STATISTICAL CONSULTING HW1

Data Analysis of titanic

Jen Wei,Lee(RE6131024)

2025-02-27

Table of contents

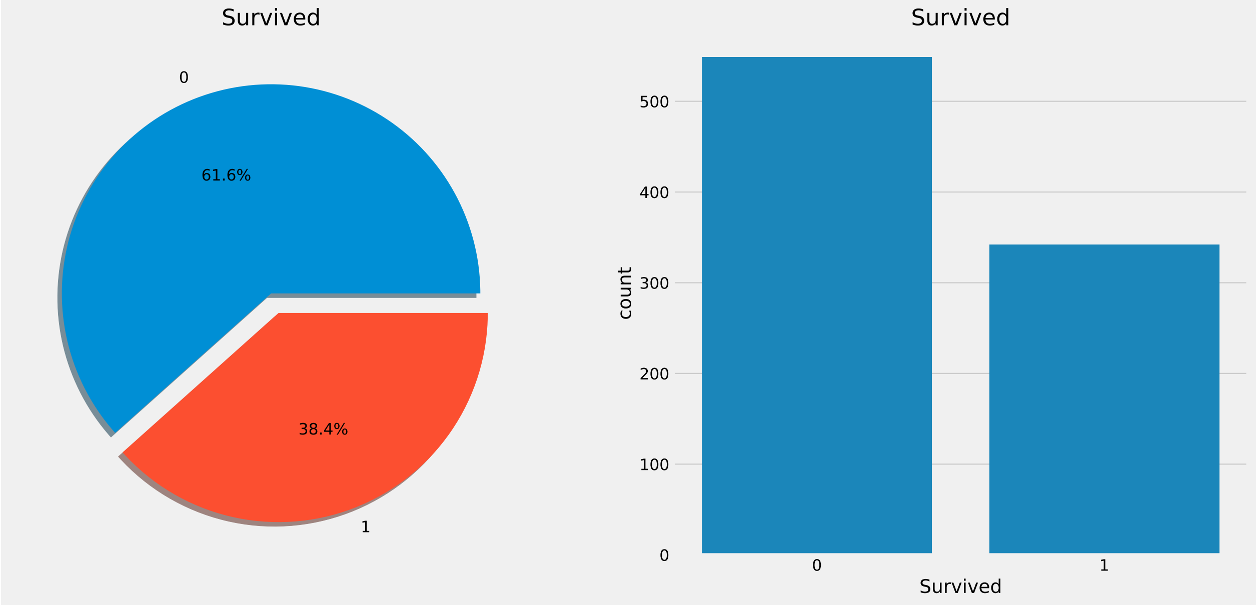
Exploratory Data Analysis(EDA)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
data=pd.read_csv('./train.csv')
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0

How many Survived??

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data['Survived'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('Survived')
ax[0].set_ylabel('')
sns.countplot(x='Survived', data=data, ax=ax[1])
ax[1].set_title('Survived')
plt.show()
```



The Age, Cabin and Embarked have null values.

```
data.isnull().sum()
```

```
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
```

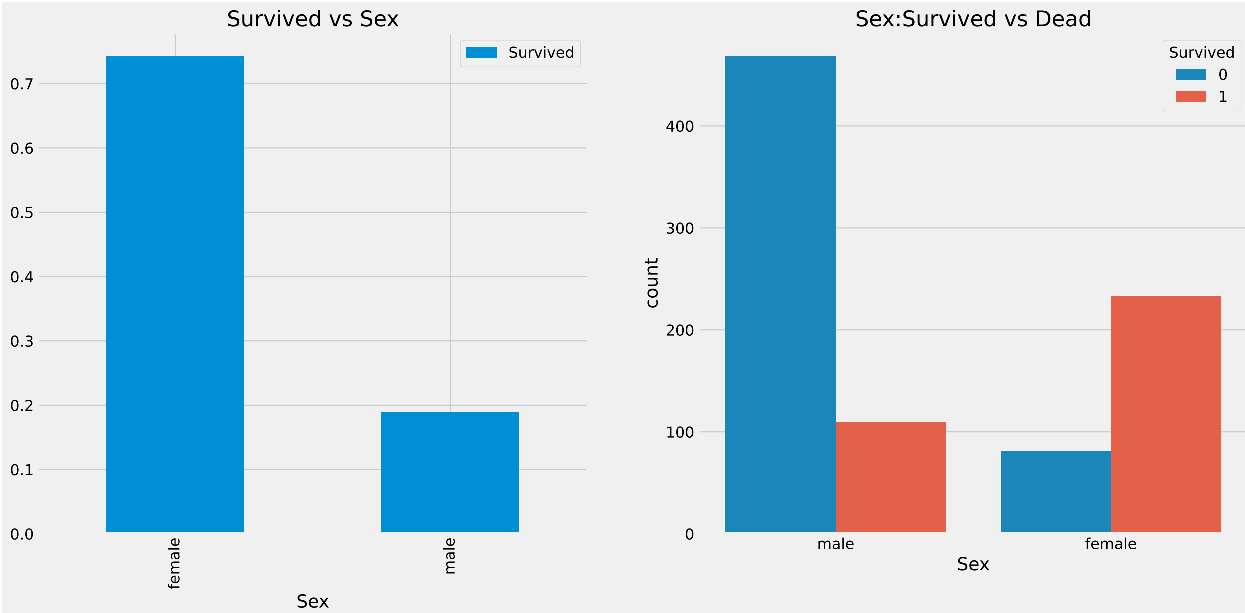
Analysing The Features

Sex–> Categorical Feature

```
data.groupby(['Sex','Survived'])['Survived'].count()
```

```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data[['Sex','Survived']].groupby(['Sex']).mean().plot.bar(ax=ax[0])
ax[0].set_title('Survived vs Sex')
sns.countplot(x = 'Sex',hue='Survived',data=data,ax=ax[1])
ax[1].set_title('Sex:Survived vs Dead')
plt.show()
```



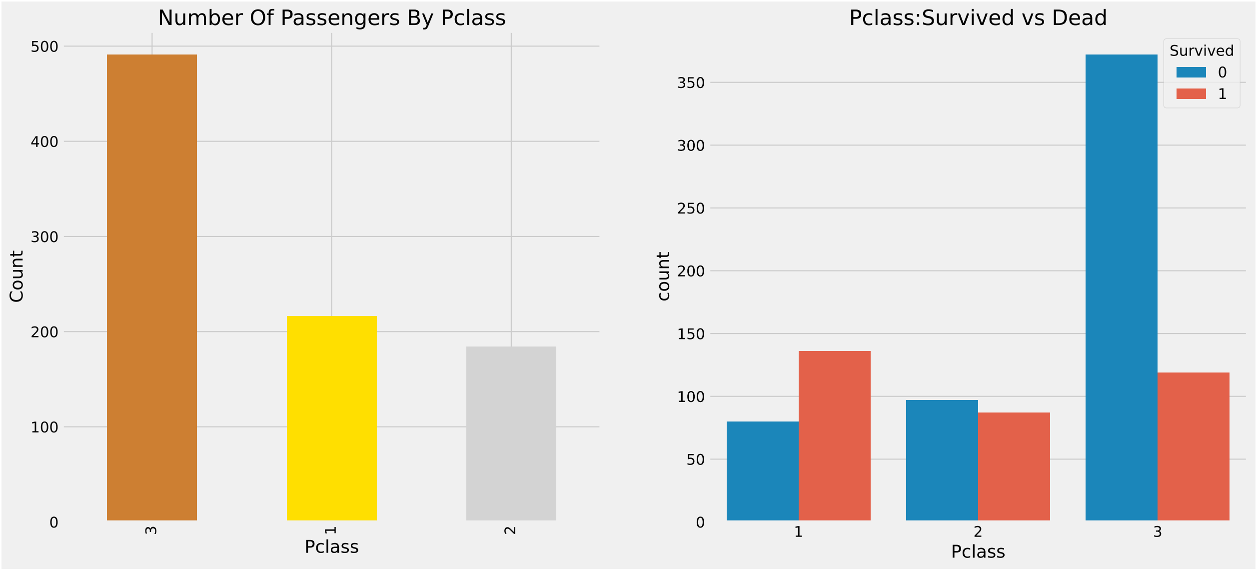
Pclass –> Ordinal Feature

```
pd.crosstab(data.Pclass,data.Survived,margins=True).style.background_gradient(cmap='summer_r')
```

Table 2

Survived Pclass	0	1	All
1	80	136	216
2	97	87	184
3	372	119	491
All	549	342	891

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data['Pclass'].value_counts().plot.bar(color=['#CD7F32','#FFDF00','#D3D3D3'],ax=ax[0])
ax[0].set_title('Number Of Passengers By Pclass')
ax[0].set_ylabel('Count')
sns.countplot(x = 'Pclass',hue='Survived',data=data,ax=ax[1])
ax[1].set_title('Pclass:Survived vs Dead')
plt.show()
```

