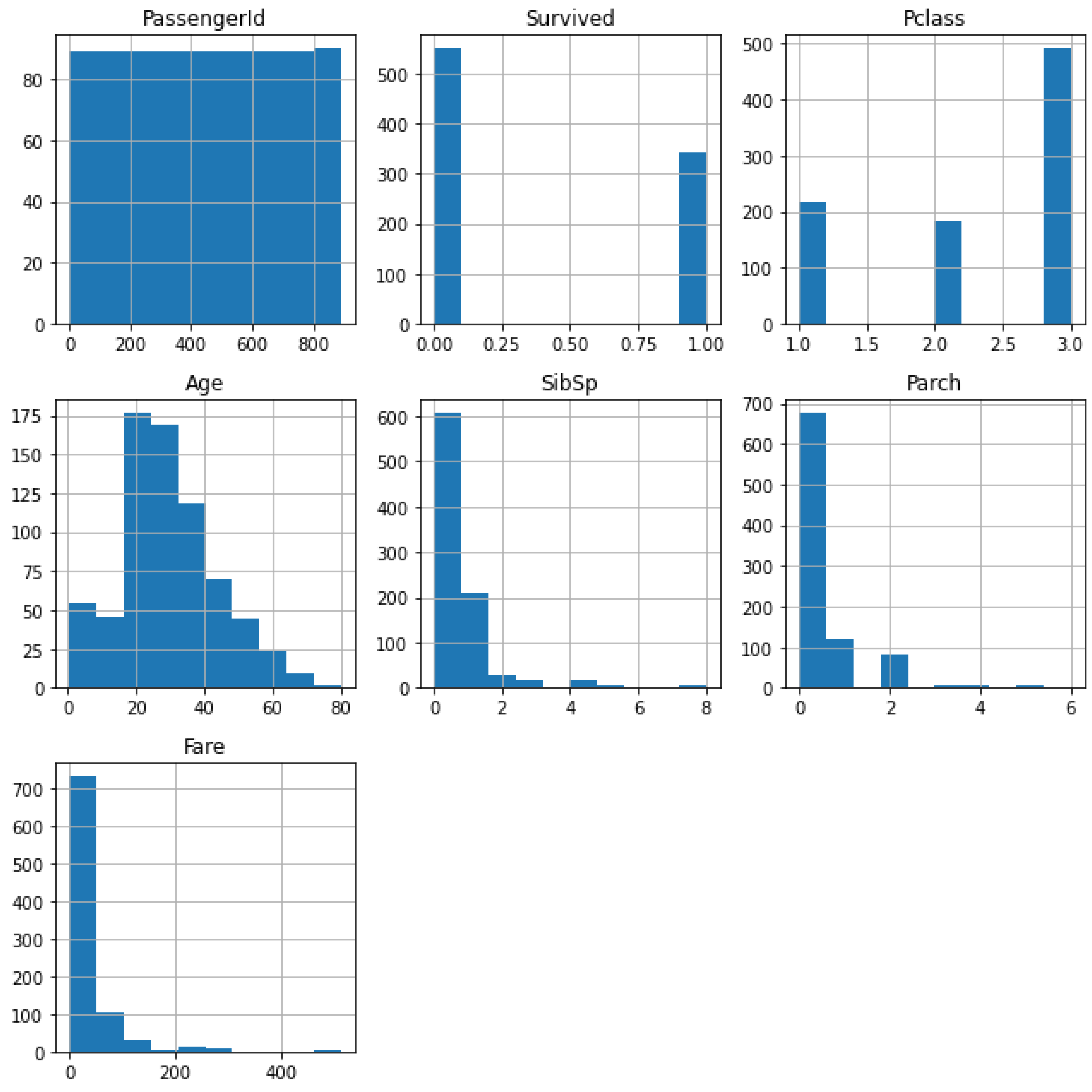
In [276]:

```
train_data_raw.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
```

- There are 891 records and there are missing values in some of the columns.

```
In [277]: train_data_raw.hist(figsize=(9,9))
          plt.tight_layout()
```



```
In [278]: train_data_raw['Survived'].value_counts().apply(lambda x:f'{x} ({x*100/len(train_data_raw):0.2f}%)')

Out[278]: 0    549 (61.62%)
          1    342 (38.38%)
          Name: Survived, dtype: object
```

Initial Inferences :

- The dataset is mildly imbalanced.
- The columns 'PassengerId' & 'Name' are unique identifiers.
- 'Survived' is the target column that we have to predict.
- The columns 'Pclass', 'Sex' and 'Embarked' are categorical columns and the rest are numerical.
- The column 'SibSp' should ideally be integer value.

We shall split our training data to train-test set before proceeding further to avoid any data leakage into test set.

In [279]:
```
from sklearn.model_selection import train_test_split
```

We shall create a copy of train_set so as to not loose the original training set during feature engineering.

In [280]:
```
train_set,test_set = train_test_split(train_data_raw,test_size=0.2,stratify=train_data_raw['Survived'],random_st
```

In [281]:
```
train_original = train_set.copy()
train_set.reset_index(drop=True,inplace=True)
```

In [282]:
```
train_set.sample(5)
```

Out[282]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
422	493	0	1	Molson, Mr. Harry Markland	male	55.0	0	0	113787	30.5000	C30	S
413	811	0	3	Alexander, Mr. William	male	26.0	0	0	3474	7.8875	NaN	S
637	535	0	3	Cacic, Miss. Marija	female	30.0	0	0	315084	8.6625	NaN	S
170	136	0	2	Richard, Mr. Emile	male	23.0	0	0	SC/PARIS 2133	15.0458	NaN	C
391	254	0	3	Lobb, Mr. William Arthur	male	30.0	1	0	A/5. 3336	16.1000	NaN	S

In [283]:
```
train_set.describe()
```

Out[283]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	712.000000	712.000000	712.000000	572.000000	712.000000	712.000000	712.000000
mean	444.730337	0.383427	2.307584	29.806678	0.485955	0.376404	31.756120
std	259.308184	0.486563	0.831550	14.836519	1.025593	0.769609	48.467739
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	218.750000	0.000000	2.000000	20.375000	0.000000	0.000000	7.895800
50%	443.500000	0.000000	3.000000	28.000000	0.000000	0.000000	13.931250
75%	668.250000	1.000000	3.000000	39.000000	1.000000	0.000000	30.500000
max	890.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

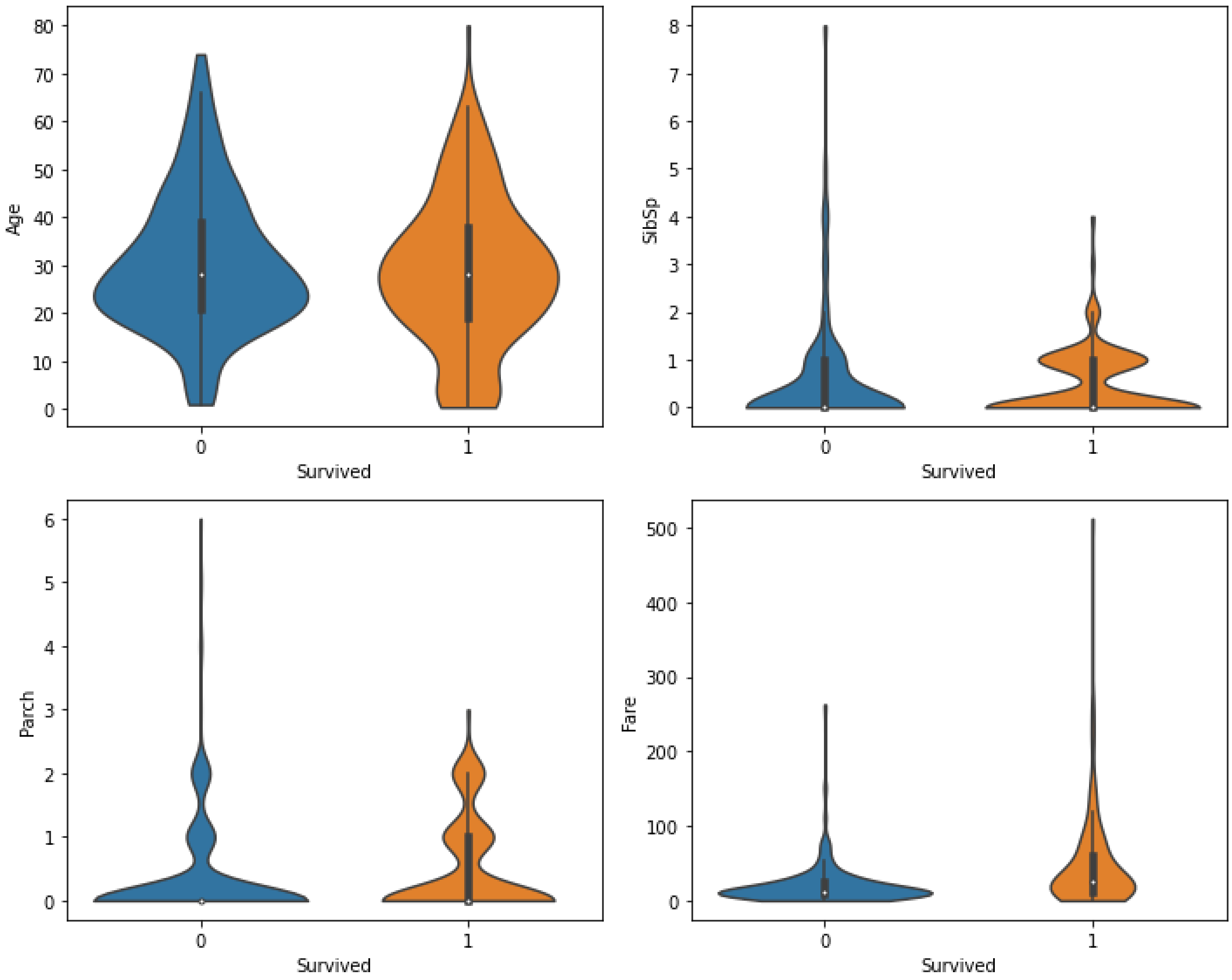
In [284]:
```
# No. of unique elements in each column
train_set.apply(lambda x: x.nunique())
```

Out[284]:
```
PassengerId    712
Survived         2
Pclass           3
Name           712
Sex              2
Age             87
SibSp            7
Parch            7
Ticket         569
Fare           226
Cabin          119
Embarked         3
dtype: int64
```

