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

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
import random
seed = 42
```

In [4]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

1.Data Analysis

In [5]:
```
train.sample(7)
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
359	360	1	3	Mockler, Miss. Helen Mary "Ellie"	female	NaN	0	0	330980	7.8792	N
804	805	1	3	Hedman, Mr. Oskar Arvid	male	27.0	0	0	347089	6.9750	N
35	36	0	1	Holverson, Mr. Alexander Oskar	male	42.0	1	0	113789	52.0000	N
584	585	0	3	Paulner, Mr. Uscher	male	NaN	0	0	3411	8.7125	N
834	835	0	3	Allum, Mr. Owen George	male	18.0	0	0	2223	8.3000	N
516	517	1	2	Lemore, Mrs. (Amelia Milley)	female	34.0	0	0	C.A. 34260	10.5000	F
238	239	0	2	Pengelly, Mr. Frederick William	male	19.0	0	0	28665	10.5000	N

In [6]:
```
test.sample(7)
```

Out[6]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
55	947	3	Rice, Master. Albert	male	10.0	4	1	382652	29.1250	NaN	
267	1159	3	Warren, Mr. Charles William	male	NaN	0	0	C.A. 49867	7.5500	NaN	
141	1033	1	Daniels, Miss. Sarah	female	33.0	0	0	113781	151.5500	NaN	
248	1140	2	Hold, Mrs. Stephen (Annie Margaret Hill)	female	29.0	1	0	26707	26.0000	NaN	
37	929	3	Cacic, Miss. Manda	female	21.0	0	0	315087	8.6625	NaN	
327	1219	1	Rosenshine, Mr. George (Mr George Thorne")"	male	46.0	0	0	PC 17585	79.2000	NaN	
330	1222	2	Davies, Mrs. John Morgan (Elizabeth Agnes Mary...	female	48.0	0	2	C.A. 33112	36.7500	NaN	

In [7]:
```
train.shape, test.shape
```

Out[7]:
```
((891, 12), (418, 11))
```

In [8]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [9]:
```
test.info()
```

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

In [10]:
```
train.isnull().sum()
```

Out[10]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [11]:
```
test.isnull().sum()
```

Out[11]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [12]:
```
#Almost 80% of the data in cabin is null; can be removed but will perform more
#Age also has missing data
#Embarked is null for 2 rows in the test data
#PassengerID, Ticket are just ref numbers; can be removed
#SibSp, Parch - Maybe we can combine both as one single column as 'family_membe
```

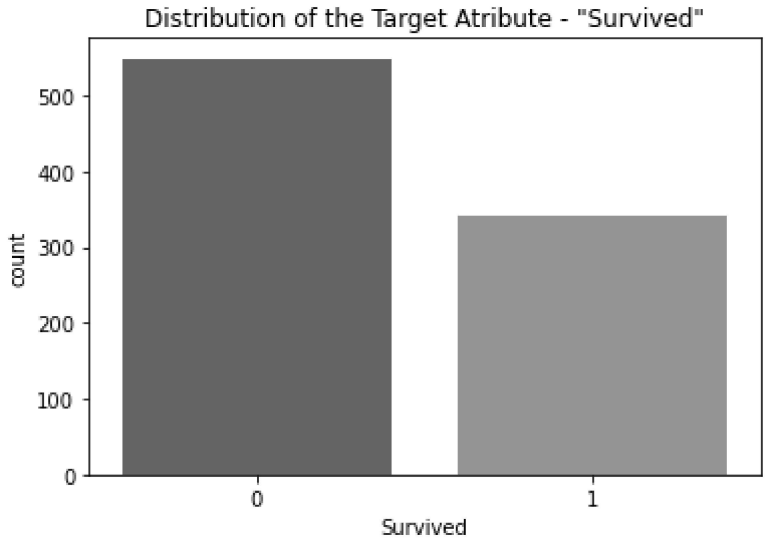
In [13]:
```
train.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
PassengerId	891.0	446.000000	257.353842	1.00	223.5000	446.0000	668.5	891.0000
Survived	891.0	0.383838	0.486592	0.00	0.0000	0.0000	1.0	1.0000
Pclass	891.0	2.308642	0.836071	1.00	2.0000	3.0000	3.0	3.0000
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

1.2 Data Visualization

In [14]:
```
#Distribution of the target

sns.countplot(x='Survived', data=train)
plt.title('Distribution of the Target Atribute - "Survived"')
plt.show()
```



In [15]:
```
#Analyzing the target variable

print('Number of occurances: ')
print(train['Survived'].value_counts())

print('\nPercentage of occurances: ')
print(train['Survived'].value_counts(normalize=True) * 100)
```

```
Number of occurances:
0    549
1    342
Name: Survived, dtype: int64

Percentage of occurances:
0    61.616162
1    38.383838
Name: Survived, dtype: float64
```

In [16]:
```
train['SibSp'].value_counts()
```

Out[16]:
```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

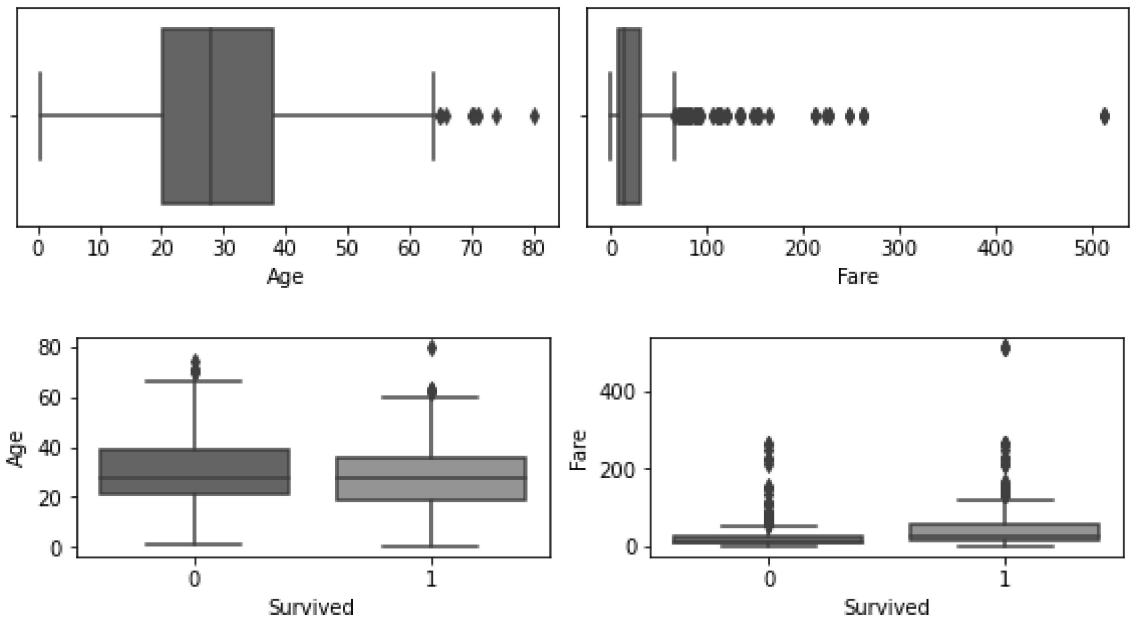
In [17]:
```
train['Parch'].value_counts()
```

Out[17]:
```
0    678
1    118
2     80
3      5
5      5
4      4
6      1
Name: Parch, dtype: int64
```