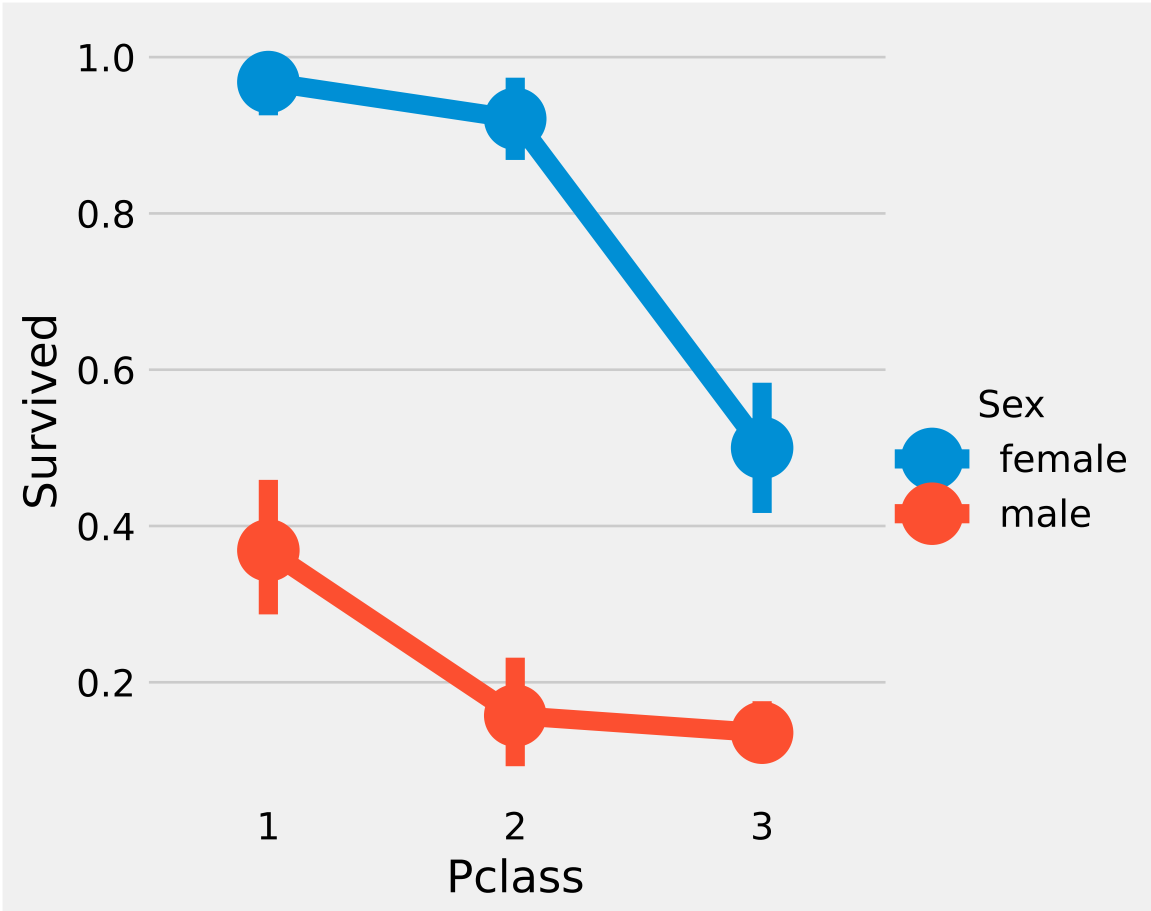



```
pd.crosstab([data.Sex,data.Survived],data.Pclass,margins=True).style.background_gradient(cmap='summer_r'
```

Table 3

Sex	Pclass / Survived	1
female	0	3
	1	91
male	0	77
	1	45
All		21

```
sns.catplot(x='Pclass', y='Survived', hue='Sex', data=data, kind='point')
plt.show()
```

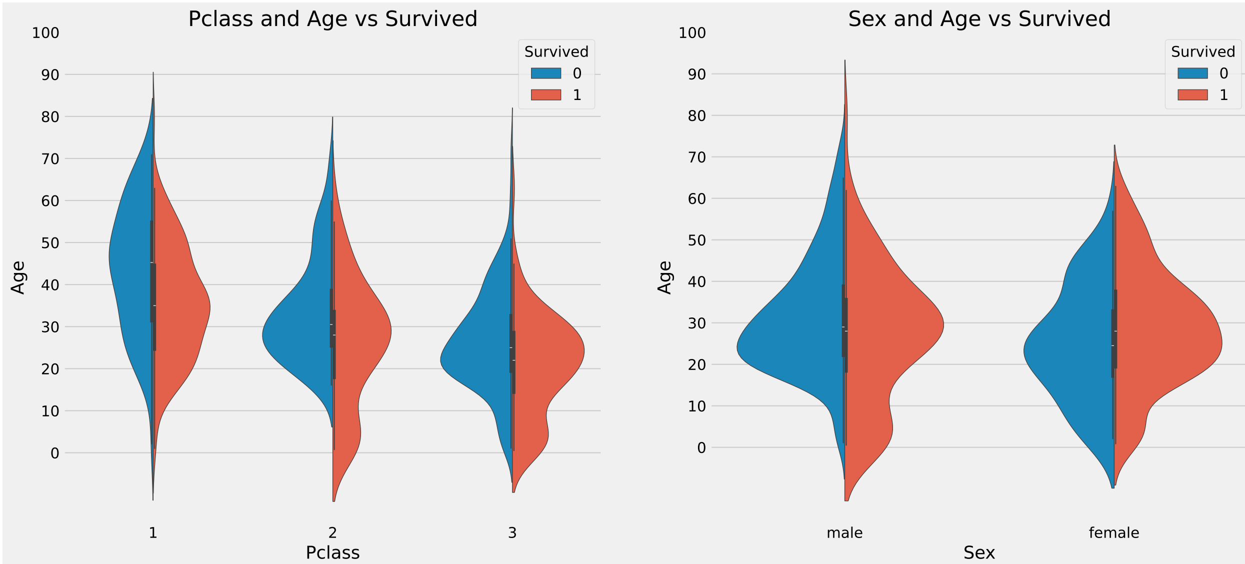


Age–> Continous Feature

```
print('Oldest Passenger was of:',data['Age'].max(),'Years')
print('Youngest Passenger was of:',data['Age'].min(),'Years')
print('Average Age on the ship:',data['Age'].mean(),'Years')
```

```
Oldest Passenger was of: 80.0 Years
Youngest Passenger was of: 0.42 Years
Average Age on the ship: 29.69911764705882 Years
```

```
f, ax = plt.subplots(1, 2, figsize=(18, 8))
sns.violinplot(x="Pclass", y="Age", hue="Survived", data=data, split=True, ax=ax[0])
ax[0].set_title('Pclass and Age vs Survived')
ax[0].set_yticks(range(0, 110, 10))
sns.violinplot(x="Sex", y="Age", hue="Survived", data=data, split=True, ax=ax[1])
ax[1].set_title('Sex and Age vs Survived')
ax[1].set_yticks(range(0, 110, 10))
plt.show()
```



name–> Filling NaN Ages

```
data['Initial']=0
for i in data:
    data['Initial']=data.Name.str.extract('([A-Za-z]+)\.')
```

```
pd.crosstab(data.Initial,data.Sex).T.style.background_gradient(cmap='summer_r')
```

Table 4

Initial Sex	Capt	Col	Countess	Don	Dr	Jonkheer	Lady	Major	Master	Miss	Mlle	Mme	Mr	Mrs	Ms
female	0	0	1	0	1	0	1	0	0	182	2	1	0	125	1
male	1	2	0	1	6	1	0	2	40	0	0	0	517	0	0

```
data['Initial'].replace(['Mlle','Mme','Ms','Dr','Major','Lady','Countess','Jonkheer','Col','Rev','Capt',
                        ['Miss','Miss','Miss','Mr','Mr','Mrs','Mrs','Other','Other','Other','Mr','Mr','M
```

```
data.groupby('Initial')['Age'].mean()
```

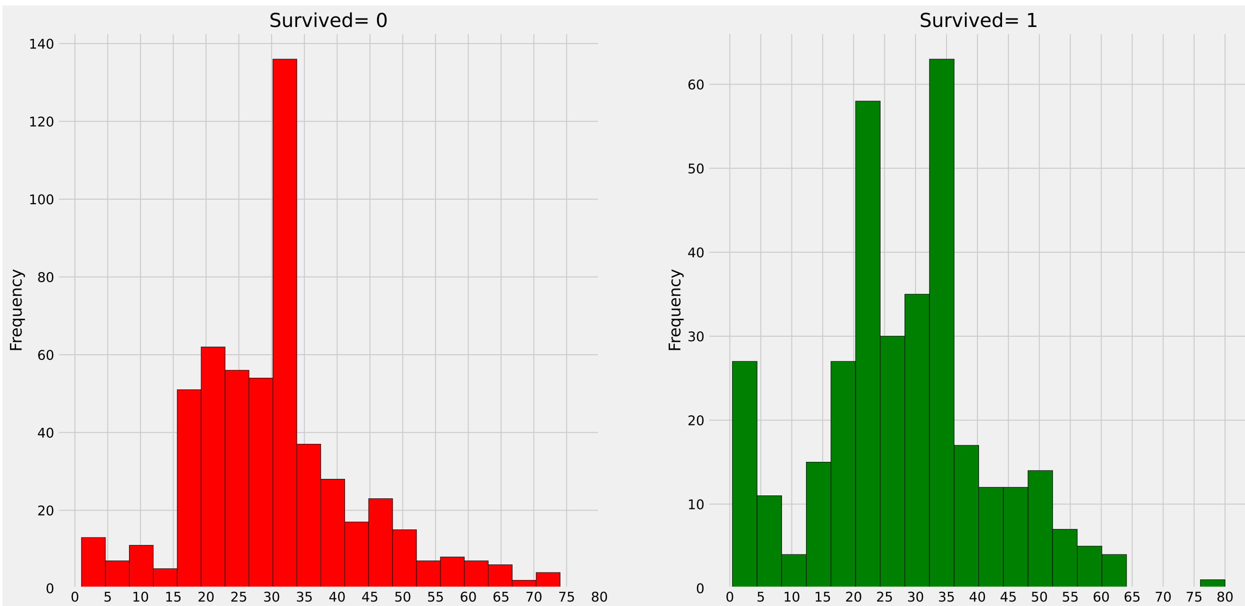
```
Initial
Master     4.574167
Miss      21.860000
Mr        32.739609
Mrs       35.981818
Other     45.888889
Name: Age, dtype: float64
```

```
data.loc[(data.Age.isnull())&(data.Initial=='Mr'),'Age']=33
data.loc[(data.Age.isnull())&(data.Initial=='Mrs'),'Age']=36
data.loc[(data.Age.isnull())&(data.Initial=='Master'),'Age']=5
data.loc[(data.Age.isnull())&(data.Initial=='Miss'),'Age']=22
data.loc[(data.Age.isnull())&(data.Initial=='Other'),'Age']=46
```