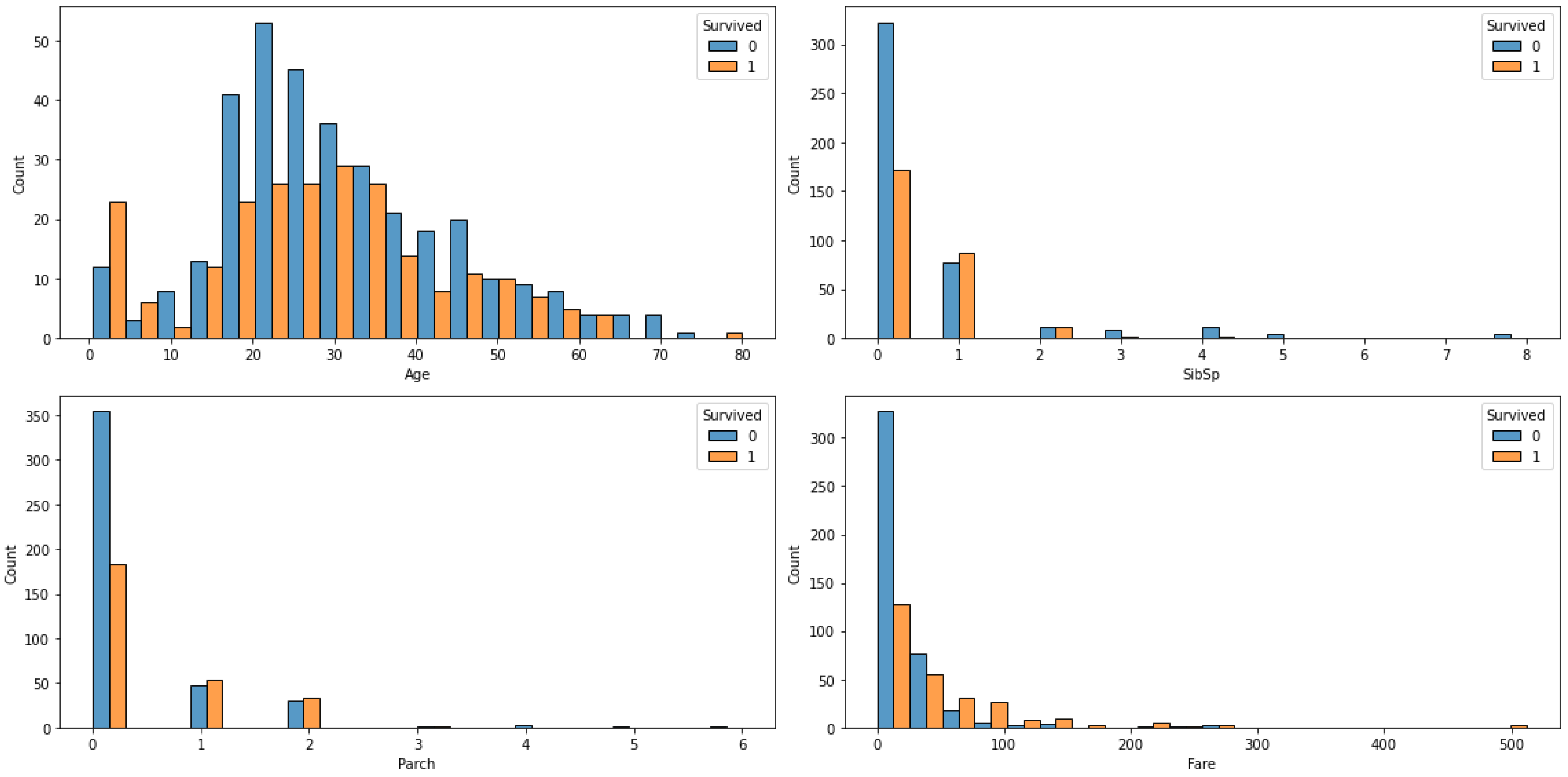
In [285]:
```
num_cols = train_set.select_dtypes('number').columns.drop(['PassengerId','Survived','Pclass']).to_numpy()
cat_cols = list(train_set.select_dtypes('object').columns.drop(['Name']))
cat_cols.extend(['Pclass'])
print("Numerical Columns : ",num_cols)
print("Categorical Columns : ",cat_cols)
```

```
Numerical Columns :  ['Age' 'SibSp' 'Parch' 'Fare']
Categorical Columns :  ['Sex', 'Ticket', 'Cabin', 'Embarked', 'Pclass']
```

```
In [286]: n_def_num_cols = len(num_cols)
          fig,ax = plt.subplots(round(n_def_num_cols/2),2,figsize=(10,n_def_num_cols*2))
          for i,col in enumerate(num_cols):
            sns.violinplot(x='Survived',y=col,data=train_set,ax=ax.ravel()[i],orient='v',cut=0)
          fig.tight_layout()
```

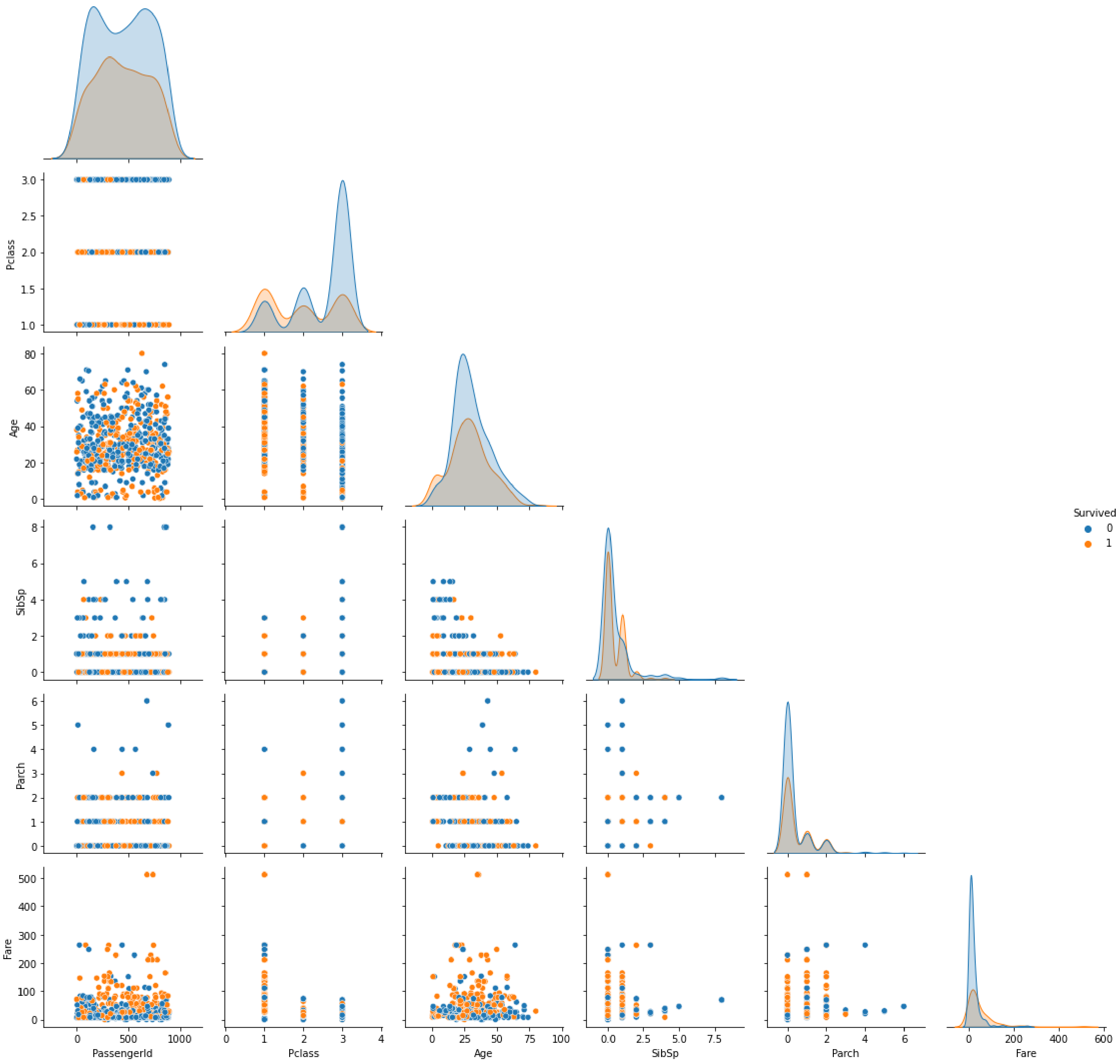


In [287]:
```
fig,ax = plt.subplots(round(n_def_num_cols/2),2,figsize=(16,n_def_num_cols*2))
for i,col in enumerate(num_cols):
  sns.histplot(x=col,data=train_set,hue='Survived',multiple='dodge',ax=ax.ravel()[i],bins=20,lw=1)
fig.tight_layout()
```



- We can see that, the survival rate is higher for the kids and elderly. The survial rate for inividuals between 20-30 is very low.
- We could also observe that the chances of survival is increasing with the price paid for the ticket.

In [288]:
```
sns.pairplot(hue='Survived',data=train_set,corner=True)
plt.tight_layout()
```



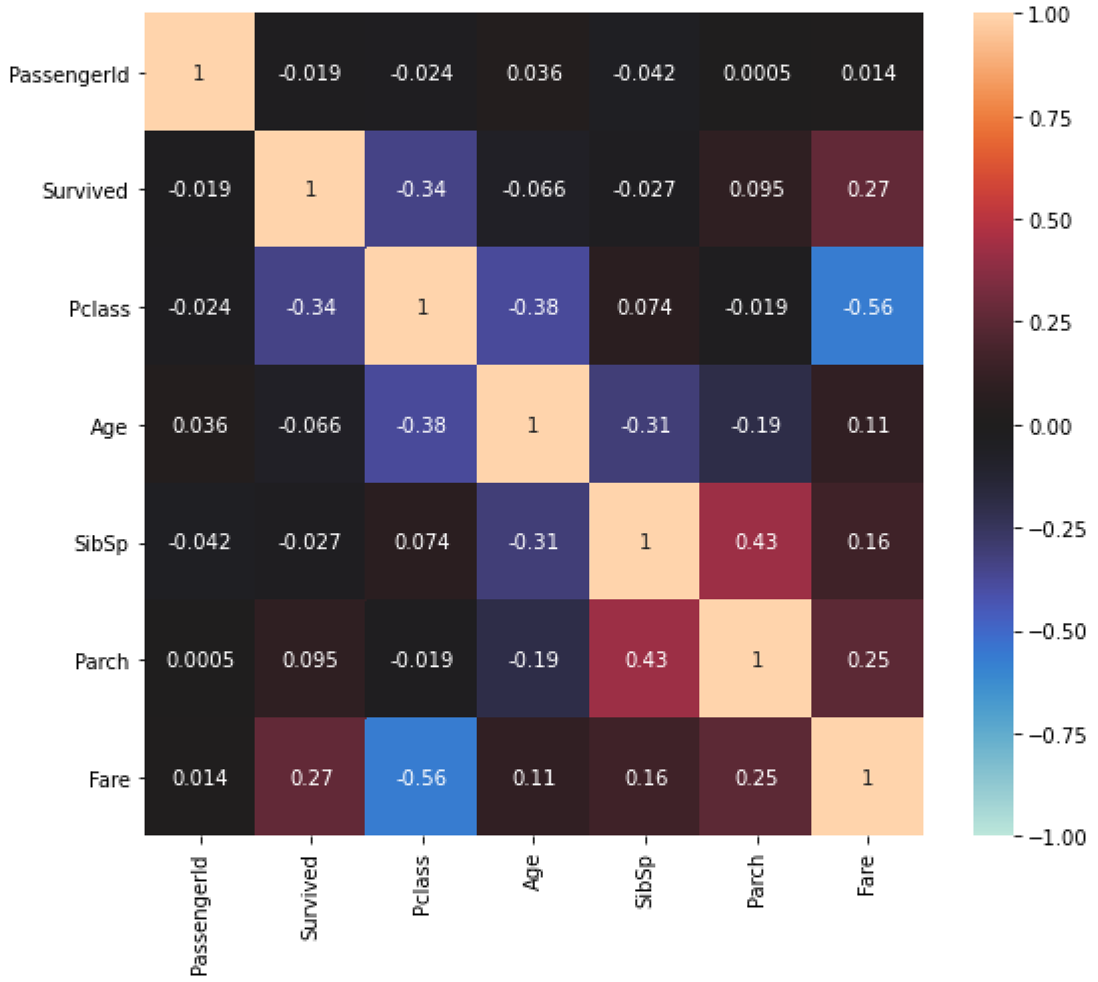
- The classification of survival isnt linearly seperable with any of the feature.
- There arent any distinct correlation within various features.