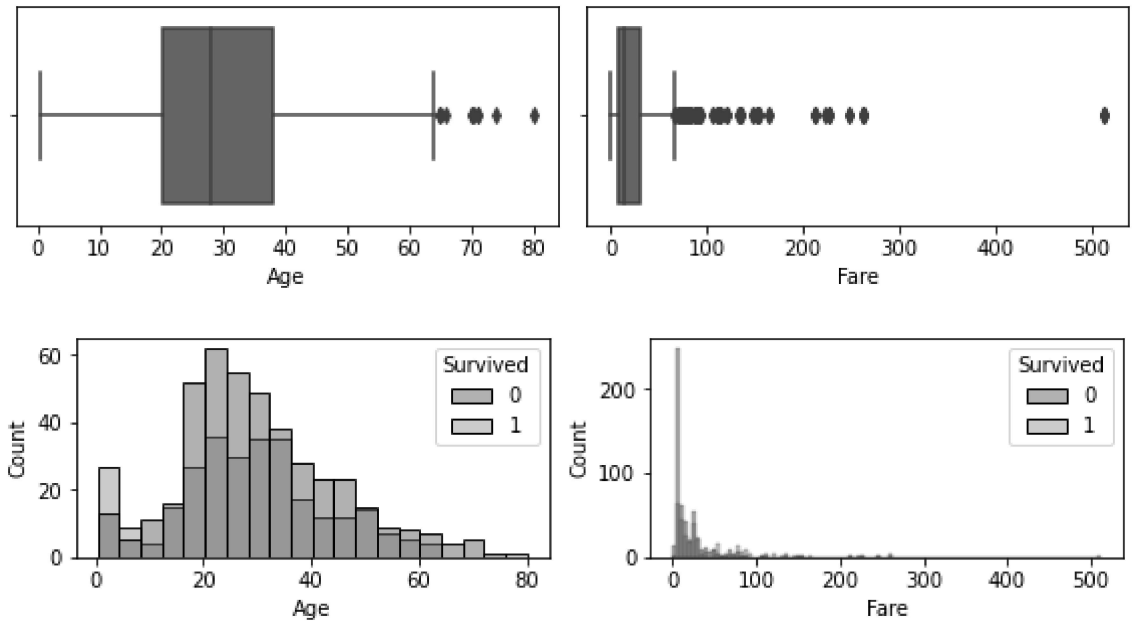
In [18]:
```
#Both 'SibSp' and 'Parch' can be considered as categories

numeric_cols = ['Age', 'Fare']
category_cols = ['Pclass', 'Sex', 'Cabin', 'SibSp', 'Parch', 'Embarked']
```

In [19]:
```
def dist_plot(df, columns, type='boxplot', label=None):
    plt.figure(figsize=(12,6))
    for idx, var in enumerate(columns):
        plt.subplot(3, 3, idx + 1)
        if (type == 'boxplot'):
            if not label:
                g = sns.boxplot(x=var, data=df, showfliers=True)
            else:
                g = sns.boxplot(y=var, data=df, showfliers=True, x=label)
        elif (type=='histplot'):
            if not label:
                g = sns.histplot(x=var, data=df)
            else:
                g = sns.histplot(x=var, data=df, hue=label)
    plt.tight_layout()

def count_plot(df, columns, label=None):
    plt.figure(figsize=(12, 6))
    for idx, var in enumerate(columns):
        plt.subplot(3, 3, idx + 1)
        if not label:
            g = sns.countplot(x=var, data=df)
        else:
            g = sns.countplot(x=var, data=df, hue = label)
    plt.tight_layout()
```

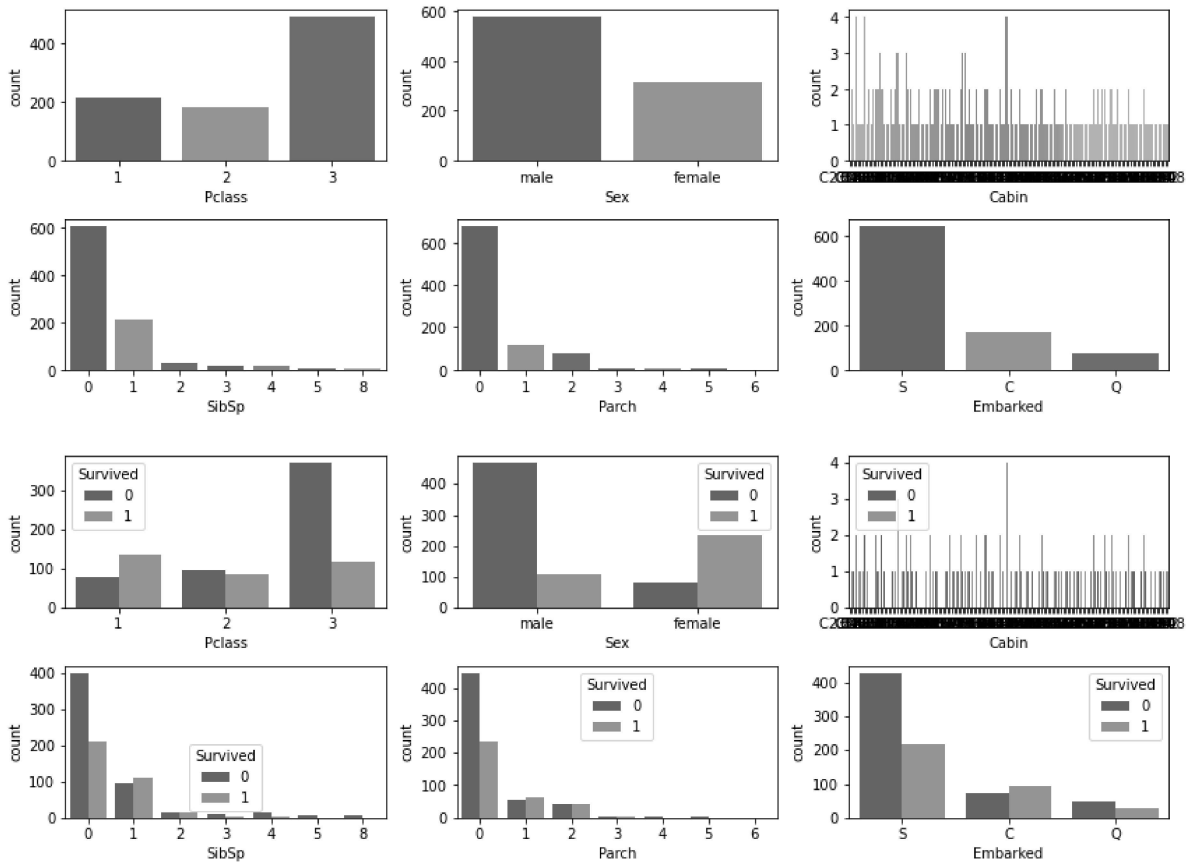
In [20]:
```
dist_plot(train, numeric_cols)
dist_plot(train, numeric_cols, label='Survived')
```



In [21]:
```
dist_plot(train, numeric_cols)
dist_plot(train, numeric_cols, label='Survived', type='histplot')
```



In [22]:
```
count_plot(train, category_cols)
count_plot(train, category_cols, label='Survived')
```



Initial Observations

- *Among survived individuals, male to female ratio is almost 1:2 i.e., double female survived compared to male.*
- *Majority of the deceased passengers are from Pclass 3; who probably don't have a cabin? Will further analyze.*
- *Looks like if the pasenger embarked his/her journey from Cherbourg, there is a lot of chance of survival. Need to check more here.*
- *Passenger deceased are majorly within the age range of 20-40, however the survived count is also better there as more passengers are from that age range. Lets try to analyze it further.*
- *The data for fare is highly skewed.*
- *Will spare a few more mins to understand if Cabin can be utilized.*

In [23]:
```
def percentage_calc(df, by, agg):
    temp = df.groupby([by])[agg].transform('sum')
    pct = round((df[agg]/temp) * 100, 2).astype('str') + '%'
    return pct

def create_group(df, grp_cols, agg_func='count', temp = 'PassengerId'):
    df_grp = df.groupby(grp_cols).aggregate({temp: agg_func}).reset_index()
    grp_cols.append('Agg')
    df_grp.columns = grp_cols
    return df_grp
```

Male vs Female Survival Stats

In [24]:
```
train_sex_grp = create_group(train, ['Sex', 'Survived'])
train_sex_grp['PCT'] = percentage_calc(train_sex_grp, 'Sex', 'Agg')

train_sex_grp
```

Out[24]:

	Sex	Survived	Agg	PCT
0	female	0	81	25.8%
1	female	1	233	74.2%
2	male	0	468	81.11%
3	male	1	109	18.89%

Hence, it is proved here - almost 75% of female survived the mishap whereas more than 80% men were died.