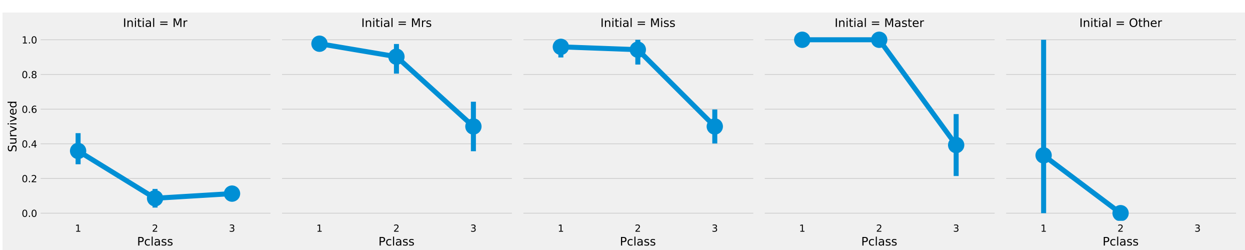
```
data.Age.isnull().any()
```

```
False
```

```
f,ax=plt.subplots(1,2,figsize=(20,10))
data[data['Survived']==0].Age.plot.hist(ax=ax[0],bins=20,edgecolor='black',color='red')
ax[0].set_title('Survived= 0')
x1=list(range(0,85,5))
ax[0].set_xticks(x1)
data[data['Survived']==1].Age.plot.hist(ax=ax[1],color='green',bins=20,edgecolor='black')
ax[1].set_title('Survived= 1')
x2=list(range(0,85,5))
ax[1].set_xticks(x2)
plt.show()
```




```
sns.catplot(x='Pclass', y='Survived', col='Initial', data=data, kind='point')
plt.show()
```



Embarked–> Categorical Value

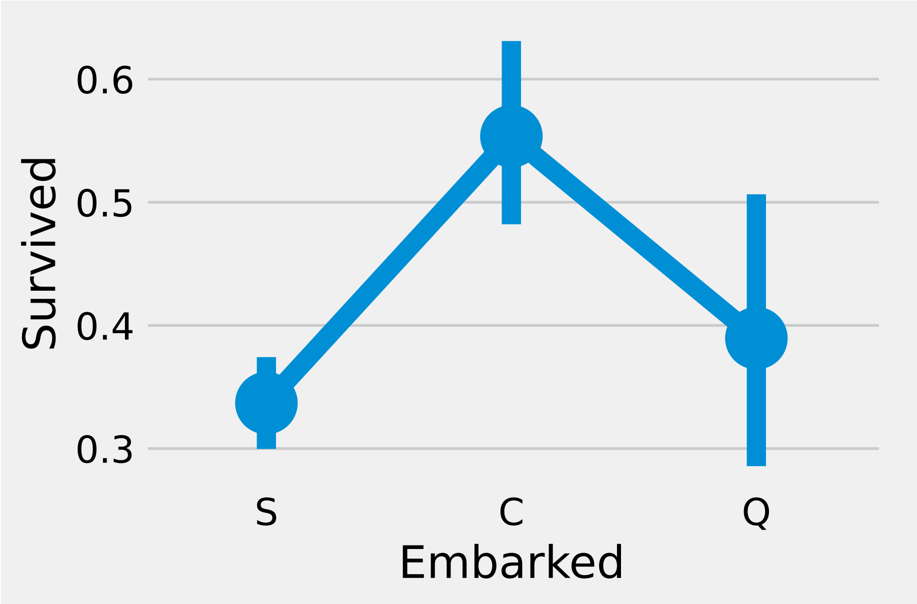
```
pd.crosstab([data.Embarked,data.Pclass],[data.Sex,data.Survived],
            margins=True).style.background_gradient(cmap='summer_r')
```

Table 5

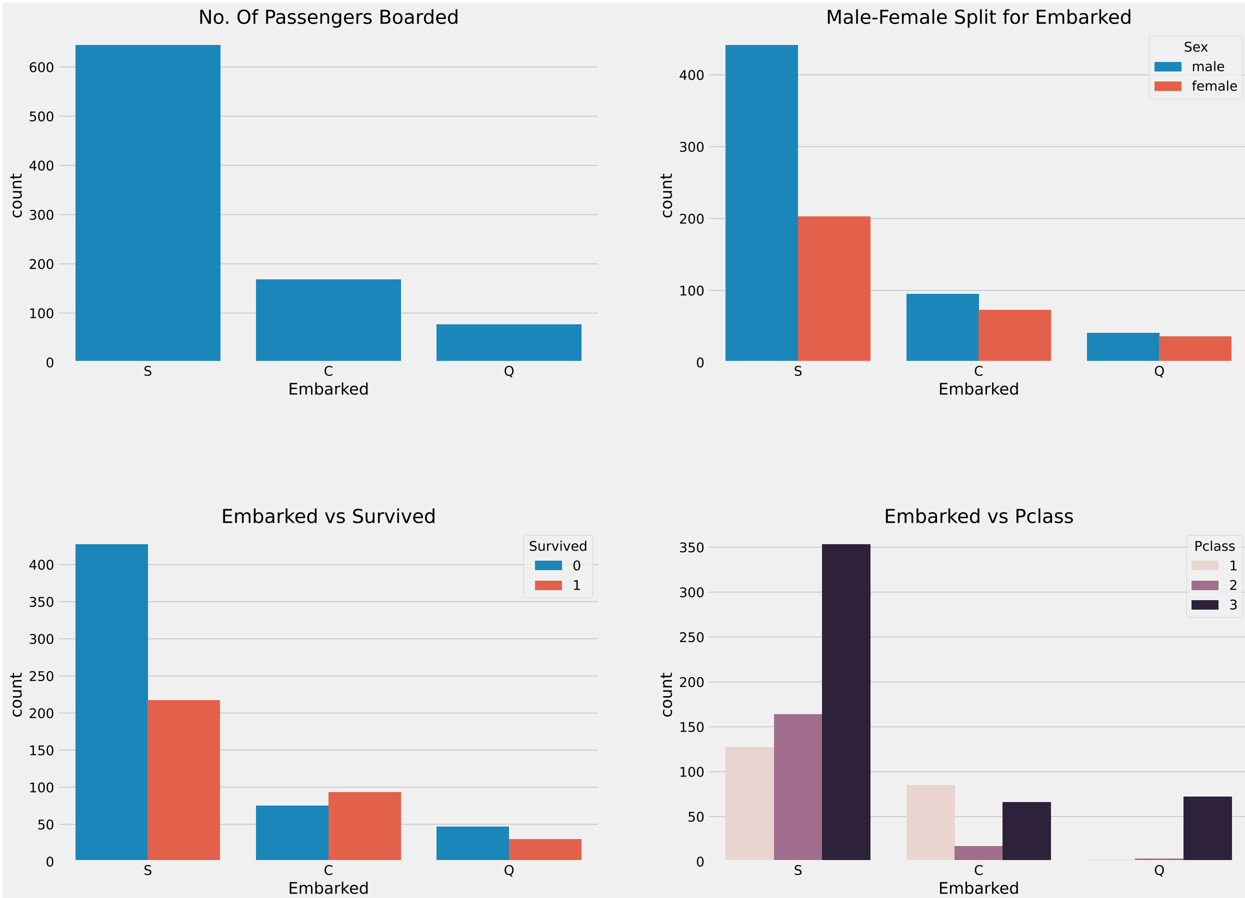
	Sex	fe
	Survived	0
Embarked	Pclass	
C	1	1
	2	0
	3	8
Q	1	0
	2	0
	3	9
S	1	2
	2	6
	3	55
All		81

Chances for Survival by Port Of Embarkation

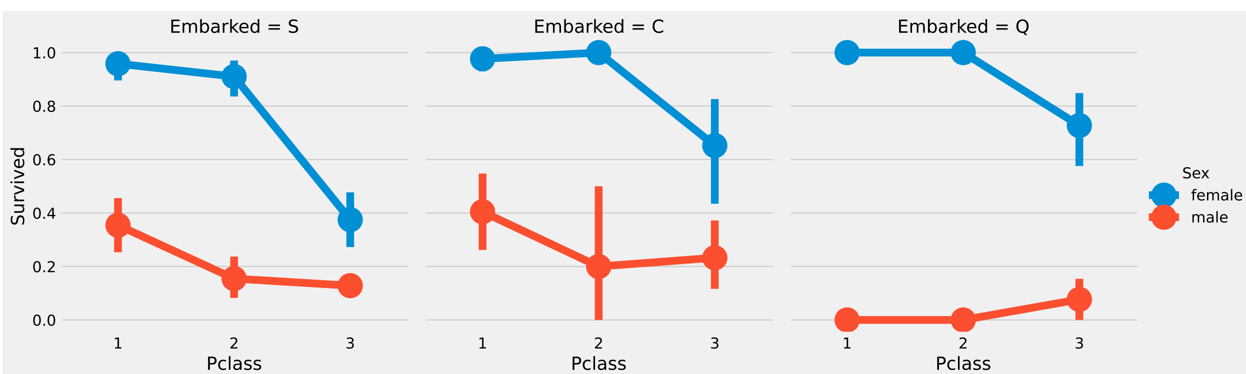
```
sns.catplot(x='Embarked', y='Survived', data=data, kind='point')
fig=plt.gcf()
fig.set_size_inches(5,3)
plt.show()
```




```
f,ax=plt.subplots(2,2,figsize=(20,15))
sns.countplot(x ='Embarked',data=data,ax=ax[0,0])
ax[0,0].set_title('No. Of Passengers Boarded')
sns.countplot(x ='Embarked',hue='Sex',data=data,ax=ax[0,1])
ax[0,1].set_title('Male-Female Split for Embarked')
sns.countplot(x ='Embarked',hue='Survived',data=data,ax=ax[1,0])
ax[1,0].set_title('Embarked vs Survived')
sns.countplot(x ='Embarked',hue='Pclass',data=data,ax=ax[1,1])
ax[1,1].set_title('Embarked vs Pclass')
plt.subplots_adjust(wspace=0.2,hspace=0.5)
plt.show()
```




```
sns.catplot(x='Pclass', y='Survived', hue='Sex', col='Embarked', data=data, kind='point')
plt.show()
```