In [289]:

```
train_corr = train_set.corr()
```

In [290]:

```
plt.subplots(figsize=(8,7))
sns.heatmap(train_corr,vmax=1,vmin=-1,annot=True,cmap=sns.color_palette("icefire", as_cmap=True))
plt.tight_layout()
```



	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1	-0.019	-0.024	0.036	-0.042	0.0005	0.014
Survived	-0.019	1	-0.34	-0.066	-0.027	0.095	0.27
Pclass	-0.024	-0.34	1	-0.38	0.074	-0.019	-0.56
Age	0.036	-0.066	-0.38	1	-0.31	-0.19	0.11
SibSp	-0.042	-0.027	0.074	-0.31	1	0.43	0.16
Parch	0.0005	0.095	-0.019	-0.19	0.43	1	0.25
Fare	0.014	0.27	-0.56	0.11	0.16	0.25	1

In [291]:

```
print("Correlation of Features with 'Survived' \n")
train_corr.loc[:,'Survived'].sort_values(ascending=False).drop('Survived')
```

```
Correlation of Features with 'Survived'
```

Out[291]:

```
Fare           0.268678
Parch          0.094806
PassengerId   -0.018821
SibSp         -0.027243
Age           -0.065538
Pclass        -0.340564
Name: Survived, dtype: float64
```

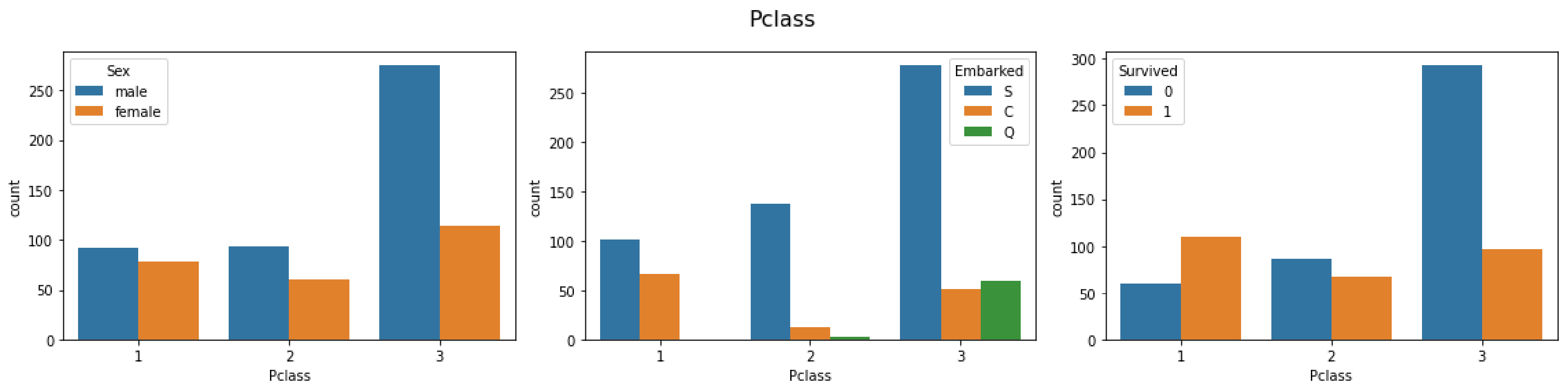
In [292]:

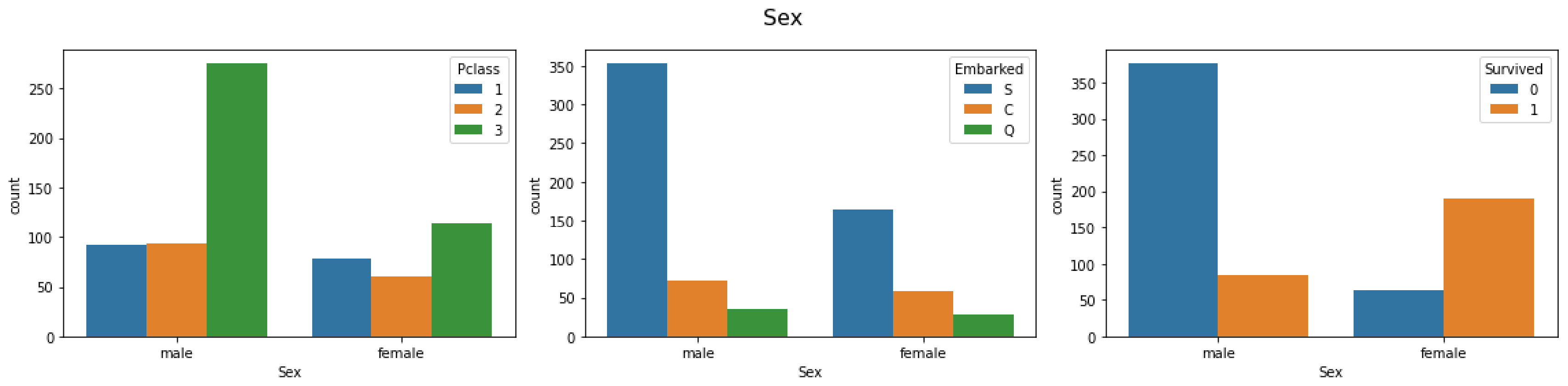
```
print("Correlation within Features  \n")
for i,y in enumerate(train_corr.index):
    for j,x in enumerate(train_corr.columns.drop('Survived')):
        if(j<i):
            continue
        if ((train_corr.loc[x,y] >0.4) or (train_corr.loc[x,y] <-0.4)) and x!=y:
            print(f'{x} - {y}  : {train_corr.loc[x,y]}')
```

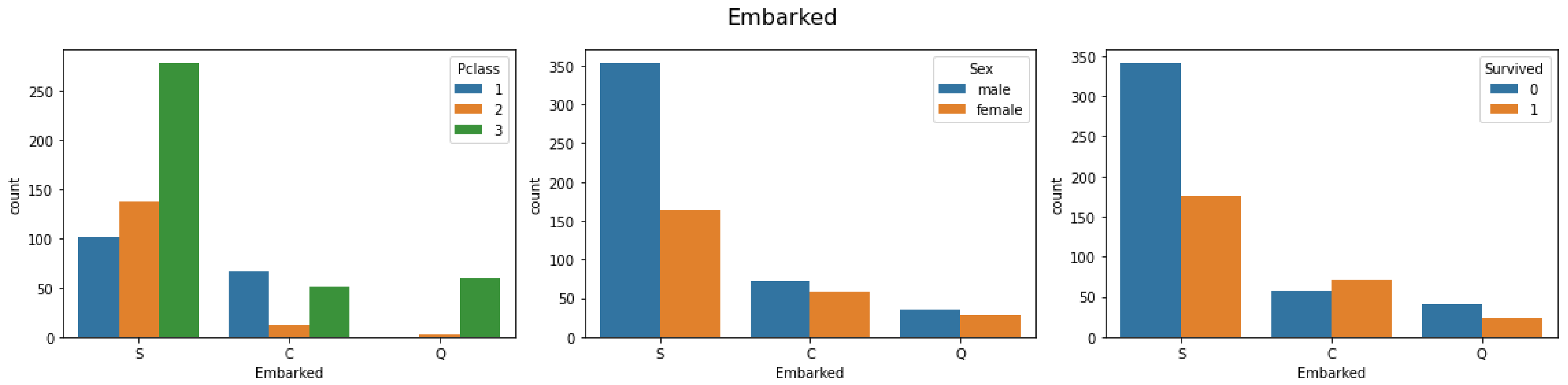
```
Correlation within Features

Fare - Pclass  : -0.5648039044618169
Parch - SibSp  : 0.4272373837023007
```

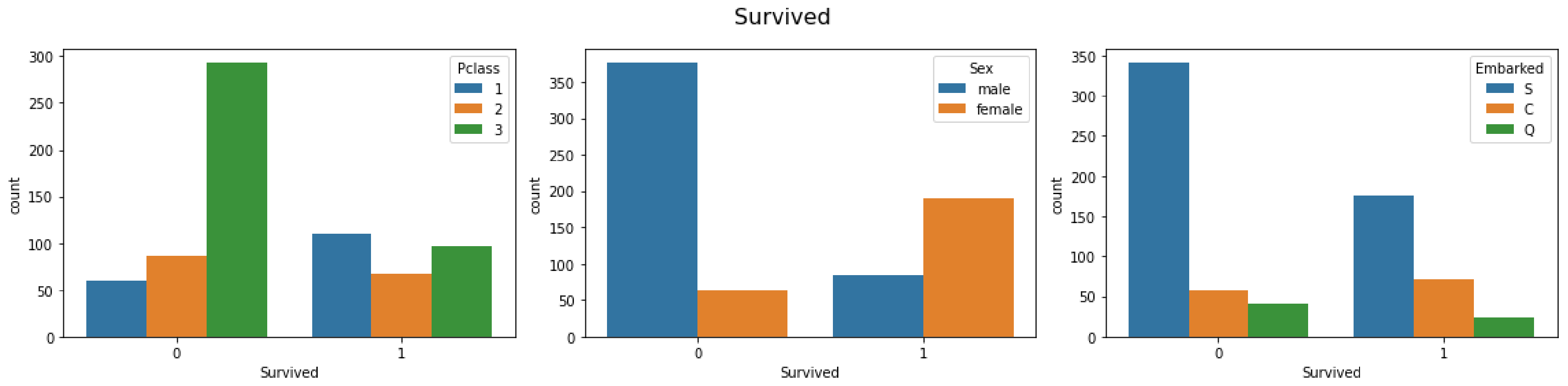
- The Ticket class has the highest correlation with the target column 'Survived'
- The ticket fare and the ticket class are correlated which makes much sense.
- Also the number of siblings/spouses aboard is correlated with the number of parents/children aboard.

In [293]:
```
for i,col in enumerate(['Pclass','Sex','Embarked','Survived']):
  j=0
  fig,ax = plt.subplots(1,3,figsize=(16,4),)
  for col1 in ['Pclass','Sex','Embarked','Survived']:
    if col1!=col:
      sns.countplot(x=col,data=train_set,hue=col1,ax=ax[j])
      j=j+1

  fig.suptitle(col,size=16)
  fig.tight_layout()
```









As we can see, some of the insights that can be drawn are

- The survival chances for females were much higher than males.
- The chances of survival were higher for Individuals with TicketClass('Pclass')-1. Passengers with Class-1 Ticket has survived more than any other class.
- Passengers who embarked from port Cherbourg has a higher survival ratio.
- Most passengers with 1st class tickets survived and the survival rate was much higher than any other ticket class. It could also be noted that there were no 1st class passengers from Queenstown.
- Passengers embarked from Cherbourg has higher survival ratio.