Survival stats on the basis of Pclass

In [25]:
```
train_class_grp = create_group(train, ['Pclass', 'Survived', 'Sex'])
train_class_grp['PCT'] = percentage_calc(train_class_grp, 'Pclass', 'Agg')

train_class_grp
```

Out[25]:

	Pclass	Survived	Sex	Agg	PCT
0	1	0	female	3	1.39%
1	1	0	male	77	35.65%
2	1	1	female	91	42.13%
3	1	1	male	45	20.83%
4	2	0	female	6	3.26%
5	2	0	male	91	49.46%
6	2	1	female	70	38.04%
7	2	1	male	17	9.24%
8	3	0	female	72	14.66%
9	3	0	male	300	61.1%
10	3	1	female	72	14.66%
11	3	1	male	47	9.57%

- *Most of the deceased individuals are from lower class - 25% of all deaths.*
- *More than 60% of the survived passengers boarded are from upper class.*
- *More than 40% of all survived individuals from upper class are female.*
- *Similar trend is observed for middle class as well; almost 40% of survived passengers are female.*
- *Even in lower class, almost 15% of survived passengers are female.*

Hence being female and if boarded one of Pclass 1 & 2 then survival percentage increases significantly.

Studying Embarkation point

In [26]:
```
train_emb_grp = create_group(train, ['Embarked', 'Survived', 'Sex', 'Pclass'])
train_emb_grp['PCT'] = percentage_calc(train_emb_grp, 'Embarked', 'Agg')

train_emb_grp
```

Out[26]:

	Embarked	Survived	Sex	Pclass	Agg	PCT
0	C	0	female	1	1	0.6%
1	C	0	female	3	8	4.76%
2	C	0	male	1	25	14.88%
3	C	0	male	2	8	4.76%
4	C	0	male	3	33	19.64%
5	C	1	female	1	42	25.0%
6	C	1	female	2	7	4.17%
7	C	1	female	3	15	8.93%
8	C	1	male	1	17	10.12%
9	C	1	male	2	2	1.19%
10	C	1	male	3	10	5.95%
11	Q	0	female	3	9	11.69%
12	Q	0	male	1	1	1.3%
13	Q	0	male	2	1	1.3%
14	Q	0	male	3	36	46.75%
15	Q	1	female	1	1	1.3%
16	Q	1	female	2	2	2.6%
17	Q	1	female	3	24	31.17%
18	Q	1	male	3	3	3.9%
19	S	0	female	1	2	0.31%
20	S	0	female	2	6	0.93%
21	S	0	female	3	55	8.54%
22	S	0	male	1	51	7.92%
23	S	0	male	2	82	12.73%
24	S	0	male	3	231	35.87%
25	S	1	female	1	46	7.14%
26	S	1	female	2	61	9.47%
27	S	1	female	3	33	5.12%
28	S	1	male	1	28	4.35%
29	S	1	male	2	15	2.33%
30	S	1	male	3	34	5.28%

Cherbourg

- *About 5% female passengers died whereas almost 40% of men were casualties of the mishap while boarding from Cherbourg.*
- *Survival chances gets increased when a female is from upper class passengers. However, the same is not true for males - almost 15% of deceased men were from upper class.*
- *All female travelers from middle class section survived the mishap.*

Queensland

- *Not a single female died from upper or middle class section who started their journey from Queensland.*
- *Lower class males died in numbers - more than 30% whereas over 45% lower class female survived.*

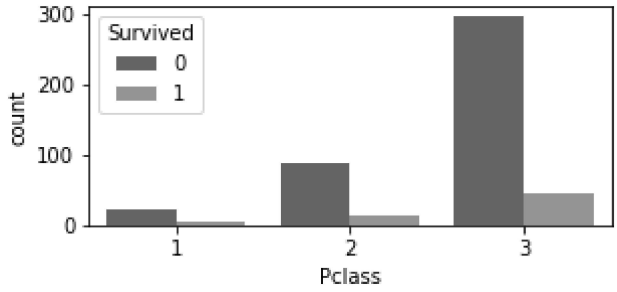
Southampton

- *Male passengers died in numbers, around 36% are from lower class whereas very minimal from upper class individuals.*
- *347 males from lower class started their journey from Southampton i.e., almost 40% of total travellers.*

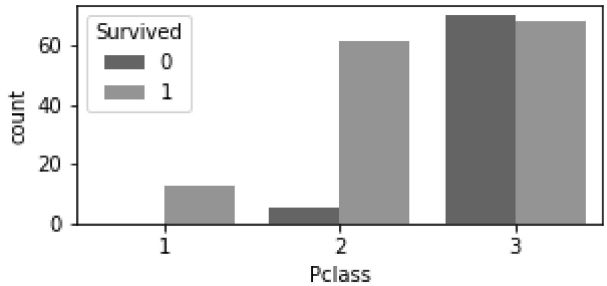
For males from lower class section significantly rises the probability of death from the accident.

Lets analyze the column for cabin

In [27]:
```
cabin_null = train.loc[(train['Cabin'].isnull()) & (train['Sex'] == 'male')]

count_plot(cabin_null, ['Pclass'], label='Survived')
```



In [28]:
```
cabin_null = train.loc[(train['Cabin'].isnull()) & (train['Sex'] == 'female')]

count_plot(cabin_null, ['Pclass'], label='Survived')
```



In [29]:
```
df_cabin = train.copy()

cabins = df_cabin['Cabin']
df_cabin['Cabin_Series'] = [str(cabin)[0] for cabin in cabins]

df_cabin_surv = df_cabin.loc[df_cabin['Survived'] == 1][['Cabin_Series', 'Pcla

print(df_cabin_surv['Cabin_Series'].value_counts(normalize=True))
```

```
n    0.602339
C    0.102339
B    0.102339
D    0.073099
E    0.070175
F    0.023392
A    0.020468
G    0.005848
Name: Cabin_Series, dtype: float64
```

In [30]:
```
df_cabin_class = create_group(df_cabin, ['Cabin_Series', 'Pclass', 'Sex'])
df_cabin_class
```

Out[30]:

	Cabin_Series	Pclass	Sex	Agg
0	A	1	female	1
1	A	1	male	14
2	B	1	female	27
3	B	1	male	20
4	C	1	female	27
5	C	1	male	32
6	D	1	female	16
7	D	1	male	13
8	D	2	female	2
9	D	2	male	2
10	E	1	female	10
11	E	1	male	15
12	E	2	female	4
13	E	3	female	1
14	E	3	male	2
15	F	2	female	4
16	F	2	male	4
17	F	3	female	1
18	F	3	male	4
19	G	3	female	4
20	T	1	male	1
21	n	1	female	13
22	n	1	male	27
23	n	2	female	66
24	n	2	male	102
25	n	3	female	138
26	n	3	male	341