Filling Embarked NaN

As we saw that maximum passengers boarded from Port S, we replace NaN with S.

```
data['Embarked'].fillna('S',inplace=True)
data.Embarked.isnull().any()
```

```
False
```

SibSip–>Discrete Feature

```
pd.crosstab([data.SibSp],
            data.Survived).style.background_gradient(cmap='summer_r')
```

Table 6

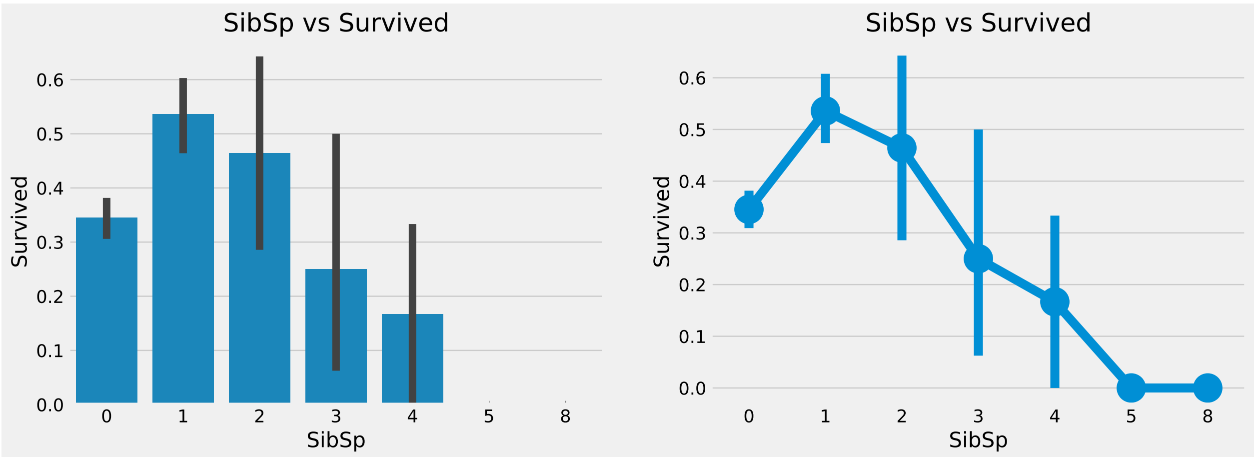
Survived SibSp	0	1
0	398	210
1	97	112
2	15	13
3	12	4
4	15	3
5	5	0
8	7	0

```
f, ax = plt.subplots(1, 2, figsize=(15, 5))

# First subplot - barplot
sns.barplot(x='SibSp', y='Survived', data=data, ax=ax[0])
ax[0].set_title('SibSp vs Survived')

# Second subplot - using catplot instead of factorplot
sns.pointplot(x='SibSp', y='Survived', data=data, ax=ax[1])
ax[1].set_title('SibSp vs Survived')

plt.show()
```




```
pd.crosstab(data.SibSp,
            data.Pclass).style.background_gradient(cmap='summer_r')
```

Table 7

Pclass SibSp	1	2	3
0	137	120	351

Pclass SibSp	1	2	3
1	71	55	83
2	5	8	15
3	3	1	12
4	0	0	18
5	0	0	5
8	0	0	7

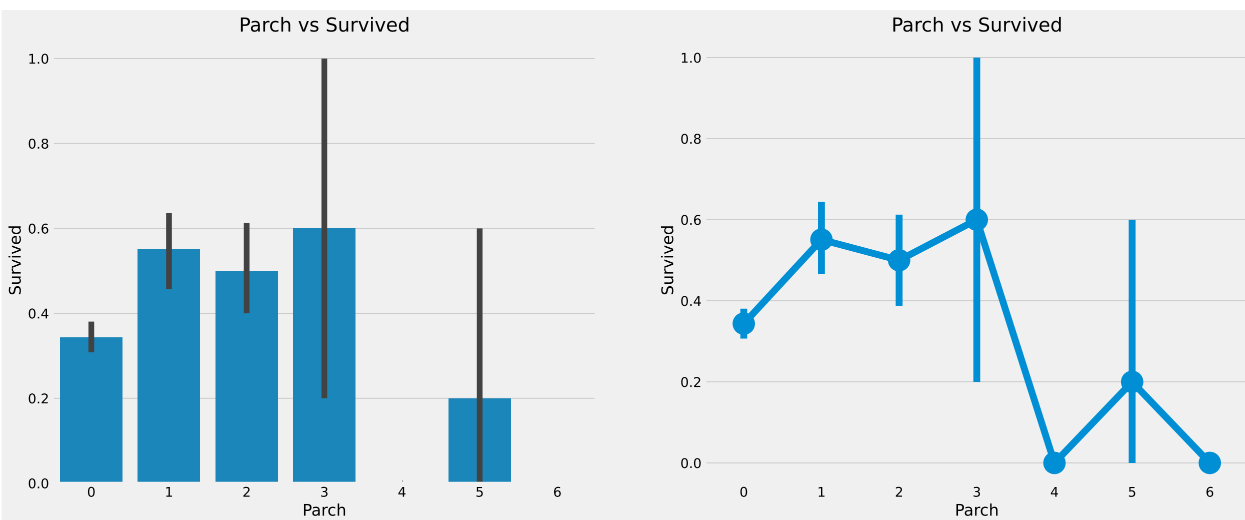
Parch

```
pd.crosstab(data.Parch,
            data.Pclass).style.background_gradient(cmap='summer_r')
```

Table 8

Pclass Parch	1	2	3
0	163	134	381
1	31	32	55
2	21	16	43
3	0	2	3
4	1	0	3
5	0	0	5
6	0	0	1

```
f,ax=plt.subplots(1,2,figsize=(20,8))
sns.barplot(x='Parch',y='Survived',data=data,ax=ax[0])
ax[0].set_title('Parch vs Survived')
sns.pointplot(x='Parch',y='Survived',data=data,ax=ax[1])
ax[1].set_title('Parch vs Survived')
plt.close(2)
plt.show()
```



Fare–> Continous Feature

```
print('Highest Fare was:',data['Fare'].max())
print('Lowest Fare was:',data['Fare'].min())
print('Average Fare was:',data['Fare'].mean())
```

```
Highest Fare was: 512.3292
Lowest Fare was: 0.0
Average Fare was: 32.204207968574636
```

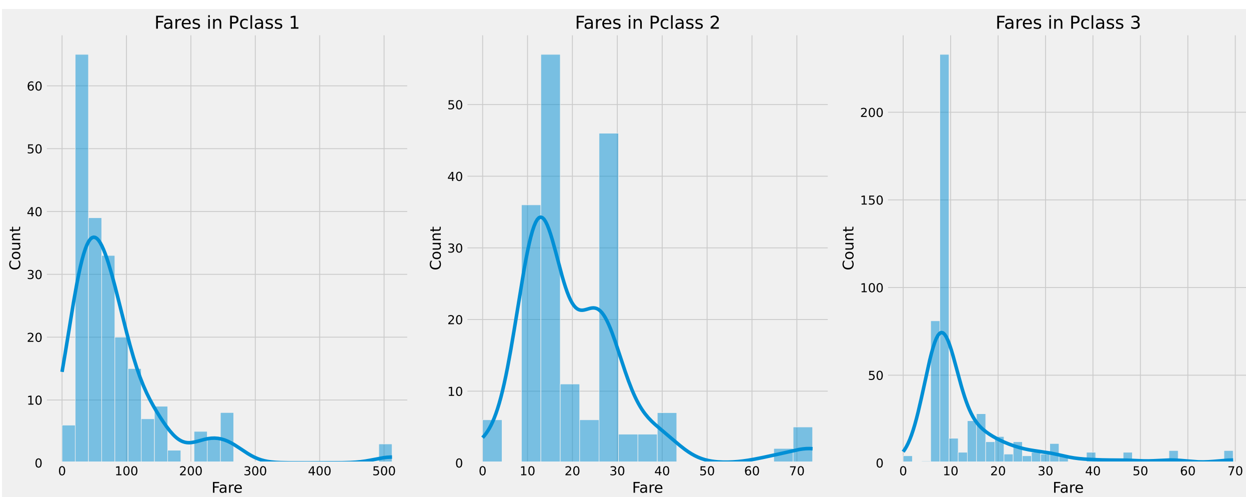
```
f, ax = plt.subplots(1, 3, figsize=(20, 8))

# Plot for Pclass 1
sns.histplot(data=data[data['Pclass']==1], x='Fare', kde=True, ax=ax[0])
ax[0].set_title('Fares in Pclass 1')

# Plot for Pclass 2
sns.histplot(data=data[data['Pclass']==2], x='Fare', kde=True, ax=ax[1])
ax[1].set_title('Fares in Pclass 2')

# Plot for Pclass 3
sns.histplot(data=data[data['Pclass']==3], x='Fare', kde=True, ax=ax[2])
ax[2].set_title('Fares in Pclass 3')

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



Observations in a Nutshell for all features:

Sex: The chance of survival for women is high as compared to men.

Pclass:There is a visible trend that being a 1st class passenger gives you better chances of survival. The survival rate for **Pclass3 is very low**. For women, the chance of survival from Pclass1 is almost 1 and is high too for those from Pclass2. **Money Wins**!!!.

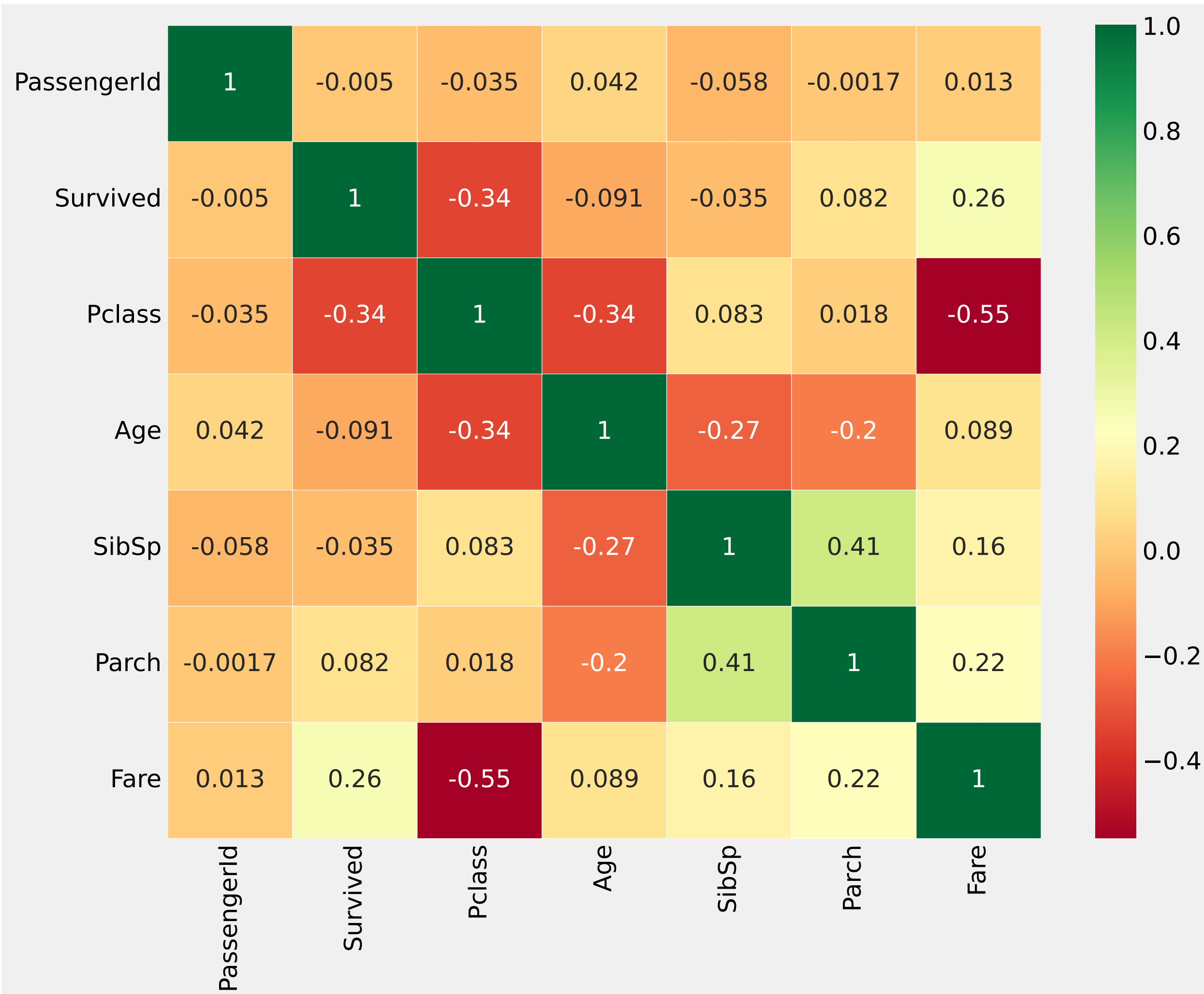
Age: Children less than 5-10 years do have a high chance of survival. Passengers between age group 15 to 35 died a lot.

Embarked: This is a very interesting feature. The chances of survival at C looks to be better than even though the majority of Pclass1 passengers got up at S. Passengers at Q were all from Pclass3.

Parch+SibSp: Having 1-2 siblings,spouse on board or 1-3 Parents shows a greater chance of probablity rather than being alone or having a large family travelling with you.

Correlation Between The Features