In [294]:
```
from scipy.stats import chi2_contingency
```

In [295]:

```
alpha = 0.05
for col in cat_cols:
  cross_table = pd.crosstab(train_set[col],train_set['Survived'])
  chi2_stat,p_value, dof, exp = chi2_contingency(cross_table)
  if p_value <= alpha:
    print(f"{col}-Survived \np-value : ",p_value)
    print("Dependent (reject H0)",'\n')
  else:
    print(f"{col}-Survived \np-value : ",p_value)
    print("Independent (fail to reject H0)",'\n')
```

```
Sex-Survived
p-value :  1.277767685540944e-49
Dependent (reject H0)

Ticket-Survived
p-value :  0.03529249290136183
Dependent (reject H0)

Cabin-Survived
p-value :  0.1860186007157923
Independent (fail to reject H0)

Embarked-Survived
p-value :  4.255379308445157e-05
Dependent (reject H0)

Pclass-Survived
p-value :  1.1461931253253146e-18
Dependent (reject H0)
```

Only 'Cabin' had no relation with 'Survived' column. This could also be due to the unavailability of over 75% of the data for 'Cabin'

Data Preparation

In [296]:

```
train_set.dtypes
```

Out[296]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [297]:

```
# Modifying DataType
#
train_set.loc[:,cat_cols] = train_set[cat_cols].astype('category',errors='ignore')
train_set.loc[:,'PassengerId'] = train_set[['PassengerId']].astype('object',errors='ignore')
```

```
D:\anaconda3\lib\site-packages\pandas\core\indexing.py:1787: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  self._setitem_single_column(loc, val, pi)
```

In [298]:

```
def missing_count(data,cols=None):
  print("Number of Instances : ",len(data))
  print("Number of Missing Values in :")
  df = pd.DataFrame(data)
  if cols==None:
    cols=df.columns
  for x in cols:
    count  = df[x].isna().sum()
    if count >=1:
      print(f' - {x} : {count}({count*100/len(df):0.2f}%)')
```

In [299]:

```
missing_count(train_set)
```

```
Number of Instances :  712
Number of Missing Values in :
 - Age : 140(19.66%)
 - Cabin : 550(77.25%)
 - Embarked : 2(0.28%)
```

- Embarked has 2 values missing, we could remove the entry/instance since its only 2.
- Age has 140 values missing, which constitutes about 20% of the whole data. We could impute these missing values.
- Cabin has more than 75% of missing values. Ideally we should drop this feature or find some way to extract any available information if possible.

Feature Engineering

Feature - 'Embarked '

In [300]:

```
train_set.dropna(subset=['Embarked'],inplace=True)
train_set.reset_index(drop=True,inplace=True)
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/2049774218.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set.dropna(subset=['Embarked'],inplace=True)
```

We shall impute the missing values in 'Age'. We had noticed that 'Pclass' had the highest correlation with 'Age', so instead of taking the median of the whole training set, we shall impute with class-wise(ticket) median age.

Feature - 'Age'

In [301]:

```
pclass_avg_age = train_set.groupby(['Pclass'])['Age'].median()
pclass_avg_age
```

Out[301]:

```
Pclass
1    38.0
2    30.0
3    24.0
Name: Age, dtype: float64
```

In [302]:

```
pd.Series(train_set.columns)
```

Out[302]:

```
0     PassengerId
1        Survived
2          Pclass
3            Name
4             Sex
5             Age
6           SibSp
7           Parch
8          Ticket
9            Fare
10          Cabin
11       Embarked
dtype: object
```

In [303]:

```
train_set.Age = train_set.apply((lambda x: pclass_avg_age[x[8]] if np.isnan(x[0]) else x[0]),axis=1)
```

```
D:\anaconda3\lib\site-packages\pandas\core\generic.py:5494: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  self[name] = value
```

In [304]:

```
missing_count(train_set)
```

```
Number of Instances :  710
Number of Missing Values in :
 - Cabin : 550(77.46%)
```

'Cabin' has over 77% of its values missing, dropping the column is the ideal choice. But before droping, we shall try to extract any information if possible.

Feature - 'Cabin'

In [305]:
```
print("No. of Entries available : ",train_set.Cabin.notna().sum(),'\n')
train_set.Cabin.unique()
```

```
No. of Entries available :  160
```

Out[305]:
```
[NaN, 'C128', 'C103', 'B35', 'C22 C26', ..., 'C2', 'A26', 'D7', 'E12', 'C125']
Length: 119
Categories (118, object): ['C128', 'C103', 'B35', 'C22 C26', ..., 'A26', 'D7', 'E12', 'C125']
```

In [306]:
```
# Checking if all the values in Cabin starts with an alphabet
pd.Series([str(x)[0].isalpha() if x!=np.nan else False for x in train_set.Cabin.unique()]).sum()
```

Out[306]:
```
119
```

In [307]:
```
# Checking if multiple people have the same cabin/s
train_set.Cabin.value_counts()
```

Out[307]:
```
G6             4
E101           3
F2             3
C23 C25 C27    3
C22 C26        3
              ..
C110           1
C106           1
C103           1
T              1
B28            0
Name: Cabin, Length: 119, dtype: int64
```

As we can see, all of 119 unique elements starts with an alphabet. We could group the Cabin codes using this initial alphabet character.

In [308]:
```
shared_cabins = train_set.Cabin.value_counts()[train_set.Cabin.value_counts()>1].index
shared_cabins
```

Out[308]:
```
CategoricalIndex(['G6', 'E101', 'F2', 'C23 C25 C27', 'C22 C26', 'B96 B98',
                  'C68', 'B57 B59 B63 B66', 'B58 B60', 'E33', 'B77', 'E25',
                  'C92', 'E121', 'C126', 'C124', 'B5', 'D35', 'D26', 'C52',
                  'C65', 'D20', 'D', 'B51 B53 B55', 'E24', 'B49', 'B20', 'F4',
                  'F33', 'F G73', 'E8', 'B18', 'E67', 'C93', 'E44'],
                 categories=['A14', 'A16', 'A19', 'A23', 'A24', 'A26', 'A31', 'A32', ...], ordered=False, dtyp
e='category')
```

In [309]:
```
cabins=[]
for x in train_set.Cabin.value_counts().index:
  if ' ' in x:
    cabins.extend(x.split(' '))
  else:
    cabins.append(x)
print(cabins)
```

```
['G6', 'E101', 'F2', 'C23', 'C25', 'C27', 'C22', 'C26', 'B96', 'B98', 'C68', 'B57', 'B59', 'B63', 'B66', 'B5
8', 'B60', 'E33', 'B77', 'E25', 'C92', 'E121', 'C126', 'C124', 'B5', 'D35', 'D26', 'C52', 'C65', 'D20', 'D',
'B51', 'B53', 'B55', 'E24', 'B49', 'B20', 'F4', 'F33', 'F', 'G73', 'E8', 'B18', 'E67', 'C93', 'E44', 'E38', 'E
31', 'D33', 'D28', 'F', 'E69', 'E40', 'F38', 'E77', 'D19', 'D17', 'D15', 'D11', 'E36', 'C99', 'D36', 'E49', 'E
17', 'D37', 'D47', 'D48', 'E34', 'D49', 'D50', 'D56', 'D7', 'D9', 'E10', 'E46', 'E12', 'E63', 'E58', 'E68', 'A
14', 'C91', 'B22', 'B78', 'B73', 'B69', 'B41', 'B39', 'B38', 'B37', 'B35', 'B30', 'B3', 'B101', 'C90', 'A7',
'A5', 'A36', 'A34', 'A32', 'A31', 'A26', 'A24', 'A23', 'A19', 'B79', 'B80', 'B86', 'C30', 'C87', 'C85', 'C83',
'C70', 'C7', 'A16', 'C50', 'C47', 'C45', 'C32', 'C2', 'C101', 'C148', 'C128', 'C125', 'C123', 'C118', 'C111',
'C110', 'C106', 'C103', 'T', 'B28']
```

In [310]:
```
cabin_cat = []
cabin_cat.extend([x[0] for x in cabins])
pd.Series(cabin_cat).value_counts()
```

Out[310]:
```
C    36
B    32
E    23
D    19
A    12
F     6
G     2
T     1
dtype: int64
```

In [311]:
```
for cabin_x in set(cabin_cat):
  train_set[f'Cabin_{cabin_x}']=[int(cabin_x in str(x)) for x in train_set.Cabin]
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/1004718502.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set[f'Cabin_{cabin_x}']=[int(cabin_x in str(x)) for x in train_set.Cabin]
```

We could also try to group Cabins by the number of passengers in it and also by Cabins with more than 1 passenger as passengers in groups may have higher chance of survival.

In [312]:

```
# Categories of Cabins with more than 1 passenger.

for cabin_ in shared_cabins:
  train_set[f'Cabin_shared_{cabin_}']=[int(x==cabin_) for x in train_set.Cabin]
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/1415746914.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set[f'Cabin_shared_{cabin_}']=[int(x==cabin_) for x in train_set.Cabin]
```

In [313]:

```
passengers_in_cabin = train_set.Cabin.value_counts()[train_set.Cabin.value_counts()>1]
passengers_in_cabin
```

Out[313]:

```
G6                 4
E101               3
F2                 3
C23 C25 C27        3
C22 C26            3
B96 B98            3
C68                2
B57 B59 B63 B66    2
B58 B60            2
E33                2
B77                2
E25                2
C92                2
E121               2
C126               2
C124               2
B5                 2
D35                2
D26                2
C52                2
C65                2
D20                2
D                  2
B51 B53 B55        2
E24                2
B49                2
B20                2
F4                 2
F33                2
F G73              2
E8                 2
B18                2
E67                2
C93                2
E44                2
Name: Cabin, dtype: int64
```

In [314]:

```
for n in passengers_in_cabin.unique():
  train_set[f'{n}_Passenger_Cabin'] =  0
for index,x in enumerate(train_set.Cabin):
  if x in passengers_in_cabin.index:
    n = passengers_in_cabin[x]
    train_set.loc[index,f'{n}_Passenger_Cabin'] =  1
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/2421986206.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set[f'{n}_Passenger_Cabin'] =  0
D:\anaconda3\lib\site-packages\pandas\core\indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  self._setitem_single_column(loc, value, pi)
```

In [315]:

```
train_set.columns
```

Out[315]:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Cabin_C', 'Cabin_A',
       'Cabin_F', 'Cabin_B', 'Cabin_T', 'Cabin_G', 'Cabin_E', 'Cabin_D',
       'Cabin_shared_G6', 'Cabin_shared_E101', 'Cabin_shared_F2',
       'Cabin_shared_C23 C25 C27', 'Cabin_shared_C22 C26',
       'Cabin_shared_B96 B98', 'Cabin_shared_C68',
       'Cabin_shared_B57 B59 B63 B66', 'Cabin_shared_B58 B60',
       'Cabin_shared_E33', 'Cabin_shared_B77', 'Cabin_shared_E25',
       'Cabin_shared_C92', 'Cabin_shared_E121', 'Cabin_shared_C126',
       'Cabin_shared_C124', 'Cabin_shared_B5', 'Cabin_shared_D35',
       'Cabin_shared_D26', 'Cabin_shared_C52', 'Cabin_shared_C65',
       'Cabin_shared_D20', 'Cabin_shared_D', 'Cabin_shared_B51 B53 B55',
       'Cabin_shared_E24', 'Cabin_shared_B49', 'Cabin_shared_B20',
       'Cabin_shared_F4', 'Cabin_shared_F33', 'Cabin_shared_F G73',
       'Cabin_shared_E8', 'Cabin_shared_B18', 'Cabin_shared_E67',
       'Cabin_shared_C93', 'Cabin_shared_E44', '4_Passenger_Cabin',
       '3_Passenger_Cabin', '2_Passenger_Cabin'],
      dtype='object')
```

We shall create a method to do the above done cleaning tasks

In [316]:

```
def clean_data(X):