- *Wherever we have cabin as null, majority of them are males and from lower class.*
- *Does that mean, the individuals with cabin as null has no dedicated cabin?*
- *Cabins with null values are less for female. If we assume the previous point as true then is this an indicative reason for a higher survival rates among females compared to males?*
- *Without cabin reduces survival chances.*
- *A series has low survival percentage - less women count*
- *Maybe we can set a flag (y/n) for having cabin or not.*

Analyzing the column for Fare

In [31]:
```
train.loc[train['Fare'] == 0]
```

Out[31]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
179	180	0	3	Leonard, Mr. Lionel	male	36.0	0	0	LINE	0.0	NaN
263	264	0	1	Harrison, Mr. William	male	40.0	0	0	112059	0.0	B94
271	272	1	3	Tornquist, Mr. William Henry	male	25.0	0	0	LINE	0.0	NaN
277	278	0	2	Parkes, Mr. Francis "Frank"	male	NaN	0	0	239853	0.0	NaN
302	303	0	3	Johnson, Mr. William Cahoone Jr	male	19.0	0	0	LINE	0.0	NaN
413	414	0	2	Cunningham, Mr. Alfred Fleming	male	NaN	0	0	239853	0.0	NaN
466	467	0	2	Campbell, Mr. William	male	NaN	0	0	239853	0.0	NaN
481	482	0	2	Frost, Mr. Anthony Wood "Archie"	male	NaN	0	0	239854	0.0	NaN
597	598	0	3	Johnson, Mr. Alfred	male	49.0	0	0	LINE	0.0	NaN
633	634	0	1	Parr, Mr. William Henry Marsh	male	NaN	0	0	112052	0.0	NaN
674	675	0	2	Watson, Mr. Ennis Hastings	male	NaN	0	0	239856	0.0	NaN
732	733	0	2	Knight, Mr. Robert J	male	NaN	0	0	239855	0.0	NaN
806	807	0	1	Andrews, Mr. Thomas Jr	male	39.0	0	0	112050	0.0	A36
815	816	0	1	Fry, Mr. Richard	male	NaN	0	0	112058	0.0	B102
822	823	0	1	Reuchlin, Jonkheer. John George	male	38.0	0	0	19972	0.0	NaN

In [32]:
```
train_fare_grp = create_group(train, ['Pclass'], agg_func = 'mean', temp='Fare

train_fare_grp.columns = ['Pclass', 'Mean_Fare']
train_fare_grp
```

Out[32]:

	Pclass	Mean_Fare
0	1	84.154687
1	2	20.662183
2	3	13.675550

There are few rows where Fare=0 ; lets put mean value as per their class.

Analyzing age of passengers

In [33]:
```
train_age_class = create_group(train, ['Pclass'], agg_func='mean', temp='Age')

train_age_class.columns = ['Pclass', 'Mean_Age']
train_age_class
```

Out[33]:

	Pclass	Mean_Age
0	1	38.233441
1	2	29.877630
2	3	25.140620

In [34]:
```
train_age_grp = create_group(train, ['Pclass', 'Sex', 'Survived'], agg_func='m

train_age_grp.columns = ['Pclass', 'Sex', 'Survived', 'Mean_Age']
train_age_grp
```

Out[34]:

	Pclass	Sex	Survived	Mean_Age
0	1	female	0	25.666667
1	1	female	1	34.939024
2	1	male	0	44.581967
3	1	male	1	36.248000
4	2	female	0	36.000000
5	2	female	1	28.080882
6	2	male	0	33.369048
7	2	male	1	16.022000
8	3	female	0	23.818182
9	3	female	1	19.329787
10	3	male	0	27.255814
11	3	male	1	22.274211

Among upper class passengers, mostly older female survived where as for other classes majority of the younger generation made the cut regardless of their gender.

Checking Sibsp & Parch

In [35]:
```
train_Sib = create_group(train, ['SibSp', 'Survived'])
train_Sib
```

Out[35]:

	SibSp	Survived	Agg
0	0	0	398
1	0	1	210
2	1	0	97
3	1	1	112
4	2	0	15
5	2	1	13
6	3	0	12
7	3	1	4
8	4	0	15
9	4	1	3
10	5	0	5
11	8	0	7

Less sibling increases the survival rate.

In [36]:
```
train_Par = create_group(train, ['Parch', 'Survived'])
train_Par
```

Out[36]:

	Parch	Survived	Agg
0	0	0	445
1	0	1	233
2	1	0	53
3	1	1	65
4	2	0	40
5	2	1	40
6	3	0	2
7	3	1	3
8	4	0	4
9	5	0	4
10	5	1	1
11	6	0	1

Almost similar trend is visible for Parch. Can be created a single feature combining both these columns.

Discovered Information

- Upper & middle class females has a better survival rates than others.
- Passengers with less family members has a better stats for survival.
- Considering missing cabin value as without a cabin and it decreases chance of survival.
- Lower class individuals have less chance of survival and most lower class people have embarked their journey from Southampton.

Actions to be taken

1. Remove 'PassengerID', 'Ticket' , 'Name' columns.
2. Create a new feature "Cabin_Flag" with value as 1 & 0 for NaN and values respectively. Then remove 'Cabin'.
3. 'Fare' with 0 should be updated with mean per their respective class.
4. Missing values for 'Emabarked' can be replaced with mode.
5. Missing 'Age' values should be replaced with mean.
6. Create a new feature 'Family' and aggregate the values from 'SibSp' & 'Parch'. Then remove 'SibSp' & 'Parch'.
7. Treat the outliers in 'Age' & 'Fare'.
8. Create dummies for 'Sex' and 'Embarked'.

2. Data Preprocessing

In [37]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.preprocessing import KBinsDiscretizer, FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer, make_column_selector
#from sklearn.pipeline import Pipeline
```

In [38]:
```
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek
from imblearn.pipeline import Pipeline
from imblearn.under_sampling import TomekLinks
```

Getting X and y seperated

In [39]:
```
y_train = train['Survived']
X_train = train.drop(['Survived'], axis=1)
```

In [40]:
```
X_train.head()
```

Out[40]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [41]:
```
y_train
```

Out[41]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

In [42]:
```
#cat_columns = ['PassengerId', 'Pclass', 'Name', 'Sex', 'SibSp', 'Parch', 'Tic
num_columns = ['Age', 'Fare', 'Parch', 'SibSp']
```

Treating the ouliers

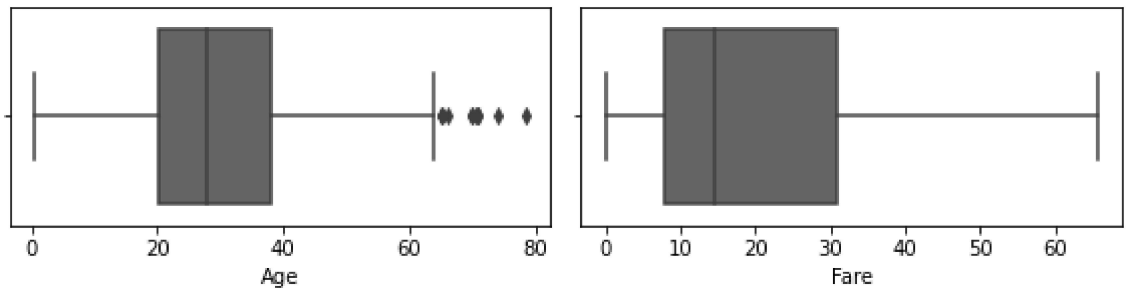
In [43]:
```
def outlier_treatment(X, columns):
    new_X = X.copy()
    X_treat = new_X[num_columns].fillna(value=0)
    for column in columns:
        Q1 = np.percentile(X_treat[column], 25.)
        Q2 = np.percentile(X_treat[column], 50.)
        Q3 = np.percentile(X_treat[column], 75.)