```
numeric_columns = data.select_dtypes(include=['float64', 'int64']).columns
correlation_matrix = data[numeric_columns].corr()
sns.heatmap(correlation_matrix,annot=True,cmap='RdYlGn',linewidths=0.2)
fig=plt.gcf()
fig.set_size_inches(10,8)
plt.show()
```



Feature Engineering and Data Cleaning

Age_band

```
data['Age_band']=0
data.loc[data['Age']<=16,'Age_band']=0
data.loc[(data['Age']>16)&(data['Age']<=32),'Age_band']=1
data.loc[(data['Age']>32)&(data['Age']<=48),'Age_band']=2
```

```
data.loc[(data['Age']>48)&(data['Age']<=64),'Age_band']=3
data.loc[data['Age']>64,'Age_band']=4
data.head(2)
```

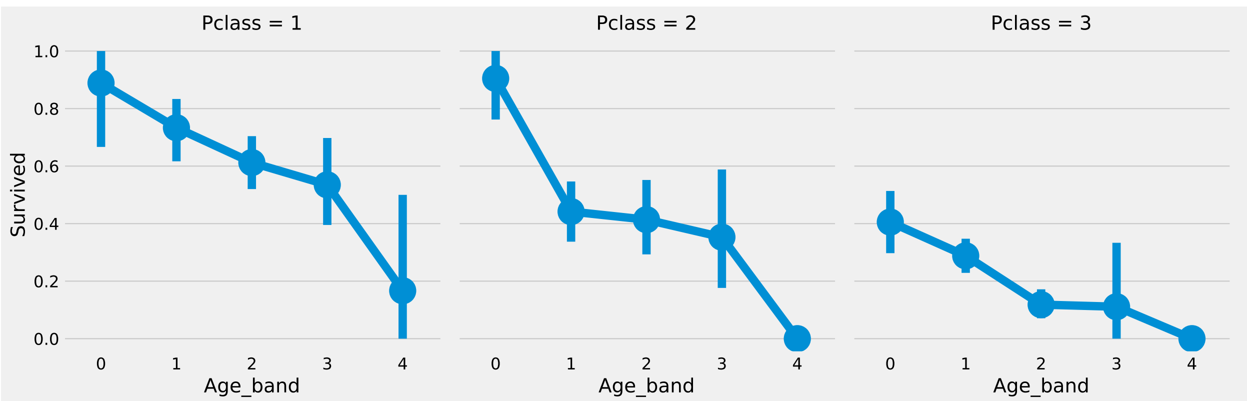
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0

```
data['Age_band'].value_counts().to_frame().style.background_gradient(cmap='summer')
```

Table 10

Age_band	count
1	382
2	325
0	104
3	69
4	11

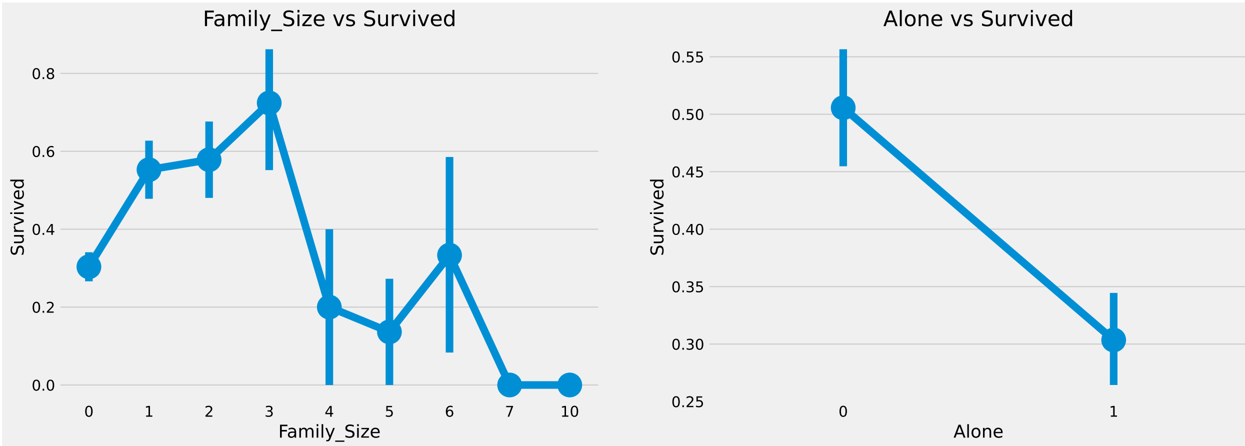
```
sns.catplot(x='Age_band',y='Survived',data=data,col='Pclass',kind='point')
plt.show()
```



Family_Size and Alone

```
data['Family_Size']=0
data['Family_Size']=data['Parch']+data['SibSp']#family size
data['Alone']=0
data.loc[data.Family_Size==0,'Alone']=1#Alone

f,ax=plt.subplots(1,2,figsize=(18,6))
sns.pointplot(x='Family_Size',y='Survived',data=data,ax=ax[0])
ax[0].set_title('Family_Size vs Survived')
sns.pointplot(x='Alone',y='Survived',data=data,ax=ax[1])
ax[1].set_title('Alone vs Survived')
plt.close(2)
plt.close(3)
plt.show()
```