    # Modifying DataType
    X.loc[:,cat_cols] = X[cat_cols].astype('category',errors='ignore')
    X.loc[:,'PassengerId'] = X[['PassengerId']].astype('object',errors='ignore')
    X.loc[:,num_cols] = X[num_cols].apply(lambda x: pd.to_numeric(x,errors='coerce'),axis=1)

    #Dropping Missing values in Embarked
    X.dropna(subset=['Embarked'],inplace=True)

    # Imputing Missing values in Age
    X.Age = X.apply((lambda x: pclass_avg_age[x[8]] if np.isnan(x[0]) else x[0]),axis=1)

    X = X.reset_index(drop=True)

    if 'Survived' in X:
      y = X.Survived
      X = X.drop(['Survived'],axis=1)

      return X,y
    else:
      return X
```

We could follow a similar approach to 'Ticket' as in 'Cabin'. We shall try to extract any useful information possible from Ticket column.

Feature - 'Ticket'

In [317]:

```
train_set.Ticket.nunique()
```

Out[317]:

```
568
```

In [318]:

```
train_set.Ticket.head(25)
```

Out[318]:

```
0            S.P. 3464
1            Fa 265302
2     C.A./SOTON 34068
3               350035
4               349242
5                29750
6               113510
7               113783
8             PC 17477
9                 2699
10              349253
11              364498
12              113781
13              349251
14              345779
15              248727
16              349909
17               26707
18              347085
19              330932
20              248738
21            PC 17757
22              345764
23               16966
24              229236
Name: Ticket, dtype: category
Categories (569, object): ['110152', '110413', '110465', '110564', ..., 'W./C. 6608', 'W.E.P. 5734', 'W/C 1420
8', 'WE/P 5735']
```

```
In [319]: # Checking if the initial text in String are random/unique or if it has any significance
          pd.Series([str(x).split(' ')[0] if ' ' in str(x) else x for x in train_set.Ticket]).value_counts()
```

```
Out[319]: PC        46
          C.A.      21
          STON/O     9
          A/5        9
          W./C.      7
                    ..
          2687       1
          349253     1
          349236     1
          2620       1
          13509      1
          Length: 466, dtype: int64
```

The tickets seems much more random at first glance apart from the fact that they are mostly numerical or numericals preceeded by some text. Individuals travelling together will have the same ticket code.

```
In [320]: ticket_codes=[]
          for x in train_set.Ticket.value_counts().index:
            if ' ' in x:
              ticket_codes.append(x.split(' ')[0])
          print(ticket_codes)
```

```
['CA', 'CA.', 'S.O.C.', 'PC', 'W./C.', 'F.C.C.', 'PC', 'PC', 'PC', 'C.A.', 'PC', 'PC', 'PP', 'C.A.', 'C.A.',
'C.A.', 'PC', 'PC', 'PC', 'PC', 'S.C./PARIS', 'A/4', 'W./C.', 'S.O./P.P.', 'SC/Paris', 'A/5.', 'C', 'PC', 'SOT
ON/O.Q.', 'SOTON/O.Q.', 'SO/C', 'SCO/W', 'SC/Paris', 'SC/PARIS', 'SC/PARIS', 'SC/PARIS', 'SC/AH', 'SC', 'S.W./
PP', 'S.P.', 'S.O.P.', 'S.O./P.P.', 'S.C./A.4.', 'PP', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC',
'PC', 'PC', 'SOTON/O.Q.', 'SOTON/O.Q.', 'SOTON/O.Q.', 'STON/O', 'W/C', 'W.E.P.', 'W./C.', 'W./C.', 'SW/PP', 'S
TON/O2.', 'STON/O2.', 'STON/O2.', 'STON/O2.', 'STON/O2.', 'STON/O', 'STON/O', 'STON/O', 'SOTON/O.Q.', 'STON/
O', 'STON/O', 'STON/O', 'STON/O', 'STON/O', 'SOTON/OQ', 'SOTON/OQ', 'SOTON/OQ', 'SOTON/OQ', 'SOTON/OQ', 'SOTO
N/O2', 'SOTON/O2', 'PC', 'PC', 'PC', 'C', 'A4.', 'A/5.', 'A/5.', 'A/5.', 'A/5.', 'A/5', 'A/5', 'A/5', 'A/5',
'A/5', 'A/5', 'A/5', 'A/5', 'A/5', 'A/4.', 'A/4.', 'A/4.', 'A/4', 'A.5.', 'A.5.', 'A./5.', 'A./5.', 'C', 'C',
'C.A.', 'F.C.C.', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'PC', 'P/PP', 'Fa', 'F.C.C.', 'CA.', 'C.A.',
'C.A./SOTON', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'C.A.', 'WE/P']
```

```
In [321]: ticket_codes = [x.replace('.','') for x in ticket_codes]
          ticket_pattern_uniq = pd.Series(ticket_codes).unique()
          pd.Series(ticket_codes).value_counts()
```

```
Out[321]: PC           34
          CA           20
          A/5          16
          SOTON/OQ     11
          STON/O        9
          W/C           5
          A/4           5
          STON/O2       5
          C             4
          SC/PARIS      4
          FCC           3
          SC/Paris      2
          A5            2
          SOTON/O2      2
          SO/PP         2
          SW/PP         2
          PP            2
          SCO/W         1
          WE/P          1
          P/PP          1
          SOC           1
          SC/AH         1
          SP            1
          SC            1
          Fa            1
          SOP           1
          CA/SOTON      1
          SC/A4         1
          A4            1
          WEP           1
          SO/C          1
          dtype: int64
```

```
In [322]: for x in ticket_pattern_uniq:
            train_set['Ticket_'+x] = [int(x == str(y).split(' ')[0].replace('.','')) for y in train_set.Ticket]
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/3030581625.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set['Ticket_'+x] = [int(x == str(y).split(' ')[0].replace('.','')) for y in train_set.Ticket]
```

In [323]:

```
shared_tickets = train_set.Ticket.value_counts()[train_set.Ticket.value_counts()>1]
shared_tickets
```

Out[323]:

```
347082          5
CA 2144         5
1601            5
3101295         4
349909          4
               ..
11967           2
SC/Paris 2123   2
16966           2
392096          2
A/5. 3336       2
Name: Ticket, Length: 103, dtype: int64
```

In [324]:

```
for n in shared_tickets.unique():
  train_set[f'{n}_Passenger_Ticket'] =  0
for index,x in enumerate(train_set.Ticket):
  if x in shared_tickets.index:
    n = shared_tickets[x]
    train_set.loc[index,f'{n}_Passenger_Ticket'] =  1
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/3651013050.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set[f'{n}_Passenger_Ticket'] =  0
D:\anaconda3\lib\site-packages\pandas\core\indexing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  self._setitem_single_column(loc, value, pi)
```

In [325]:

```
train_set.sample(5)
```

Out[325]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	...	Ticket_A4	Ticket_A5	Ticket_P/PP	Ticl
52	123	0	2	Nasser, Mr. Nicholas	male	123	1	0	237736	30.0708	...	0	0	0	
60	860	0	3	Razi, Mr. Raihed	male	860	0	0	2629	7.2292	...	0	0	0	
92	341	1	2	Navratil, Master. Edmond Roger	male	341	1	1	230080	26.0000	...	0	0	0	
266	229	0	2	Fahlstrom, Mr. Arne Jonas	male	229	0	0	236171	13.0000	...	0	0	0	
250	480	1	3	Hirvonen, Miss. Hildur E	female	480	0	1	3101298	12.2875	...	0	0	0	

5 rows × 93 columns

In [ ]:

Feature - 'PassengerId'

In [326]:

```
train_set['PassengerId']
```

Out[326]:

```
0       68
1      155
2      884
3      500
4      520
      ...
705    575
706    248
707    189
708    329
709    207
Name: PassengerId, Length: 710, dtype: object
```

- PassengerId column contains unique integer values only, no useful information can be extracted from them. Dropping is ideal.

In [327]:

```
train_set['Name']
```

Out[327]:

```
0                             Crease, Mr. Ernest James
1                               Olsen, Mr. Ole Martin
2                          Banfield, Mr. Frederick James
3                                  Svensson, Mr. Olof
4                                 Pavlovic, Mr. Stefo
                              ...
705                          Rush, Mr. Alfred George John
706                        Hamalainen, Mrs. William (Anna)
707                                  Bourke, Mr. John
708    Goldsmith, Mrs. Frank John (Emily Alice Brown)
709                           Backstrom, Mr. Karl Alfred
Name: Name, Length: 710, dtype: object
```

- For passenger 'Name', all seems to have a 'Title'.

In [328]:

```
train_set['Title'] = train_set['Name'].apply(lambda x: x.split(', ')[1].split('.')[0])
print(train_set['Title'].unique())
train_set['Title'].nunique()
```

```
['Mr' 'Mrs' 'Miss' 'Mlle' 'Master' 'Dr' 'Major' 'Rev' 'Lady'
 'the Countess' 'Don' 'Jonkheer' 'Ms' 'Col']

C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/1176858206.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set['Title'] = train_set['Name'].apply(lambda x: x.split(', ')[1].split('.')[0])
```

Out[328]:

```
14
```

- All passengers have a title in their name and to be specific there are 14 titles.
- Mlle is French for Ms, so we shall replace this.

In [329]:

```
train_set['Title'] = train_set['Title'].replace(['Mlle'],['Ms'])
print(train_set['Title'].unique())
train_set['Title'].nunique()
```

```
['Mr' 'Mrs' 'Miss' 'Ms' 'Master' 'Dr' 'Major' 'Rev' 'Lady' 'the Countess'
 'Don' 'Jonkheer' 'Col']

C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/1482636708.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set['Title'] = train_set['Title'].replace(['Mlle'],['Ms'])
```

Out[329]:

```
13
```

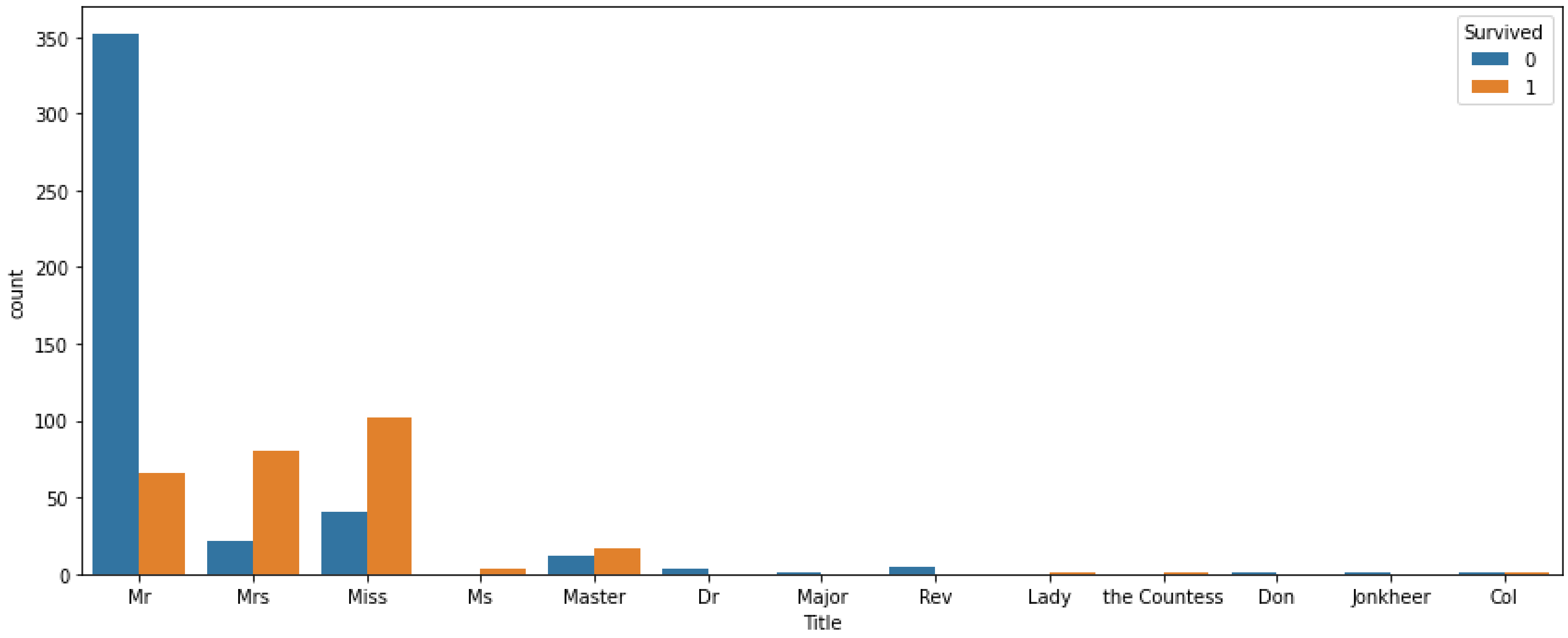
In [330]:

```
pd.crosstab(train_set['Survived'],train_set['Title'])
```

Out[330]:

Title	Col	Don	Dr	Jonkheer	Lady	Major	Master	Miss	Mr	Mrs	Ms	Rev	the Countess
Survived													
0	1	1	3	1	0	1	12	41	352	22	0	5	0
1	1	0	0	0	1	0	17	102	66	80	3	0	1

In [331]:
```
fig,ax = plt.subplots(figsize=(12,5))
sns.countplot(hue='Survived',data=train_set,x='Title',)
plt.tight_layout()
```



We could also add a feature of Family Size

Feature - 'SibSp' & 'Parch'

In [332]:
```
train_set['FamilySize'] = train_set.SibSp + train_set.Parch
train_set['FamilySize'].unique()
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/3392671291.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  train_set['FamilySize'] = train_set.SibSp +   train_set.Parch
```

Out[332]:
```
array([ 0,  2,  3,  4,  1,  5, 10,  7,  6], dtype=int64)
```

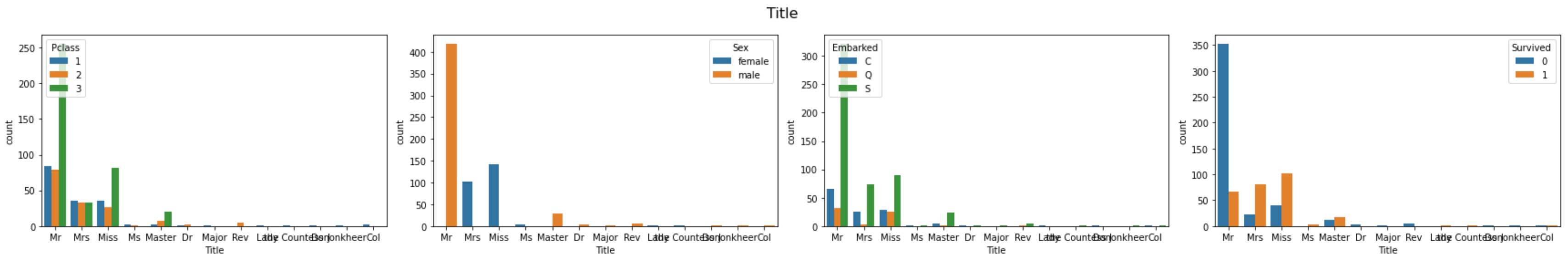
We have now extracted information from the features 'Cabin','Ticket' and 'Name' and now we shall drop these columns along with 'PassengerId'.

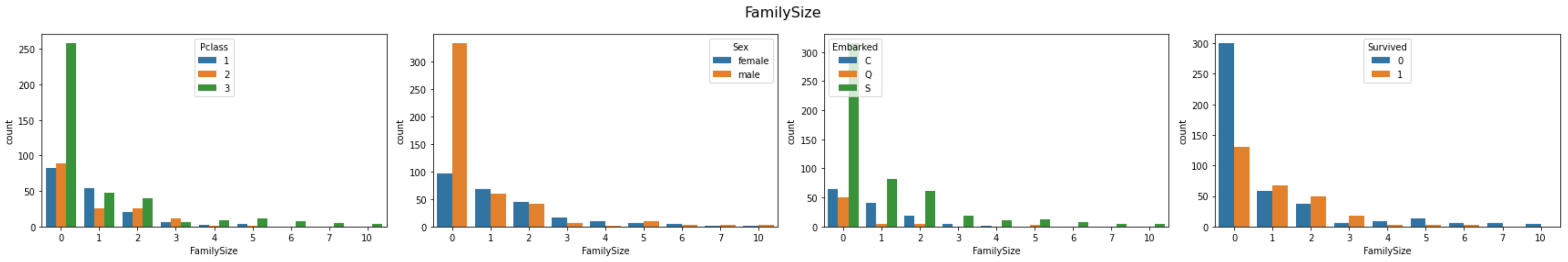
In [333]:
```
train_set.drop(['Cabin','Ticket','PassengerId','Name'],axis=1,inplace=True)
```

```
D:\anaconda3\lib\site-packages\pandas\core\frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  return super().drop(
```

In [334]:
```
cat_cols_updated = list(cat_cols)
cat_cols_updated.append('Title')
num_cols_updated = list(num_cols)
num_cols_updated.append('FamilySize')
```

In [335]:
```
for i,col in enumerate(['Title','FamilySize']):
  j=0
  fig,ax = plt.subplots(1,4,figsize=(24,4),)
  for col1 in ['Pclass','Sex','Embarked','Survived']:
      sns.countplot(x=col,data=train_set,hue=col1,ax=ax[j])
      j=j+1

  fig.suptitle(col,size=16)
  fig.tight_layout()
```





We will create a Custom Transformer to extract/create new features

In [336]:
```
from sklearn.base import TransformerMixin,BaseEstimator
```

In [337]:

```
class FeatureEngineering(TransformerMixin,BaseEstimator):

  def __init__(self):
    self

  def fit(self,X,y=None):
    return self

  def transform(self,X,y=None):

    X = X.reset_index(drop=True)

    # Creating Feature 'Title'
    X['Title'] = X['Name'].apply(lambda x: x.split(', ')[1].split('.')[0])
    X['Title'] = X['Title'].replace(['Mlle'],['Ms'])

    # Creating Feature 'FamilySize'
    X['FamilySize'] = X.SibSp + X.Parch


    #cabins=[]
    #cabin_cat = []
    #for x in X.Cabin.value_counts().index:
    #  if ' ' in x:
    #    cabins.extend(x.split(' '))
    #  else:
    #    cabins.append(x)
    #cabin_cat.extend([x[0] for x in cabins])
    for cabin_x in set(cabin_cat):
      X[f'Cabin_{cabin_x}']=[int(cabin_x in str(x)) for x in X.Cabin]

    #shared_cabins = X.Cabin.value_counts()[X.Cabin.value_counts()>1].index
    for cabin_ in shared_cabins:
      X[f'Cabin_shared_{cabin_}']=[int(x==cabin_) for x in X.Cabin]

    #passengers_in_cabin = X.Cabin.value_counts()[X.Cabin.value_counts()>1]
    for n in passengers_in_cabin.unique():
      X[f'{n}_Passenger_Cabin'] =  0
    for index,x in enumerate(X.Cabin):
      if x in passengers_in_cabin.index:
        n = passengers_in_cabin[x]
        X.loc[index,f'{n}_Passenger_Cabin'] =  1


    #ticket_codes=[]
    #for x in X.Ticket.value_counts().index:
    #  if ' ' in x:
    #    ticket_codes.append(x.split(' ')[0])
    #ticket_codes = [x.replace('.','') for x in ticket_codes]
    #ticket_pattern_uniq = pd.Series(ticket_codes).unique()
    for x in ticket_pattern_uniq:
      X['Ticket_'+x] = [int(x == str(y).split(' ')[0].replace('.','')) for y in X.Ticket]

    for ticket_ in shared_tickets.index:
      X[f'Ticket_shared_{ticket_}']=[int(x==ticket_) for x in X.Ticket]

    #shared_tickets = X.Ticket.value_counts()[X.Ticket.value_counts()>1]
    for n in shared_tickets.unique():
      X[f'{n}_Passenger_Ticket'] =  0
    for index,x in enumerate(X.Ticket):
      if x in shared_tickets.index:
        n = shared_tickets[x]
        X.loc[index,f'{n}_Passenger_Ticket'] =  1

    X = X.drop(['PassengerId','Name','Ticket', 'Cabin'],axis=1)

    return X
```

In [338]:

```
# A custom transformer to view the data inbetween the various stages of the pipeline
class TransformationSubStage(TransformerMixin,BaseEstimator):

  def __init__(self):
    self
    self.transformed_X = None
    self.transformed_y = None

  def fit(self,X,y=None):
    return self

  def transform(self,X,y=None):
    self.transformed_X = X
    self.transformed_y = y
    return X
```

Building a Pipeline

In [339]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
```

In [340]:
```
from sklearn.preprocessing import StandardScaler,OneHotEncoder
```

In [341]:
```
sub_pipe1 = Pipeline([
                      ('imputer',SimpleImputer(strategy='most_frequent')),
                      (('ohe',OneHotEncoder(handle_unknown='ignore')))
])
```

In [342]:
```
coltransformer = ColumnTransformer([
                                    ('num_impute',SimpleImputer(strategy='median'),['Age', 'SibSp', 'Parch', 'Fa
                                    ('num_impute2',SimpleImputer(strategy='mean'),['Fare']),
                                    ('cat_impute',sub_pipe1,['Sex', 'Embarked', 'Pclass', 'Title'])
],remainder='passthrough')
```

In [343]:
```
pipe = Pipeline([
                 ('feat_engg',FeatureEngineering()),
                 ('substage_feat_engg',TransformationSubStage()),
                 ('coltransformer',coltransformer),
                 ('substage_coltransformer',TransformationSubStage()),
                 ('num',StandardScaler()),
])
```

In [344]:
```
X_train,y_train = clean_data(train_original)
X_train = pipe.fit_transform(X_train)
```

In [345]:
```
X_train.shape
```

Out[345]:
```
(710, 210)
```

In [346]:
```
X_test,y_test = clean_data(test_set)
X_test = pipe.transform(X_test)
```

```
D:\anaconda3\lib\site-packages\pandas\core\indexing.py:1787: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  self._setitem_single_column(loc, val, pi)
C:\Users\Public\Documents\Wondershare\CreatorTemp/ipykernel_1884/3957259135.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  X.dropna(subset=['Embarked'],inplace=True)
D:\anaconda3\lib\site-packages\pandas\core\generic.py:5494: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
a-view-versus-a-copy)
  self[name] = value
```

In [347]:
```
from sklearn.linear_model import LogisticRegression,LogisticRegressionCV
from sklearn.svm import SVC,LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,VotingClassifier,AdaBoostClassifier,GradientBoostingClassifi
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

In [348]:
```
from sklearn.metrics import accuracy_score
```

In [349]:
```
from sklearn.model_selection import cross_val_score,GridSearchCV,RandomizedSearchCV
```

In [350]:
```
models = {}
```

ML Modeling

Logistic Regression

In [351]:
```
logreg_gridSearch = LogisticRegressionCV(solver='saga',penalty='elasticnet',Cs=[0.1,0.2,0.5,1,10,15,20,25,50,100
logreg_gridSearch.fit(X_train,y_train)
```