        cut_off = (Q3-Q1) * 1.5
        upper, lower = Q3 + cut_off, Q1 - cut_off
        print(upper, lower)
        new_X.loc[(new_X[column] < lower), column] = lower
        new_X.loc[(new_X[column] > upper), column] = upper

    return new_X
```

In [44]:
```
#X_train[num_columns] = X_train[num_columns].fillna(value=0) #Filling all NaN v
X_train.isnull().sum()
```

Out[44]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [45]:
```
train_out = outlier_treatment(X_train, ['Age', 'Fare'])
dist_plot(train_out, ['Age', 'Fare'])
```

```
78.5 -37.5
65.6344 -26.724
```



In [46]:
```
train_out
```

Out[46]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	65.6344	C85	
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	
...	...	...	...	...	...	...	...	...	...	...	
886	887	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	
887	888	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	
888	889	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	
889	890	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	
890	891	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	

891 rows × 11 columns

Removing and creating applicable columns

In [47]:
```
train_out.isnull().sum()
```

Out[47]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [48]:
```
class CategoryTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.cols_to_remove = ['PassengerId', 'Ticket', 'Name']
        self.attr_combo = ['Cabin', 'SibSp', 'Parch']
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        temp = X.copy()
        temp['Cabin_Flag'] = temp['Cabin'].apply(lambda x: 0 if x is np.nan els
        temp['Family'] = temp['SibSp'] + temp['Parch']
        for i in range(len(self.attr_combo)):
            self.cols_to_remove.append(self.attr_combo[i])
        new_X = temp.drop(self.cols_to_remove, axis=1)
        other_cols = ['Age', 'Fare', 'Embarked', 'Sex']
        trans_X = new_X.drop(other_cols, axis=1)
        return trans_X.to_numpy()
```

In [49]:
```
cat_trans = CategoryTransformer()
train_attr = cat_trans.transform(train_out)

train_attr
```

Out[49]:
```
array([[3, 0, 1],
       [1, 1, 1],
       [3, 0, 0],
       ...,
       [3, 0, 3],
       [1, 1, 0],
       [3, 0, 0]], dtype=int64)
```

Data Pipeline creation

1. Imputing missing values for numeric columns - Age, Fare
2. Unwanted feature removal & new columns creation
3. OneHotEncoding for Embarked & Sex

In [50]:
```
num_cols = ['Age', 'Fare']
cat_cols = ['Embarked', 'Sex']
all_cols = list(train_out.columns)
```

In [51]:
```
imputer1 = SimpleImputer(missing_values=0, strategy='mean')
imputer2 = SimpleImputer(strategy='mean')
imputer3 = SimpleImputer(strategy='most_frequent')

encoder = OneHotEncoder(dtype = 'int')
transformer = CategoryTransformer()
```

In [52]:
```
num_pipeline = Pipeline([('imputer2', SimpleImputer(strategy='mean')),
                         ('imputer1', SimpleImputer(missing_values=0, strategy
                        ])

clean_pipeline = Pipeline([('transformer', CategoryTransformer())])

cat_pipeline = Pipeline([('imputer3', SimpleImputer(strategy='most_frequent'))
                         ('encoder', OneHotEncoder(dtype = 'int'))])


data_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_cols),
    ('clean', clean_pipeline, all_cols),
    ('cat', cat_pipeline, cat_cols)
])

data_pipeline
```

Out[52]:
```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer2', SimpleImputer
()),
                                                 ('imputer1',
                                                  SimpleImputer(missing_value
s=0))]),
                                 ['Age', 'Fare']),
                                ('clean',
                                 Pipeline(steps=[('transformer',
                                                  CategoryTransformer())]),
                                 ['PassengerId', 'Pclass', 'Name', 'Sex', 'Ag
e',
                                  'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabi
n',
                                  'Embarked']),
                                ('cat',
                                 Pipeline(steps=[('imputer3',
                                                  SimpleImputer(strategy='mos
t_frequent')),
                                                 ('encoder',
                                                  OneHotEncoder(dtype='in
t'))]),
                                 ['Embarked', 'Sex'])])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:
```
# num_pipeline = Pipeline([('imputer1', SimpleImputer(missing_values=0, strate
#                         ('imputer2', SimpleImputer(strategy='mean'))])

# clean_pipeline = Pipeline([('transformer', CategoryTransformer())])

# cat_pipeline = Pipeline([('imputer3', SimpleImputer(strategy='most_frequent'
#                          ('encoder', OneHotEncoder(dtype = 'int'))])


# data_pipeline = ColumnTransformer([
#     ('num', num_pipeline, num_cols),
#     ('clean', clean_pipeline, all_cols),
#     ('cat', cat_pipeline, cat_cols)
# ])

# data_pipeline
```

In [54]:
```
data_prepared = data_pipeline.fit_transform(train_out)

data_prepared
```

Out[54]:
```
array([[22.        ,  7.25      ,  3.        , ...,  1.        ,
         0.        ,  1.        ],
       [38.        , 65.6344    ,  1.        , ...,  0.        ,
         1.        ,  0.        ],
       [26.        ,  7.925     ,  3.        , ...,  1.        ,
         1.        ,  0.        ],
       ...,
       [29.69701681, 23.45      ,  3.        , ...,  1.        ,
         1.        ,  0.        ],
       [26.        , 30.        ,  1.        , ...,  0.        ,
         0.        ,  1.        ],
       [32.        ,  7.75      ,  3.        , ...,  0.        ,
         0.        ,  1.        ]])
```

In [55]:
```
data_prepared[0]
```

Out[55]:
```
array([22.  ,  7.25,  3.  ,  0.  ,  1.  ,  0.  ,  0.  ,  1.  ,  0.  ,
       1.  ])
```

In [56]:
```
smote = SMOTE()
smotek = SMOTETomek()

X_smote, y_smote = smote.fit_resample(data_prepared, y_train)

X_smotek, y_smotek = smotek.fit_resample(data_prepared, y_train)
```

3.Model Evaluation

In [57]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

In [58]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from imblearn.ensemble import BalancedBaggingClassifier
from imblearn.ensemble import BalancedRandomForestClassifier
```

In [59]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

In [60]:
```
def model_set():
    model_set = []

    model_set.append(('LR', LogisticRegression()))
    model_set.append(('DT', DecisionTreeClassifier(random_state=seed)))
    model_set.append(('RF', RandomForestClassifier(random_state=seed)))
    model_set.append(('SV', SVC(random_state=seed)))
    model_set.append(('EX', ExtraTreesClassifier(random_state=seed)))
    model_set.append(('KN', KNeighborsClassifier()))
    model_set.append(('GB', GradientBoostingClassifier(random_state=seed)))
    model_set.append(('AB', AdaBoostClassifier(random_state=seed)))
    model_set.append(('NB', GaussianNB()))
    model_set.append(('BRF', BalancedRandomForestClassifier(random_state=seed)

    return model_set
```

In [61]:
```
def score_models(X, y, modelset, scale = None):
    cols = ['Model', 'Accuracy Mean', 'Accuracy STD', 'F1 Score Mean', 'F1 Sco