```
sns.catplot(x='Alone', y='Survived', hue='Sex', col='Pclass', data=data, kind='point')
plt.show()
```

Fare_Range

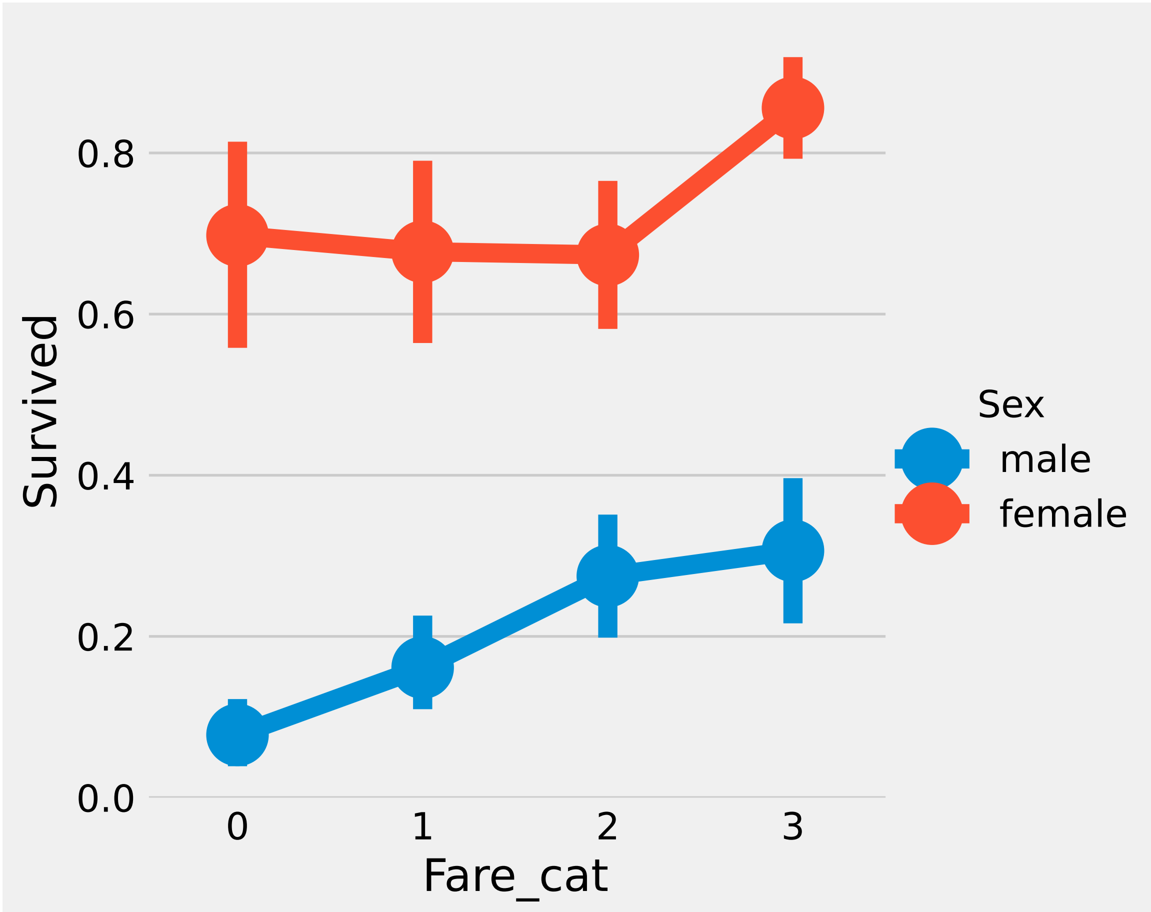
```
data['Fare_Range']=pd.qcut(data['Fare'],4)
data.groupby(['Fare_Range'])['Survived'].mean().to_frame().style.background_gradient(cmap='summer_r')
```

Table 11

	Survived
Fare_Range	
(-0.001, 7.91]	0.197309
(7.91, 14.454]	0.303571
(14.454, 31.0]	0.454955
(31.0, 512.329]	0.581081

```
data['Fare_cat']=0
data.loc[data['Fare']<=7.91,'Fare_cat']=0
data.loc[(data['Fare']>7.91)&(data['Fare']<=14.454),'Fare_cat']=1
data.loc[(data['Fare']>14.454)&(data['Fare']<=31),'Fare_cat']=2
data.loc[(data['Fare']>31)&(data['Fare']<=513),'Fare_cat']=3
```

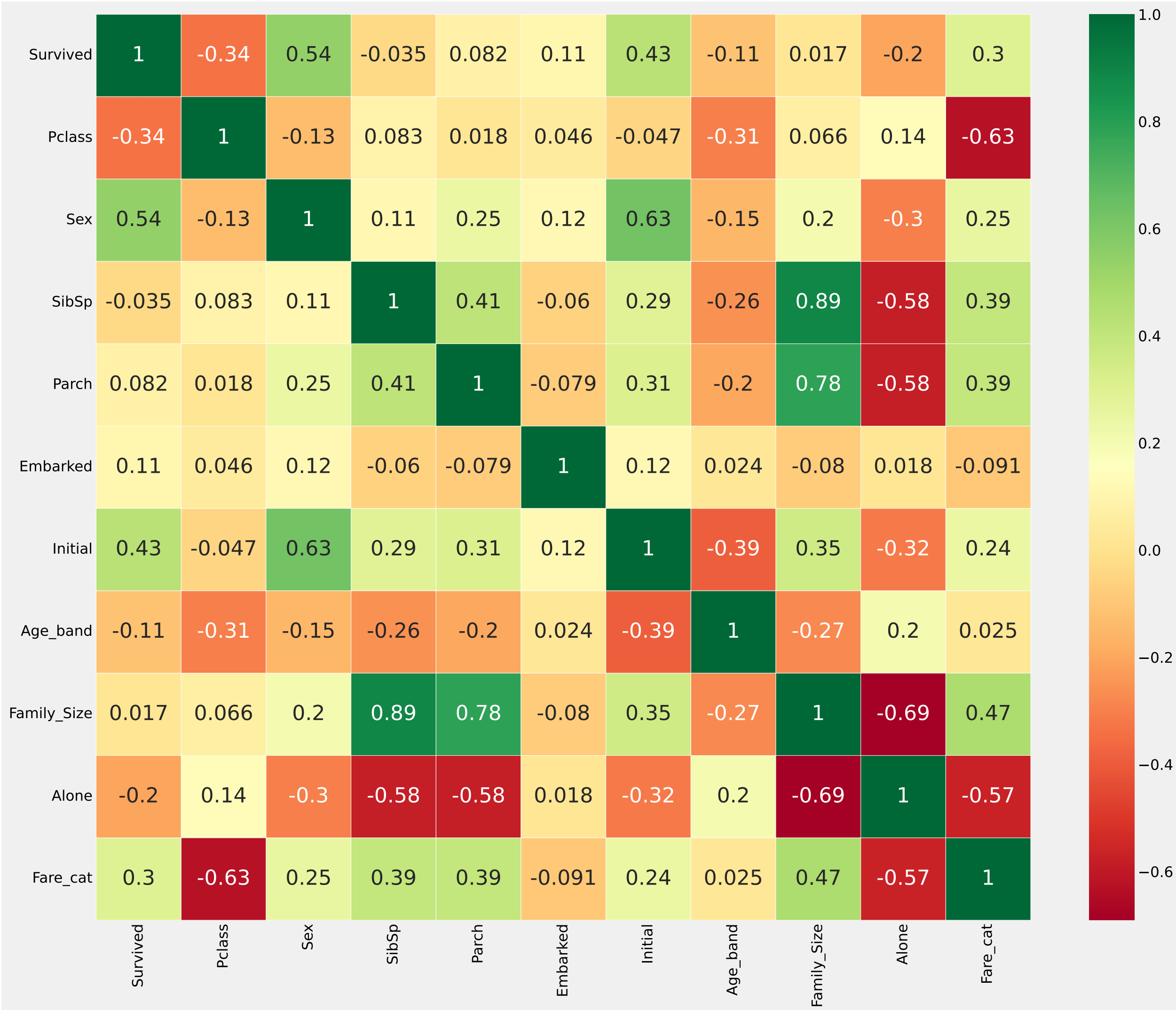
```
sns.catplot(x='Fare_cat', y='Survived', data=data, hue='Sex', kind='point')
plt.show()
```



Converting String Values into Numeric

```
data['Sex'].replace(['male','female'],[0,1],inplace=True)
data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace=True)
```

```
data.drop(['Name','Age','Ticket','Fare','Cabin','Fare_Range','PassengerId'],axis=1,inplace=True)
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'size':20})
fig=plt.gcf()
fig.set_size_inches(18,15)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



Predictive Modeling

```
#importing all the required ML packages
from sklearn.linear_model import LogisticRegression #logistic regression
from sklearn import svm #support vector Machine
from sklearn.ensemble import RandomForestClassifier #Random Forest
from sklearn.neighbors import KNeighborsClassifier #KNN
from sklearn.naive_bayes import GaussianNB #Naive bayes
from sklearn.tree import DecisionTreeClassifier #Decision Tree
from sklearn.model_selection import train_test_split #training and testing data split
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
```

Radial Support Vector Machines(rbf-SVM)

```
train,test=train_test_split(data,test_size=0.3,random_state=0,stratify=data['Survived'])
train_X=train[train.columns[1:]]
```

```
train_Y=train[train.columns[:1]]
test_X=test[test.columns[1:]]
test_Y=test[test.columns[:1]]
X=data[data.columns[1:]]
Y=data['Survived']
```

```
model=svm.SVC(kernel='rbf',C=1,gamma=0.1)
model.fit(train_X,train_Y)
prediction1=model.predict(test_X)
print('Accuracy for rbf SVM is ',metrics.accuracy_score(prediction1,test_Y))
```

```
Accuracy for rbf SVM is  0.835820895522388
```

Linear Support Vector Machine(linear-SVM)

```
model=svm.SVC(kernel='linear',C=0.1,gamma=0.1)
model.fit(train_X,train_Y)
prediction2=model.predict(test_X)
print('Accuracy for linear SVM is',metrics.accuracy_score(prediction2,test_Y))
```

```
Accuracy for linear SVM is 0.8171641791044776
```

Logistic Regression

```
model = LogisticRegression()
model.fit(train_X,train_Y)
prediction3=model.predict(test_X)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction3,test_Y))
```

```
The accuracy of the Logistic Regression is 0.8134328358208955
```

Decision Tree

```
model=DecisionTreeClassifier()
model.fit(train_X,train_Y)
prediction4=model.predict(test_X)
print('The accuracy of the Decision Tree is',metrics.accuracy_score(prediction4,test_Y))
```

```
The accuracy of the Decision Tree is 0.8059701492537313
```

K-Nearest Neighbours(KNN)

```
model=KNeighborsClassifier()
model.fit(train_X,train_Y)
prediction5=model.predict(test_X)
print('The accuracy of the KNN is',metrics.accuracy_score(prediction5,test_Y))
```