```
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reache
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reache
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reache
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reache
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reache
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reache
d which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

In [352]:
```
logreg = LogisticRegression(solver='saga',penalty='elasticnet',C=logreg_gridSearch.C_[0],l1_ratio=logreg_gridSea
logreg.fit(X_train,y_train)
```

```
D:\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

Out[352]:
```
LogisticRegression(C=0.1, l1_ratio=0.65, n_jobs=-1, penalty='elasticnet',
                   random_state=0, solver='saga')
```

In [353]:
```
accuracy = accuracy_score(y_test,logreg.predict(X_test))
accuracy
```

Out[353]:
```
0.8379888268156425
```

In [354]:
```
models['Logistic Regression'] = accuracy
```

Linear SVC

In [355]:
```
params ={'C':[0.01,0.1,1,2,5,10,20,50,100,1000],
         'penalty':['l1','l2']}
lin_svc = GridSearchCV(LinearSVC(random_state=0),params)
```

In [356]:
```
lin_svc.fit(X_train,y_train)
lin_svc.best_params_
```

```
D:\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, i
ncrease the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
D:\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, i
ncrease the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
D:\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, i
ncrease the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
D:\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, i
ncrease the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
D:\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:610: FitFailedWarning: Estimator fit f
ailed. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "D:\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 593, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "D:\anaconda3\lib\site-packages\sklearn\svm\_classes.py", line 234, in fit
    self.coef_, self.intercept_, self.n_iter_ = _fit_liblinear(
  File "D:\anaconda3\lib\site-packages\sklearn\svm\_base.py", line 974, in _fit_liblinear
```

In [357]:
```
lin_svc = lin_svc.best_estimator_
```

In [358]:
```
accuracy = accuracy_score(y_test,lin_svc.predict(X_test))
accuracy
```

Out[358]:
```
0.8324022346368715
```

In [359]:
```
models['Linear SVC'] = accuracy
```

SVC

In [360]:
```
params ={'C':[0.01,0.1,1,2,5,10,20,50,100,1000],
         'kernel':['rbf','sigmoid']}
svc = GridSearchCV(SVC(random_state =0,probability=True),params)
```

In [361]:
```
svc.fit(X_train,y_train)
```

Out[361]:
```
GridSearchCV(estimator=SVC(probability=True, random_state=0),
             param_grid={'C': [0.01, 0.1, 1, 2, 5, 10, 20, 50, 100, 1000],
                         'kernel': ['rbf', 'sigmoid']})
```

In [362]:
```
svc.best_params_
```

Out[362]:
```
{'C': 2, 'kernel': 'sigmoid'}
```

In [363]:
```
svc = svc.best_estimator_
```

In [364]:
```
accuracy = accuracy_score(y_test,svc.predict(X_test))
accuracy
```

Out[364]:
```
0.8379888268156425
```

In [365]:
```
models['SVC'] = accuracy
```

Decision Tree Classifier

In [366]:
```
dt_clf = DecisionTreeClassifier(random_state =0)
dt_clf.fit(X_train,y_train)
```

Out[366]:
```
DecisionTreeClassifier(random_state=0)
```

In [367]:
```
accuracy = accuracy_score(y_test,dt_clf.predict(X_test))
accuracy
```

Out[367]:
```
0.7597765363128491
```

In [368]:
```
models['Decision Tree'] = accuracy
```

Random Forest Classifier (Ensemble)

In [369]:
```
!pip install -q optuna
```

In [370]:
```
import optuna
```

In [371]:
```
def objective(trial):

  max_features=trial.suggest_float('max_features',0.3,1,step=0.05)
  max_samples=trial.suggest_float('max_samples',0.3,0.95,step=0.05)
  min_samples_split=trial.suggest_float('min_samples_split',0.01,0.11,step=0.01)
  class_weight=trial.suggest_categorical('class_weight',['balanced', 'balanced_subsample',None])

  clf = RandomForestClassifier(max_features=max_features, max_samples=max_samples, min_samples_split=min_samples

  return cross_val_score(clf,X_train,y_train,cv=3,n_jobs=-1,scoring='accuracy').mean()
```

In [372]:
```
study = optuna.create_study(direction='maximize')
study.optimize(objective,n_trials=50)
```

```
s': 0.8500000000000001, 'max_samples': 0.5, 'min_samples_split': 0.01, 'class_weight': None}. Best is trial
 1 with value: 0.8351688955636606.
[I 2021-11-03 21:28:37,552] Trial 28 finished with value: 0.8140241722091112 and parameters: {'max_feature
s': 0.75, 'max_samples': 0.6000000000000001, 'min_samples_split': 0.05, 'class_weight': None}. Best is trial
1 with value: 0.8351688955636606.
[I 2021-11-03 21:28:37,802] Trial 29 finished with value: 0.801348065508117 and parameters: {'max_features':
0.95, 'max_samples': 0.3, 'min_samples_split': 0.09, 'class_weight': None}. Best is trial 1 with value: 0.83
51688955636606.
[I 2021-11-03 21:28:38,052] Trial 30 finished with value: 0.8098047629264107 and parameters: {'max_feature
s': 0.7, 'max_samples': 0.35, 'min_samples_split': 0.03, 'class_weight': None}. Best is trial 1 with value:
 0.8351688955636606.
[I 2021-11-03 21:28:38,347] Trial 31 finished with value: 0.8281425063767909 and parameters: {'max_feature
s': 0.95, 'max_samples': 0.55, 'min_samples_split': 0.01, 'class_weight': None}. Best is trial 1 with value:
0.8351688955636606.
```

```
In [373]: best_trial = study.best_trial
          print("Accuracy : ",best_trial.value)
          best_trial.params

          Accuracy :  0.8351688955636606

Out[373]: {'max_features': 0.75,
           'max_samples': 0.65,
           'min_samples_split': 0.02,
           'class_weight': 'balanced_subsample'}
```

```
In [374]: rf_clf = RandomForestClassifier(**best_trial.params,random_state =0)
          rf_clf.fit(X_train,y_train)

Out[374]: RandomForestClassifier(class_weight='balanced_subsample', max_features=0.75,
                                 max_samples=0.65, min_samples_split=0.02,
                                 random_state=0)
```

```
In [375]: accuracy = accuracy_score(y_test,rf_clf.predict(X_test))
          accuracy

Out[375]: 0.8268156424581006
```

```
In [376]: models['Random Forest'] = accuracy
```

K-Nearest Neighbor Classifier

```
In [377]: params = {'n_neighbors' : [2,3,4,5,6,7,8,9,10]}
          knn_clf = GridSearchCV(KNeighborsClassifier(), params)
```

```
In [378]: knn_clf.fit(X_train,y_train)

Out[378]: GridSearchCV(estimator=KNeighborsClassifier(),
                       param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10]})
```

```
In [379]: knn_clf.best_params_

Out[379]: {'n_neighbors': 5}
```

```
In [380]: accuracy = accuracy_score(y_test,knn_clf.predict(X_test))
          accuracy

Out[380]: 0.7932960893854749
```

```
In [381]: models['K-Nearest Neighbor'] = accuracy
```

Gaussian Naive Bayes Classifier

```
In [382]: params = {'var_smoothing': np.logspace(0,-9, num=100)}
          nb_clf = GridSearchCV(GaussianNB(), params)
```

```
In [383]: nb_clf.fit(X_train,y_train)

Out[383]: GridSearchCV(estimator=GaussianNB(),
                       param_grid={'var_smoothing': array([1.00000000e+00, 8.11130831e-01, 6.57933225e-01, 5.33669923e-0
          1,
                 4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
                 1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
                 8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
                 3.51119173e-02, 2.84803587e-02, 2.3101297...
                 1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
                 5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
                 2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
                 1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
                 4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
                 1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-09])})
```

```
In [384]: nb_clf.best_params_

Out[384]: {'var_smoothing': 0.0657933224657568}
```

```
In [385]: accuracy = accuracy_score(y_test,nb_clf.predict(X_test))
          accuracy

Out[385]: 0.659217877094972
```

```
In [386]: models['Gaussian Naive Bayes'] = accuracy
```

```
In [387]: import xgboost as xgb
```

In [388]:

```
cv = cross_val_score(xgb.XGBClassifier(),X_train,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[21:28:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[21:28:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[21:28:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[21:28:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[21:28:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[0.82394366 0.77464789 0.78873239 0.82394366 0.80985915]
0.804225352112676

D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

In [389]:

```
tst = xgb.XGBClassifier().fit(X_train,y_train)
accuracy_score(y_test,tst.predict(X_test))
```

```
D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[21:28:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[389]:

```
0.7821229050279329
```

In [390]:

```
n_iter = []
def objective_xgb(trial):

  params = {
      'learning_rate' : trial.suggest_loguniform('learning_rate',1e-8,0.5),
      'max_depth' : trial.suggest_int('max_depth',8,33),
      'subsample' : trial.suggest_float('subsample',0.5,1),
      'colsample_bynode' : trial.suggest_float('colsample_bynode',0.5,1),
      'lambda' : trial.suggest_loguniform("lambda", 1e-8, 1.0),
      'alpha': trial.suggest_loguniform("alpha", 1e-8, 1.0),
      'gamma' : trial.suggest_loguniform("gamma", 1e-8, 1.0),

      'objective':'binary:logistic','random_state':0
  }

  dtrain = xgb.DMatrix(X_train,y_train)


  cv = xgb.cv(params, dtrain, num_boost_round=1000, metrics='auc', early_stopping_rounds=50)
  n_iter.append(len(cv))

  return cv.mean()['test-auc-mean']
```

In [391]:
```
study = optuna.create_study(direction='maximize')
study.optimize(objective_xgb,n_trials=800)
```

```
[I 2021-11-03 21:34:01,830] Trial 528 finished with value: 0.8726211977401129 and parameters: {'learning_rat
e': 4.2121600898491705e-06, 'max_depth': 30, 'subsample': 0.7072769240519612, 'colsample_bynode': 0.75321331
14183021, 'lambda': 1.4246712547634085e-08, 'alpha': 0.25641152027526204, 'gamma': 1.2673846721679747e-06}.
 Best is trial 108 with value: 0.880475652173913.
[I 2021-11-03 21:34:02,597] Trial 529 finished with value: 0.8713858619047616 and parameters: {'learning_rat
e': 7.631232652230972e-07, 'max_depth': 31, 'subsample': 0.6978234469976564, 'colsample_bynode': 0.723047643
2329899, 'lambda': 2.5374191858315338e-08, 'alpha': 1.1876113202022686e-06, 'gamma': 3.4054165657923254e-0
5}. Best is trial 108 with value: 0.880475652173913.
[I 2021-11-03 21:34:03,110] Trial 530 finished with value: 0.8756164583333335 and parameters: {'learning_rat
e': 1.1347787952904512e-06, 'max_depth': 31, 'subsample': 0.7256100198886333, 'colsample_bynode': 0.71186024
57371496, 'lambda': 7.589726638012149e-05, 'alpha': 0.38880106371405443, 'gamma': 2.2063476895954695e-07}. B
est is trial 108 with value: 0.880475652173913.
[I 2021-11-03 21:34:04,122] Trial 531 finished with value: 0.8721482561983475 and parameters: {'learning_rat
e': 6.174876837220331e-07, 'max_depth': 33, 'subsample': 0.6821321347185139, 'colsample_bynode': 0.743654361
9885226, 'lambda': 6.26264392090065e-08, 'alpha': 6.265562454343735e-06, 'gamma': 2.0159786628961237e-05}. B
est is trial 108 with value: 0.880475652173913.
[I 2021-11-03 21:34:04,713] Trial 532 finished with value: 0.8786416763285022 and parameters: {'learning_rat
e': 3.4495990039324967e-06, 'max_depth': 32, 'subsample': 0.7077102132281233, 'colsample_bynode': 0.69410981
```

In [392]:
```
best_trial = study.best_trial
print("Accuracy : ",best_trial.value)
best_trial.params
```