    models = []

    accuracy_mean = []
    accuracy_std = []

    f1_score_mean = []
    f1_score_std = []

    if scale == 'MinMax':
        scale = MinMaxScaler()
        X = scale.fit_transform(X)
    if scale == 'Standard':
        scale = StandardScaler()
        X = scale.fit_transform(X)

    for name, model in modelset:
        kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
        cv_acc = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
        cv_f1 = cross_val_score(model, X, y, cv=kfold, scoring='f1')

        models.append(model.__class__.__name__)

        accuracy_mean.append(cv_acc.mean())
        accuracy_std.append(cv_acc.std())

        f1_score_mean.append(cv_f1.mean())
        f1_score_std.append(cv_f1.std())


    score_matrix = list(zip(models, accuracy_mean, accuracy_std, f1_score_mean
    df = pd.DataFrame(data = score_matrix, columns=cols)
    return df
```

Base Model evaluation scores without resampling

In [62]:
```
models = model_set()
score_models(data_prepared, y_train, models)
```

Out[62]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.799064	0.037585	0.730350	0.058506
1	DecisionTreeClassifier	0.765331	0.045112	0.702622	0.053360
2	RandomForestClassifier	0.815918	0.025841	0.756316	0.042669
3	SVC	0.691311	0.029392	0.482603	0.056587
4	ExtraTreesClassifier	0.795693	0.033083	0.730129	0.048515
5	KNeighborsClassifier	0.723945	0.037999	0.628993	0.049982
6	GradientBoostingClassifier	0.827116	0.023920	0.756547	0.039828
7	AdaBoostClassifier	0.805768	0.030879	0.739623	0.047969
8	GaussianNB	0.765431	0.038417	0.715620	0.039644
9	BalancedRandomForestClassifier	0.802422	0.029201	0.751337	0.046309

Base Model evaluation with SMOTE

In [63]:
```
score_models(X_smote, y_smote, models)
```

Out[63]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.788749	0.022972	0.784768	0.026128
1	DecisionTreeClassifier	0.810601	0.034161	0.809841	0.034064
2	RandomForestClassifier	0.847957	0.024222	0.846425	0.025806
3	SVC	0.661176	0.047051	0.620726	0.058292
4	ExtraTreesClassifier	0.834254	0.025703	0.833783	0.024486
5	KNeighborsClassifier	0.770517	0.030191	0.780005	0.031287
6	GradientBoostingClassifier	0.848849	0.030432	0.842808	0.033815
7	AdaBoostClassifier	0.817882	0.039681	0.816507	0.039873
8	GaussianNB	0.765046	0.032393	0.768306	0.029077
9	BalancedRandomForestClassifier	0.846138	0.027528	0.844098	0.030377

Base Model evaluation with SMOTETomek

In [64]:
```
score_models(X_smotek, y_smotek, models)
```

Out[64]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.809338	0.035629	0.805428	0.039026
1	DecisionTreeClassifier	0.846806	0.030793	0.847844	0.028746
2	RandomForestClassifier	0.871598	0.027680	0.870586	0.029881
3	SVC	0.667065	0.040909	0.640159	0.050203
4	ExtraTreesClassifier	0.856785	0.032544	0.857195	0.033312
5	KNeighborsClassifier	0.760930	0.050260	0.768569	0.047452
6	GradientBoostingClassifier	0.864706	0.027821	0.861515	0.030334
7	AdaBoostClassifier	0.843914	0.028686	0.842802	0.028554
8	GaussianNB	0.790526	0.038196	0.791164	0.040187
9	BalancedRandomForestClassifier	0.868618	0.030445	0.866758	0.033406

Scoring Models with MinMaxScaler without resampling

In [65]:
```
score_models(data_prepared, y_train, models, scale = 'MinMax')
```

Out[65]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.794569	0.029987	0.723052	0.048290
1	DecisionTreeClassifier	0.764207	0.046099	0.700491	0.055904
2	RandomForestClassifier	0.814794	0.028652	0.754390	0.046466
3	SVC	0.810275	0.029737	0.703444	0.061947
4	ExtraTreesClassifier	0.795693	0.033083	0.730129	0.048515
5	KNeighborsClassifier	0.806904	0.024315	0.733069	0.040632
6	GradientBoostingClassifier	0.827116	0.023920	0.756547	0.039828
7	AdaBoostClassifier	0.805768	0.030879	0.739623	0.047969
8	GaussianNB	0.765431	0.038417	0.715620	0.039644
9	BalancedRandomForestClassifier	0.803546	0.029645	0.752382	0.046908

Scoring Models with MinMaxScaler with SMOTE

In [66]:
```
score_models(X_smote, y_smote, models, scale = 'MinMax')
```

Out[66]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.785104	0.026753	0.783762	0.027475
1	DecisionTreeClassifier	0.810601	0.034161	0.809841	0.034064
2	RandomForestClassifier	0.846138	0.024678	0.844379	0.025790
3	SVC	0.806080	0.032318	0.798189	0.034198
4	ExtraTreesClassifier	0.834254	0.025703	0.833783	0.024486
5	KNeighborsClassifier	0.826964	0.029098	0.823673	0.029357
6	GradientBoostingClassifier	0.848849	0.030432	0.842808	0.033815
7	AdaBoostClassifier	0.817882	0.039681	0.816507	0.039873
8	GaussianNB	0.765046	0.032393	0.768306	0.029077
9	BalancedRandomForestClassifier	0.844320	0.026906	0.841647	0.030011

Scoring Models with MinMaxScaler with SMOTETomek

In [67]:
```
score_models(X_smotek, y_smotek, models, scale = 'MinMax')
```

Out[67]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.803397	0.039974	0.800989	0.042914
1	DecisionTreeClassifier	0.846806	0.030793	0.847844	0.028746
2	RandomForestClassifier	0.871598	0.028379	0.870252	0.030944
3	SVC	0.824189	0.041971	0.815653	0.046990
4	ExtraTreesClassifier	0.856785	0.032544	0.857195	0.033312
5	KNeighborsClassifier	0.810348	0.034048	0.805508	0.035507
6	GradientBoostingClassifier	0.864706	0.027821	0.861515	0.030334
7	AdaBoostClassifier	0.843914	0.028686	0.842802	0.028554
8	GaussianNB	0.790526	0.038196	0.791164	0.040187
9	BalancedRandomForestClassifier	0.867628	0.031763	0.865534	0.035104

Scoring Models with StandardScaler without resampling

In [68]:
```
score_models(data_prepared, y_train, models, scale = 'Standard')
```

Out[68]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.799076	0.035815	0.731140	0.056581
1	DecisionTreeClassifier	0.765331	0.045112	0.701325	0.055559
2	RandomForestClassifier	0.815918	0.025841	0.755319	0.044225
3	SVC	0.810275	0.032184	0.720590	0.054237
4	ExtraTreesClassifier	0.795693	0.033083	0.730129	0.048515
5	KNeighborsClassifier	0.803546	0.030485	0.733545	0.042208
6	GradientBoostingClassifier	0.827116	0.023920	0.756547	0.039828
7	AdaBoostClassifier	0.805768	0.030879	0.739623	0.047969
8	GaussianNB	0.765431	0.038417	0.715620	0.039644
9	BalancedRandomForestClassifier	0.802422	0.029201	0.751337	0.046309

Scoring Models with StandardScaler and SMOTE

In [69]:
```
score_models(X_smote, y_smote, models, scale = 'Standard')
```

Out[69]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.786931	0.029668	0.783851	0.031631
1	DecisionTreeClassifier	0.811510	0.033987	0.810585	0.033840
2	RandomForestClassifier	0.846138	0.025011	0.843963	0.027029
3	SVC	0.826080	0.028539	0.820868	0.031379
4	ExtraTreesClassifier	0.834254	0.025703	0.833783	0.024486
5	KNeighborsClassifier	0.829691	0.035625	0.830007	0.035721
6	GradientBoostingClassifier	0.848849	0.030432	0.842808	0.033815
7	AdaBoostClassifier	0.817882	0.039681	0.816507	0.039873
8	GaussianNB	0.765046	0.032393	0.768306	0.029077
9	BalancedRandomForestClassifier	0.847048	0.026102	0.844948	0.028710

Scoring Models with StandardScaler and SMOTETomek

In [70]:
```
score_models(X_smotek, y_smotek, models, scale = 'Standard')
```

Out[70]:

	Model	Accuracy Mean	Accuracy STD	F1 Score Mean	F1 Score STD
0	LogisticRegression	0.810328	0.039469	0.807013	0.042711
1	DecisionTreeClassifier	0.848787	0.028733	0.849373	0.027709
2	RandomForestClassifier	0.874568	0.027307	0.873096	0.029990
3	SVC	0.828140	0.041173	0.820838	0.045000
4	ExtraTreesClassifier	0.856785	0.032544	0.857195	0.033312
5	KNeighborsClassifier	0.816308	0.035341	0.813652	0.035573
6	GradientBoostingClassifier	0.865696	0.027862	0.862369	0.030476
7	AdaBoostClassifier	0.843914	0.028686	0.842802	0.028554
8	GaussianNB	0.790526	0.038196	0.791164	0.040187
9	BalancedRandomForestClassifier	0.870598	0.029576	0.868734	0.032730

- Apart from GradientBoosting, overall SMOTETomek gives better accuracy and F1 Score with MinMaxScaler

In [71]:
```
#Lets take a look on the scores for ensemble as well