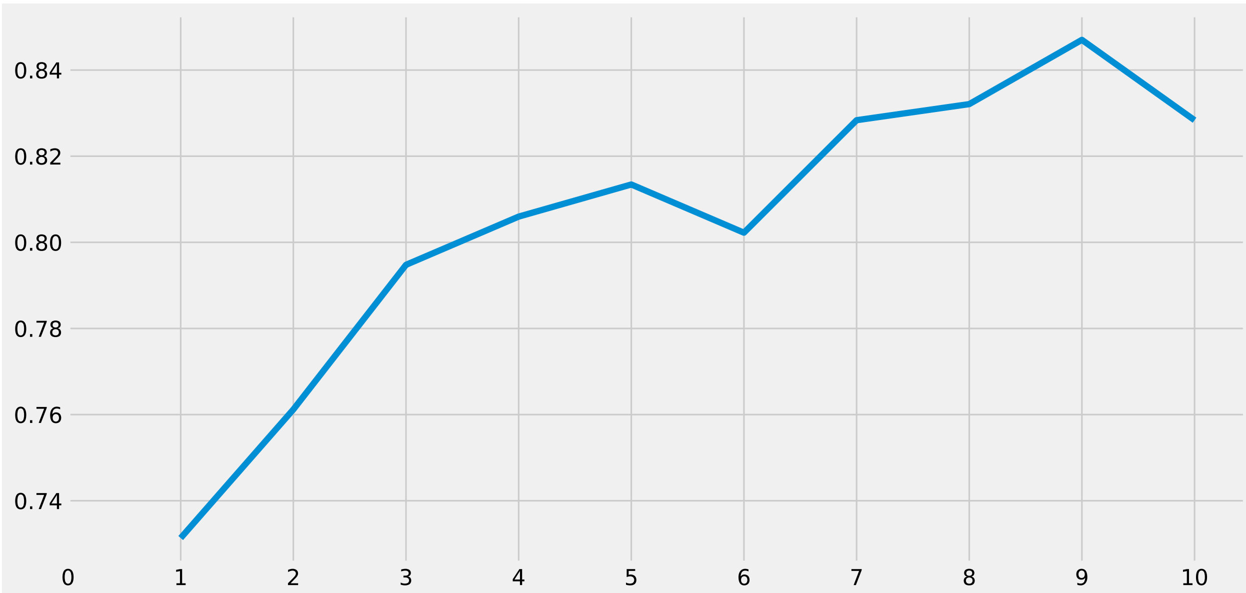
```
The accuracy of the KNN is 0.8134328358208955
```

```
a_index = list(range(1,11))
a = pd.Series(dtype=float)  # Initialize with explicit dtype
x = [0,1,2,3,4,5,6,7,8,9,10]

for i in list(range(1,11)):
```

```
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X, train_Y)
    prediction = model.predict(test_X)
    # Use concat instead of append
    a = pd.concat([a, pd.Series([metrics.accuracy_score(prediction, test_Y)])])

plt.plot(a_index, a)
plt.xticks(x)
fig = plt.gcf()
fig.set_size_inches(12,6)
plt.show()
print('Accuracies for different values of n are:', a.values, 'with the max value as ', a.values.max())
```




```
Accuracies for different values of n are: [0.73134328 0.76119403 0.79477612 0.80597015 0.81343284 0.8022
 0.82835821 0.83208955 0.84701493 0.82835821] with the max value as  0.8470149253731343
```

Gaussian Naive Bayes

```
model=GaussianNB()
model.fit(train_X,train_Y)
prediction6=model.predict(test_X)
print('The accuracy of the NaiveBayes is',metrics.accuracy_score(prediction6,test_Y))
```

```
The accuracy of the NaiveBayes is 0.8134328358208955
```

Random Forests

```
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_Y)
prediction7=model.predict(test_X)
print('The accuracy of the Random Forests is',metrics.accuracy_score(prediction7,test_Y))
```

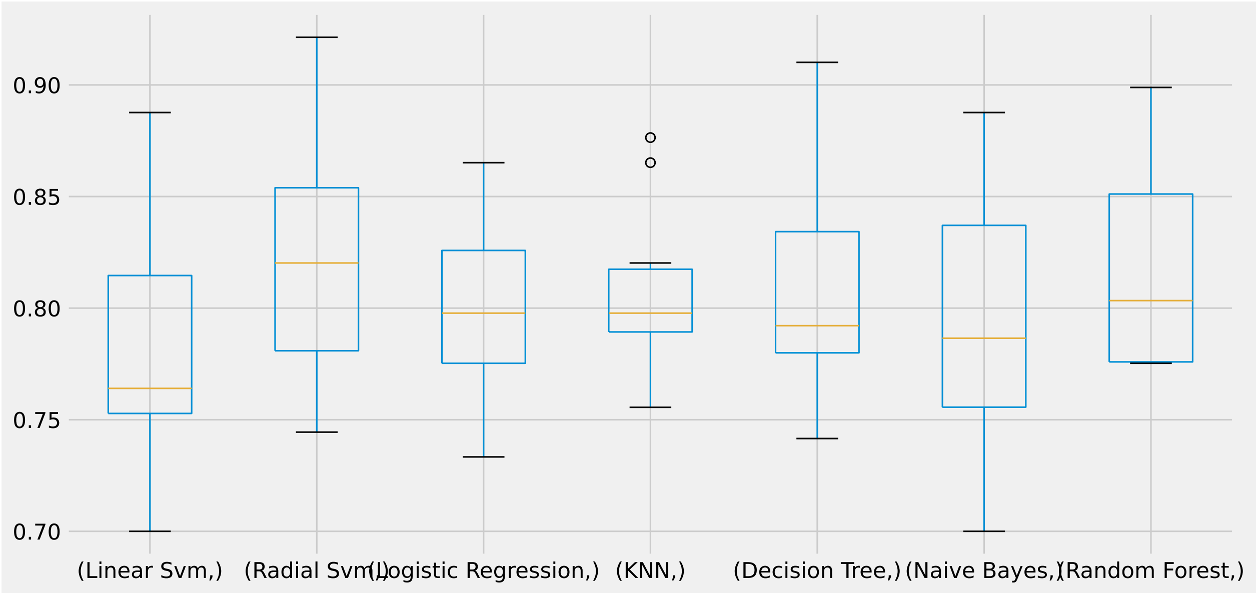
```
The accuracy of the Random Forests is 0.8208955223880597
```

Cross Validation

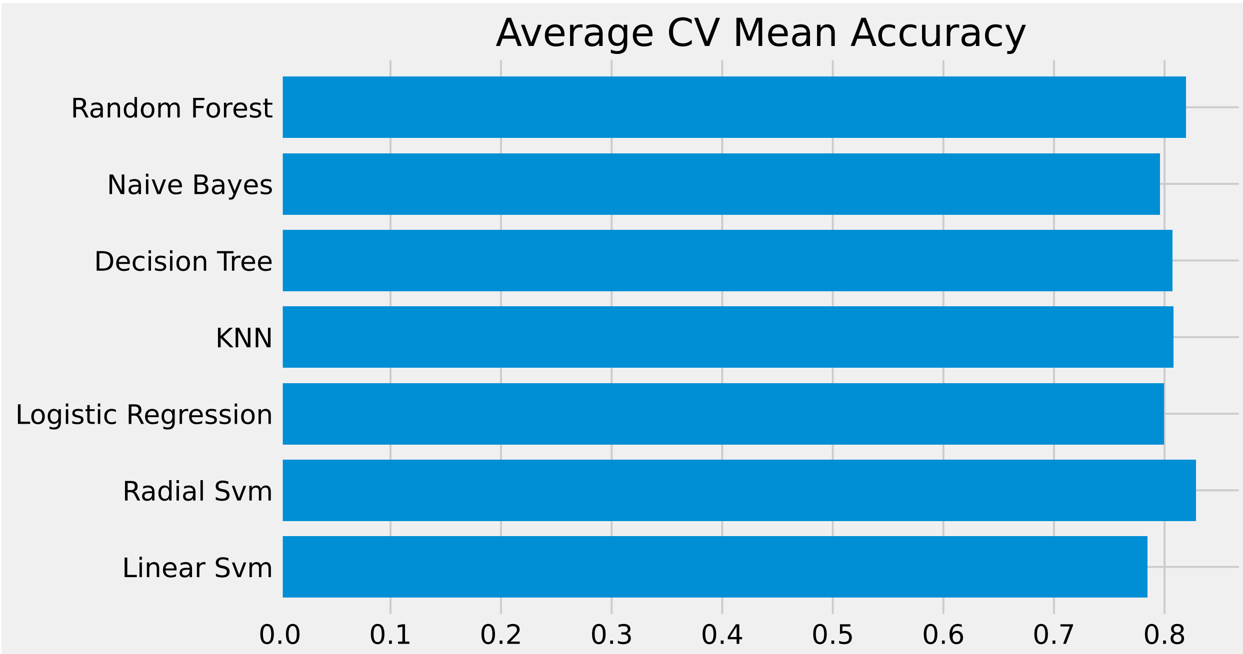
```
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
kfold = KFold(n_splits=10,shuffle=True, random_state=22) # k=10, split the data into 10 equal parts
xyz=[]
accuracy=[]
std=[]
classifiers=['Linear Svm','Radial Svm','Logistic Regression','KNN','Decision Tree','Naive Bayes','Randon
models=[svm.SVC(kernel='linear'),svm.SVC(kernel='rbf'),LogisticRegression(),KNeighborsClassifier(n_neigh
for i in models:
    model = i
    cv_result = cross_val_score(model,X,Y, cv = kfold,scoring = "accuracy")
    cv_result=cv_result
    xyz.append(cv_result.mean())
    std.append(cv_result.std())
    accuracy.append(cv_result)
new_models_dataframe2=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)
new_models_dataframe2
```

	CV Mean	Std
Linear Svm	0.784607	0.057841
Radial Svm	0.828377	0.057096
Logistic Regression	0.799176	0.040154
KNN	0.808140	0.035630
Decision Tree	0.806991	0.045000
Naive Bayes	0.795843	0.054861
Random Forest	0.819351	0.045670

```
plt.subplots(figsize=(12,6))
box=pd.DataFrame(accuracy,index=[classifiers])
box.T.boxplot()
```