```
Accuracy :  0.880475652173913
```

Out[392]:
```
{'learning_rate': 3.1840786124748217e-06,
 'max_depth': 28,
 'subsample': 0.7103658428153488,
 'colsample_bynode': 0.7474094734990366,
 'lambda': 1.4841825190094607e-08,
 'alpha': 0.7459600831468086,
 'gamma': 3.105407613303861e-05}
```

In [393]:
```
n_iter[best_trial.number]
```

Out[393]:
```
69
```

In [394]:
```
xgb_clf = xgb.XGBClassifier(**best_trial.params,n_estimators=n_iter[best_trial.number],random_state =0)
xgb_clf
```

Out[394]:
```
XGBClassifier(alpha=0.7459600831468086, base_score=None, booster=None,
              colsample_bylevel=None, colsample_bynode=0.7474094734990366,
              colsample_bytree=None, enable_categorical=False,
              gamma=3.105407613303861e-05, gpu_id=None, importance_type=None,
              interaction_constraints=None, lambda=1.4841825190094607e-08,
              learning_rate=3.1840786124748217e-06, max_delta_step=None,
              max_depth=28, min_child_weight=None, missing=nan,
              monotone_constraints=None, n_estimators=69, n_jobs=None,
              num_parallel_tree=None, predictor=None, random_state=0,
              reg_alpha=None, reg_lambda=None, scale_pos_weight=None,
              subsample=0.7103658428153488, tree_method=None,
              validate_parameters=None, verbosity=None)
```

In [395]:
```
xgb_clf.fit(X_train,y_train)
```

```
[21:36:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'er
ror' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

D:\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass optio
n use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers st
arting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Out[395]:
```
XGBClassifier(alpha=0.7459600831468086, base_score=0.5, booster='gbtree',
              colsample_bylevel=1, colsample_bynode=0.7474094734990366,
              colsample_bytree=1, enable_categorical=False,
              gamma=3.105407613303861e-05, gpu_id=-1, importance_type=None,
              interaction_constraints='', lambda=1.4841825190094607e-08,
              learning_rate=3.1840786124748217e-06, max_delta_step=0,
              max_depth=28, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=69, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0.745960057, reg_lambda=1.48418247e-08,
              scale_pos_weight=1, subsample=0.7103658428153488,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [396]:
```
accuracy = accuracy_score(y_test,xgb_clf.predict(X_test))
accuracy
```

Out[396]:
```
0.7877094972067039
```

In [397]:
```
models['XGBoost'] = accuracy
```

In [398]:
```
models
```

Out[398]:
```
{'Logistic Regression': 0.8379888268156425,
 'Linear SVC': 0.8324022346368715,
 'SVC': 0.8379888268156425,
 'Decision Tree': 0.7597765363128491,
 'Random Forest': 0.8268156424581006,
 'K-Nearest Neighbor': 0.7932960893854749,
 'Gaussian Naive Bayes': 0.659217877094972,
 'XGBoost': 0.7877094972067039}
```

AdaBoost Classifier (Ensemble)

In [399]:
```
adaboost_base = AdaBoostClassifier(random_state=0)
adaboost_base.fit(X_train,y_train)
accuracy = accuracy_score(y_test,adaboost_base.predict(X_test))
accuracy
```

Out[399]:
```
0.8324022346368715
```

In [400]:
```
def objective_adaboost(trial):

  params = {
      'n_estimators':trial.suggest_int('n_estimators',2,200),
      'learning_rate' : trial.suggest_loguniform('learning_rate',1e-6,0.5)
  }

  clf = AdaBoostClassifier(**params,random_state=0)

  cv_score = cross_val_score(clf,X_train, y_train , scoring='accuracy', cv=3, n_jobs=-1,)

  return cv_score.mean()
```

In [401]:
```
study = optuna.create_study(direction='maximize')
study.optimize(objective_adaboost,100)
```

In [402]:
```
best_trial = study.best_trial
best_trial.params
```

Out[402]:
```
{'n_estimators': 97, 'learning_rate': 0.3356974294185799}
```

In [403]:
```
adaboost_clf = AdaBoostClassifier(**best_trial.params,random_state =0).fit(X_train,y_train)
accuracy = accuracy_score(y_test,adaboost_clf.predict(X_test))
accuracy
# 0.8379888268156425
```

Out[403]:
```
0.8435754189944135
```

In [404]:
```
models['AdaBoost Classifier'] = accuracy
models
```

Out[404]:
```
{'Logistic Regression': 0.8379888268156425,
 'Linear SVC': 0.8324022346368715,
 'SVC': 0.8379888268156425,
 'Decision Tree': 0.7597765363128491,
 'Random Forest': 0.8268156424581006,
 'K-Nearest Neighbor': 0.7932960893854749,
 'Gaussian Naive Bayes': 0.659217877094972,
 'XGBoost': 0.7877094972067039,
 'AdaBoost Classifier': 0.8435754189944135}
```

Voting Classifier (Ensemble)

In [405]:
```
votting_clf = VotingClassifier([('Linear SVC',lin_svc),('Logistic Regression',logreg),('SVC',svc),('Random Fores
votting_clf.fit(X_train,y_train)
```

Out[405]:
```
VotingClassifier(estimators=[('Linear SVC', LinearSVC(C=5, random_state=0)),
                             ('Logistic Regression',
                              LogisticRegression(C=0.1, l1_ratio=0.65,
                                                 n_jobs=-1,
                                                 penalty='elasticnet',
                                                 random_state=0,
                                                 solver='saga')),
                             ('SVC',
                              SVC(C=2, kernel='sigmoid', probability=True,
                                  random_state=0)),
                             ('Random Forest',
                              RandomForestClassifier(class_weight='balanced_subsample',
                                                     max_features=0.75,
                                                     max_samples=0.65,
                                                     min_samples_split=0.02,
                                                     random_state=0)),
                             ('K-Nearest Neighbor',
                              GridSearchCV(estimator=KNeighborsClassifier(),
                                           param_grid={'n_neighbors': [2, 3, 4,
                                                                       5, 6, 7,
                                                                       8, 9,
                                                                       10]}))],
                 n_jobs=-1)
```

In [406]:
```
accuracy = accuracy_score(y_test,votting_clf.predict(X_test))
accuracy
```

Out[406]:
```
0.8379888268156425
```

In [407]:
```
votting_clf2 = VotingClassifier([('Logistic Regression',logreg),('SVC',svc),('Random Forest',rf_clf),('K-Nearest
votting_clf2.fit(X_train,y_train)
```

Out[407]:
```
VotingClassifier(estimators=[('Logistic Regression',
                              LogisticRegression(C=0.1, l1_ratio=0.65,
                                                 n_jobs=-1,
                                                 penalty='elasticnet',
                                                 random_state=0,
                                                 solver='saga')),
                             ('SVC',
                              SVC(C=2, kernel='sigmoid', probability=True,
                                  random_state=0)),
                             ('Random Forest',
                              RandomForestClassifier(class_weight='balanced_subsample',
                                                     max_features=0.75,
                                                     max_samples=0.65,
                                                     min_samples_split=0.02,
                                                     random_state=0)),
                             ('K-Nearest Neighbor',
                              GridSearchCV(estimator=KNeighborsClassifier(),
                                           param_grid={'n_neighbors': [2, 3, 4,
                                                                       5, 6, 7,
                                                                       8, 9,
                                                                       10]}))],
                 n_jobs=-1, voting='soft')
```

In [408]:
```
accuracy = accuracy_score(y_test,votting_clf2.predict(X_test))
accuracy
```

Out[408]:
```
0.8324022346368715
```

In [409]:

```
votting_clf3 = VotingClassifier([('SVC',svc),('AdaBoost',adaboost_clf),('XGBoost',xgb_clf),],voting='soft',n_job
votting_clf3.fit(X_train,y_train)
```

Out[409]:

```
VotingClassifier(estimators=[('SVC',
                              SVC(C=2, kernel='sigmoid', probability=True,
                                  random_state=0)),
                             ('AdaBoost',
                              AdaBoostClassifier(learning_rate=0.3356974294185799,
                                                 n_estimators=97,
                                                 random_state=0)),
                             ('XGBoost',
                              XGBClassifier(alpha=0.7459600831468086,
                                            base_score=0.5, booster='gbtree',
                                            colsample_bylevel=1,
                                            colsample_bynode=0.7474094734990366,
                                            colsample_bytree=1,
                                            e...
                                            learning_rate=3.1840786124748217e-06,
                                            max_delta_step=0, max_depth=28,
                                            min_child_weight=1, missing=nan,
                                            monotone_constraints='()',
                                            n_estimators=69, n_jobs=8,
                                            num_parallel_tree=1,
                                            predictor='auto', random_state=0,
                                            reg_alpha=0.745960057,
                                            reg_lambda=1.48418247e-08,
                                            scale_pos_weight=1,
                                            subsample=0.7103658428153488,
                                            tree_method='exact',
                                            validate_parameters=1,
                                            verbosity=None))],
                 n_jobs=-1, voting='soft')
```

In [410]:

```
accuracy = accuracy_score(y_test,votting_clf3.predict(X_test))
accuracy
```

Out[410]:

```
0.8379888268156425
```

Kaggle Submission

In [411]:

```
test_data_raw.shape
```

Out[411]:

```
(418, 11)
```

In [412]:

```
test_data_raw.describe()
```

Out[412]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

In [413]:

```
X_test_data = clean_data(test_data_raw)
X_test_data = pipe.transform(X_test_data)
```

In [414]:

```
vot_clf1_result = votting_clf.predict(X_test_data).astype(int)
vot_clf2_result = votting_clf2.predict(X_test_data).astype(int)
vot_clf3_result = votting_clf3.predict(X_test_data).astype(int)
```

In [415]:

```
svc_result = svc.predict(X_test_data).astype(int)
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': svc_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_svc.csv', index =False)
```

In [416]:

```
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': vot_clf1_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_vot_clf1.csv', index =False)
```

In [417]:

```
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': vot_clf2_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_vot_clf2.csv', index =False)
```

In [418]:

```
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': vot_clf3_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_vot_clf3.csv', index =False)
```

In [419]:

```
logreg_result = logreg.predict(X_test_data).astype(int)
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': logreg_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_logreg.csv', index =False)
```

In [420]:

```
lin_SVC_result = lin_svc.predict(X_test_data).astype(int)
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': lin_SVC_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_linSVC.csv', index =False)
```

In [421]:

```
xgboost_result = xgb_clf.predict(X_test_data).astype(int)
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': xgboost_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_xgboost.csv', index =False)
```

In [422]:

```
adaboost_result = adaboost_clf.predict(X_test_data).astype(int)
sub_data = {'PassengerId': test_data_raw.PassengerId, 'Survived': adaboost_result}
submission = pd.DataFrame(data=sub_data)
submission.to_csv('submission_adaboost_result.csv', index =False)
```