#LR + RF + SVC
#EX + KN + GB
#AB + SVC + NB
#KN + AB + GB
#KN + SVC + GB + RF + AB

# def create_ensembles():
#     ensembles = []
#     ensembles.append(('VC1', VotingClassifier([('LR', LogisticRegression()),
#                                                voting='hard')))
#     ensembles.append(('VC2', VotingClassifier([('EX', ExtraTreesClassifier()),
#                                                voting='hard')))
#     ensembles.append(('VC3', VotingClassifier([('AB', AdaBoostClassifier()),
#                                                voting='hard')))
#     ensembles.append(('VC4', VotingClassifier([('KN', KNeighborsClassifier()),
#                                                voting='hard')))
#     ensembles.append(('VC5', VotingClassifier([('KN', KNeighborsClassifier()),
#                                                ('AB', AdaBoostClassifier())],
#     return ensembles
```

Ensemble with default scaling

In [72]:
```
# ensemble_list = create_ensembles()
# score_models(data_prepared, y_train, ensemble_list)
```

Ensemble with scaled data

In [73]:
```
# score_models(data_prepared, y_train, ensemble_list, scale = 'MinMax')
```

In [74]:
```
# score_models(data_prepared, y_train, ensemble_list, scale = 'Standard')
```

- Lets scale the date with MinMaxScaler with SMOTETomek
- Promising algorithms:
  - GradientBoostingClassfier
  - RandomForestClassifier
  - AdaBoostClassfier
  - SVC
  - KNeighborsClassifier
  - BalancedBaggingClassifier
  - VotingClassifier(with the above models)

4. Model Selection

In [75]:
```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_smotek)
```

In [76]:
```
models = []

gb = GradientBoostingClassifier(random_state=seed)
models.append(gb)

rf = RandomForestClassifier(random_state=seed)
models.append(rf)

ab = AdaBoostClassifier(random_state=seed)
models.append(ab)

sv = SVC(random_state=seed)
models.append(sv)

kn = KNeighborsClassifier()
models.append(kn)

brf = BalancedRandomForestClassifier(random_state=seed)
models.append(brf)


param_grid = [

    {
        'loss': ['log_loss', 'exponential'],
        'n_estimators': [100, 200, 400, 500, 600],
        'max_features': [2, 3, 4],
        'learning_rate': [0.1, 0.5, 1]
    },
    {
        'n_estimators': [100, 200, 400, 500, 600],
        'max_features': [2, 3, 4],
        'criterion': ['gini', 'entropy']
    },
    {
        'n_estimators': [100, 200, 400, 500, 600],
        'learning_rate': [0.1, 0.5, 1]
    },
    {
        'C': [1, 1.5, 2.0],
        'kernel': ['linear', 'poly', 'rbf']
    },
    {
        'n_neighbors': [5, 8, 10],
        'weights': ['uniform', 'distance']
    },
    {
        'n_estimators': [100, 200, 400, 500, 600],
        'criterion': ['gini', 'entropy'],
        'max_features': ['auto', 'sqrt', 'log2']
    }

]
```

In [77]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [78]:
```
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, roc_auc_score
```

In [79]:
```
from sklearn.model_selection import cross_val_predict
```

In [80]:
```
model_param_map = dict(zip(models, param_grid))
model_param_map
```

Out[80]:
```
{GradientBoostingClassifier(random_state=42): {'loss': ['log_loss',
   'exponential'],
  'n_estimators': [100, 200, 400, 500, 600],
  'max_features': [2, 3, 4],
  'learning_rate': [0.1, 0.5, 1]},
 RandomForestClassifier(random_state=42): {'n_estimators': [100,
   200,
   400,
   500,
   600],
  'max_features': [2, 3, 4],
  'criterion': ['gini', 'entropy']},
 AdaBoostClassifier(random_state=42): {'n_estimators': [100,
   200,
   400,
   500,
   600],
  'learning_rate': [0.1, 0.5, 1]},
 SVC(random_state=42): {'C': [1, 1.5, 2.0],
  'kernel': ['linear', 'poly', 'rbf']},
 KNeighborsClassifier(): {'n_neighbors': [5, 8, 10],
  'weights': ['uniform', 'distance']},
 BalancedRandomForestClassifier(random_state=42): {'n_estimators': [100,
   200,
   400,
   500,
   600],
  'criterion': ['gini', 'entropy'],
  'max_features': ['auto', 'sqrt', 'log2']}}
```

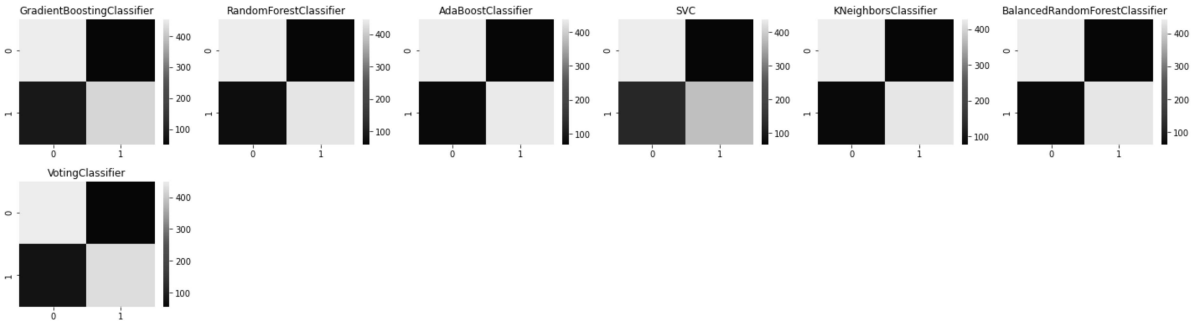
In [81]:
```
best_estimator_list = []
best_score_list = []
for best in model_param_map.items():
    rand_search = RandomizedSearchCV(best[0], best[1], scoring='accuracy', cv=
    rand_search.fit(X_train_scaled, y_smotek)
    best_estimator_list.append(rand_search.best_estimator_)
    print(rand_search.best_estimator_)
    best_score_list.append(rand_search.best_score_)
    print(rand_search.best_score_)
```

```
GradientBoostingClassifier(learning_rate=0.5, max_features=4, random_state=4
2)
0.868653367799834
RandomForestClassifier(criterion='entropy', max_features=4, n_estimators=400,
                       random_state=42)
0.8775447495488464
AdaBoostClassifier(learning_rate=1, n_estimators=500, random_state=42)
0.8656489294249623
SVC(C=1, random_state=42)
0.8162171389552748
KNeighborsClassifier(weights='distance')
0.8271472467443788
BalancedRandomForestClassifier(max_features='log2', n_estimators=400,
                               random_state=42)
0.8745744525191436
```

In [82]:
```
best_models = []
best_gb = GradientBoostingClassifier(loss='exponential', max_features=3, n_est
                             random_state=42)
best_models.append(best_gb)
best_rf = RandomForestClassifier(criterion='entropy', max_features=4, n_estima
                       random_state=42)
best_models.append(best_rf)
best_ab = AdaBoostClassifier(learning_rate=1, n_estimators=500, random_state=4
best_models.append(best_ab)
best_svc = SVC(C=2.0, kernel='poly', random_state=42)
best_models.append(best_svc)
best_kn = KNeighborsClassifier(n_neighbors=10, weights='distance')
best_models.append(best_kn)
best_brf = BalancedRandomForestClassifier(criterion='entropy', n_estimators=60
best_models.append(best_brf)

best_vc = VotingClassifier([('best_gb', best_gb),
                            ('best_rf', best_rf),
                            ('best_ab', best_ab),
                            ('best_svc', best_svc),
                            ('best_kn', best_kn)])
best_models.append(best_vc)
```

In [86]:
```
# def get_model_name(model):
#     temp = str(model)
#     end = temp.index('(')
#     title = temp[0 : end]
#     return title

def get_confusion_matrix(models, X_train, y_train):
    plt.figure(figsize=(20,16))
    kfold = StratifiedKFold(shuffle=True, random_state=42)
    for idx, model in enumerate(models):
        f = plt.subplot(6, 6, idx+1)
        y_train_pred = cross_val_predict(model, X_train, y_train, cv=kfold)
        conf_mx = confusion_matrix(y_train, y_train_pred)
        title = model.__class__.__name__
        f.set_title(title)
        g = sns.heatmap(conf_mx)
    plt.tight_layout()


def get_classification_scores(models, X_train, y_train):
    model_names = []
    accuracy_scores = []
    recall_scores = []
    precision_scores = []
    f1_scores = []
    roc_auc_scores = []
    kfold = StratifiedKFold(shuffle=True, random_state=42)
    for model in models:
        model_names.append(model.__class__.__name__)
        y_train_pred = cross_val_predict(model, X_train, y_train, cv=kfold)
        accuracy_scores.append(round(accuracy_score(y_train, y_train_pred) * 1
        recall_scores.append(round(recall_score(y_train, y_train_pred) * 100,
        precision_scores.append(round(precision_score(y_train, y_train_pred) *
        f1_scores.append(round(f1_score(y_train, y_train_pred) * 100, 2).astyp
        roc_auc_scores.append(round(roc_auc_score(y_train, y_train_pred) * 100

    metrics = list(zip(model_names, accuracy_scores, recall_scores, precision_
    df = pd.DataFrame(data=metrics, columns = ['Estimators', 'Accuracy', 'Reca
    return df
```

In [84]:
```
get_confusion_matrix(best_models, X_train_scaled, y_smotek)
```



In [87]:
```
get_classification_scores(best_models, X_train_scaled, y_smotek)
```

Out[87]:

	Estimators	Accuracy	Recall	Precision	F1 Score	ROC-AUC-Score
0	GradientBoostingClassifier	86.76%	83.6%	89.24%	86.33%	86.76%
1	RandomForestClassifier	87.25%	86.17%	88.08%	87.11%	87.25%
2	AdaBoostClassifier	85.97%	85.38%	86.4%	85.88%	85.97%
3	SVC	81.32%	75.69%	85.3%	80.21%	81.32%
4	KNeighborsClassifier	83.99%	83.2%	84.54%	83.86%	83.99%
5	BalancedRandomForestClassifier	86.96%	86.17%	87.55%	86.85%	86.96%
6	VotingClassifier	86.86%	84.58%	88.61%	86.55%	86.86%

RandomForestClassifier gives better metrics

5. Model Implementation

In [88]:
```
# final_pipeline = Pipeline([('preproc', data_pipeline),
#                            ('scaler', MinMaxScaler()),
#                            ('best_gb', GradientBoostingClassifier(loss='expone


# final_pipeline
```

In [89]:
```
final_pipeline = Pipeline([('preproc', data_pipeline),
                           ('sampling', smotek),
                           ('scaler', MinMaxScaler()),
                           ('best_vc', best_vc)])

final_pipeline
```

Out[89]:
```
Pipeline(steps=[('preproc',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer
2',
                                                                   SimpleImpu
ter()),
                                                                  ('imputer
1',
                                                                   SimpleImpu
ter(missing_values=0))]),
                                                  ['Age', 'Fare']),
                                                 ('clean',
                                                  Pipeline(steps=[('transform
er',
                                                                   CategoryTr
ansformer())]),
                                                  ['PassengerId', 'Pclass',
                                                   'Name', 'Sex', 'Age',
                                                   'SibSp', 'Parch', 'Ticke
t',
                                                   'Fare', 'Cabin',
                                                   'Embarked']),
                                                 ('cat',
                                                  Pipeline...
                                                                       max
_features=3,
                                                                       n_e
stimators=400,
                                                                       ran
dom_state=42)),
                                                 ('best_rf',
                                                  RandomForestClassifier(criteri
on='entropy',
                                                                         max_fea
tures=4,
                                                                         n_estim
ators=400,
                                                                         random_
state=42)),
                                                 ('best_ab',
                                                  AdaBoostClassifier(learning_ra
te=1,
                                                                     n_estimator
s=500,
                                                                     random_stat
e=42)),
                                                 ('best_svc',
                                                  SVC(C=2.0, kernel='poly',
```

```
                                                  random_state=42)),
                                                ('best_kn',
                                                 KNeighborsClassifier(n_neighbo
rs=10,
                                                                      weights
='distance'))]))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [90]:
```
final_pipeline.fit(train_out, y_train)
predictions = final_pipeline.predict(test)
predictions
```

Out[90]:
```
array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [91]:
```
submission = pd.DataFrame({'PassengerId':test['PassengerId'],'Survived':predic

submission.sample(10)
```

Out[91]:

	PassengerId	Survived
332	1224	0
221	1113	0
237	1129	0
210	1102	0
124	1016	0
2	894	0
19	911	1
247	1139	0
137	1029	0
82	974	0

In [92]:
```
filename = 'Titanic_Survival_Predictions.csv'

submission.to_csv(filename,index=False)

print('Saved file: ' + filename)
```

```
Saved file: Titanic_Survival_Predictions.csv
```

In [93]:
```
import joblib
```

In [94]:
```
joblib.dump(final_pipeline, 'predict_survivor_V2.pkl')
```

Out[94]:
```
['predict_survivor_V2.pkl']
```

In [ ]:
```

```