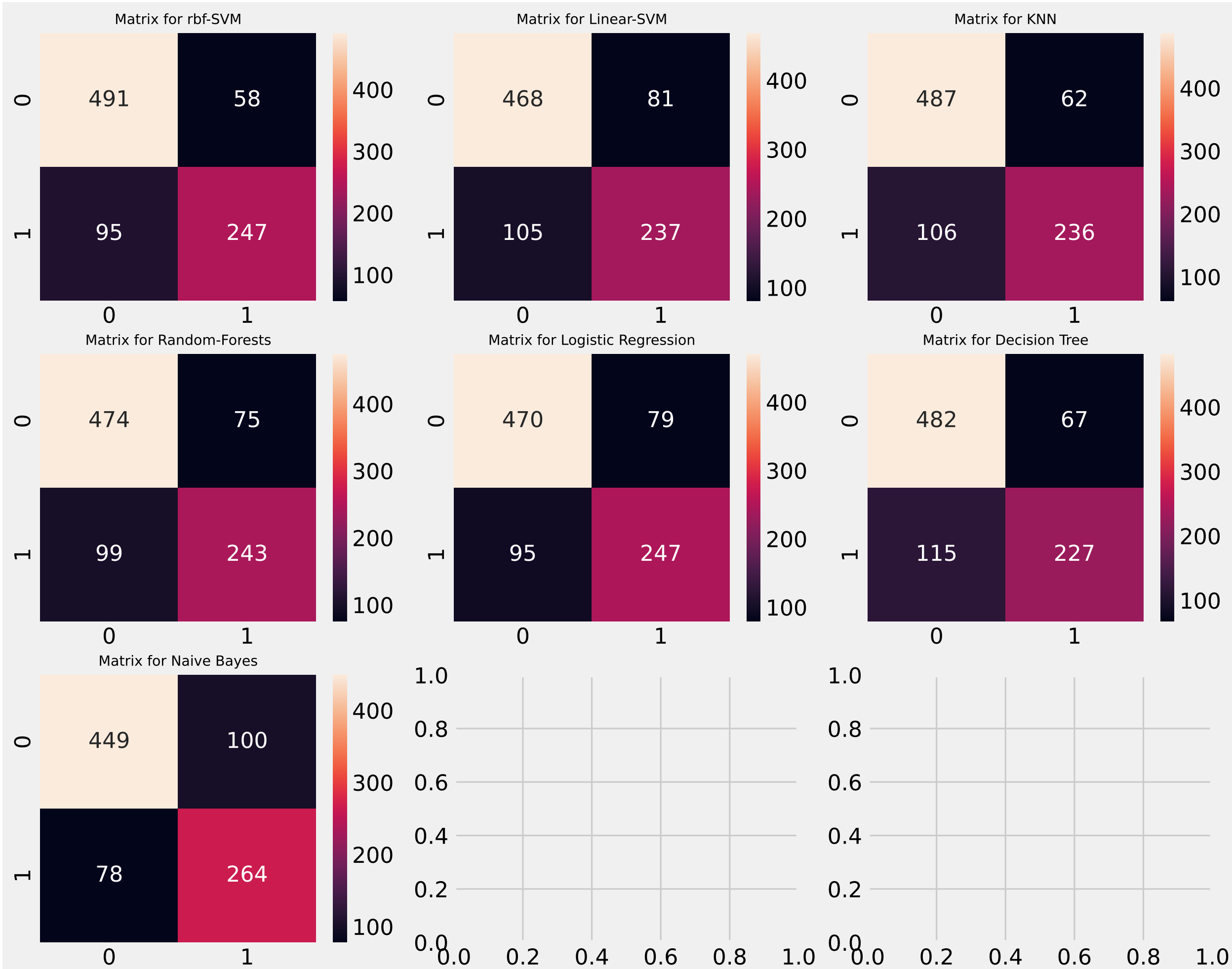


```
new_models_dataframe2['CV Mean'].plot.barh(width=0.8)
plt.title('Average CV Mean Accuracy')
fig=plt.gcf()
fig.set_size_inches(8,5)
plt.show()
```



Confusion Matrix

```
f,ax=plt.subplots(3,3,figsize=(12,10))
y_pred = cross_val_predict(svm.SVC(kernel='rbf'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,0],annot=True,fmt='2.0f')
ax[0,0].set_title('Matrix for rbf-SVM',fontsize=9)
y_pred = cross_val_predict(svm.SVC(kernel='linear'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,1],annot=True,fmt='2.0f')
ax[0,1].set_title('Matrix for Linear-SVM',fontsize=9)
y_pred = cross_val_predict(KNeighborsClassifier(n_neighbors=9),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,2],annot=True,fmt='2.0f')
ax[0,2].set_title('Matrix for KNN',fontsize=9)
y_pred = cross_val_predict(RandomForestClassifier(n_estimators=100),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,0],annot=True,fmt='2.0f')
ax[1,0].set_title('Matrix for Random-Forests',fontsize=9)
y_pred = cross_val_predict(LogisticRegression(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,1],annot=True,fmt='2.0f')
ax[1,1].set_title('Matrix for Logistic Regression',fontsize=9)
y_pred = cross_val_predict(DecisionTreeClassifier(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,2],annot=True,fmt='2.0f')
ax[1,2].set_title('Matrix for Decision Tree',fontsize=9)
y_pred = cross_val_predict(GaussianNB(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[2,0],annot=True,fmt='2.0f')
ax[2,0].set_title('Matrix for Naive Bayes',fontsize=9)
plt.subplots_adjust(hspace=0.2,wspace=0.2)
plt.show()
```



Hyper-Parameters Tuning

SVM

```
from sklearn.model_selection import GridSearchCV
C=[0.05,0.1,0.2,0.3,0.25,0.4,0.5,0.6,0.7,0.8,0.9,1]
gamma=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
kernel=['rbf','linear']
hyper={'kernel':kernel,'C':C,'gamma':gamma}
gd=GridSearchCV(estimator=svm.SVC(),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
0.8282593685267716
SVC(C=0.4, gamma=0.3)
```

Random Forests

```
n_estimators=range(100,1000,100)
hyper={'n_estimators':n_estimators}
```

```
gd=GridSearchCV(estimator=RandomForestClassifier(random_state=0),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
0.819327098110602
RandomForestClassifier(n_estimators=300, random_state=0)
```

Voting Classifier

```
from sklearn.ensemble import VotingClassifier
ensemble_lin_rbf=VotingClassifier(estimators=[('KNN',KNeighborsClassifier(n_neighbors=10)),
                                              ('RBF',svm.SVC(probability=True,kernel='rbf',C=0.5,gamma=(
                                              ('RFor',RandomForestClassifier(n_estimators=500,random_sta
                                              ('LR',LogisticRegression(C=0.05)),
                                              ('DT',DecisionTreeClassifier(random_state=0)),
                                              ('NB',GaussianNB()),
                                              ('svm',svm.SVC(kernel='linear',probability=True))
                                             ],
                       voting='soft').fit(train_X,train_Y)
print('The accuracy for ensembled model is:',ensemble_lin_rbf.score(test_X,test_Y))
cross=cross_val_score(ensemble_lin_rbf,X,Y, cv = 10,scoring = "accuracy")
print('The cross validated score is',cross.mean())
```

```
The accuracy for ensembled model is: 0.8208955223880597
The cross validated score is 0.8249188514357053
```

Bagging

Bagged KNN

```
from sklearn.ensemble import BaggingClassifier

model = BaggingClassifier(
    estimator=KNeighborsClassifier(n_neighbors=3),
    random_state=0,
    n_estimators=700
)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged KNN is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged KNN is:',result.mean())
```

```
The accuracy for bagged KNN is: 0.832089552238806
The cross validated score for bagged KNN is: 0.8104244694132333
```

Bagged DecisionTree

```
model=BaggingClassifier(estimator=DecisionTreeClassifier(),random_state=0,n_estimators=100)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged Decision Tree is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged Decision Tree is:',result.mean())
```

```
The accuracy for bagged Decision Tree is: 0.8208955223880597
The cross validated score for bagged Decision Tree is: 0.8171410736579275
```

Boosting

AdaBoost(Adaptive Boosting)

```
from sklearn.ensemble import AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.1)
result=cross_val_score(ada,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for AdaBoost is:',result.mean())
```

```
The cross validated score for AdaBoost is: 0.8249188514357055
```

Stochastic Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
grad=GradientBoostingClassifier(n_estimators=500,random_state=0,learning_rate=0.1)
result=cross_val_score(grad,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for Gradient Boosting is:',result.mean())
```

```
The cross validated score for Gradient Boosting is: 0.8115230961298376
```

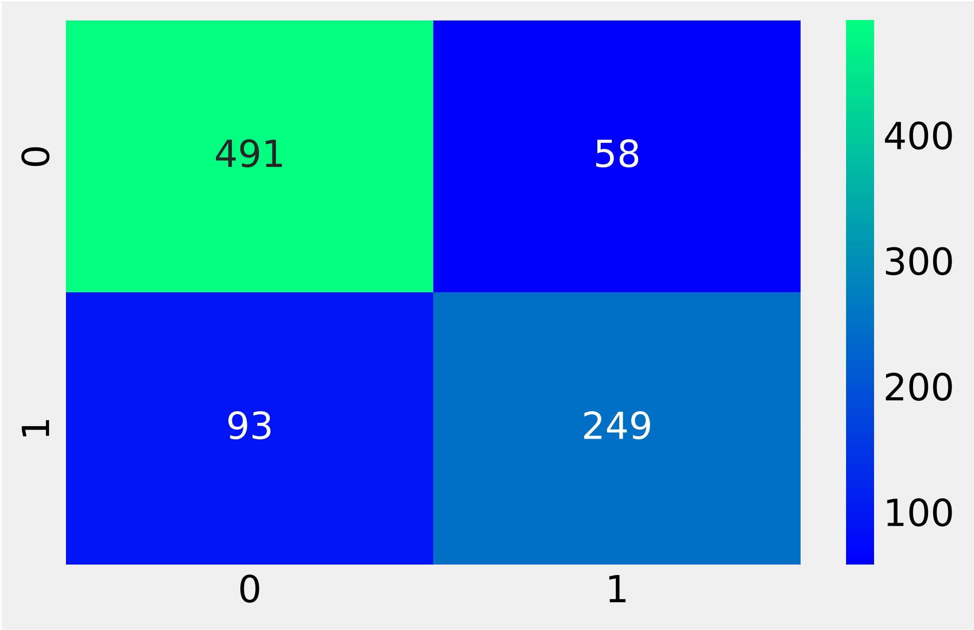
XGBoost

```
import xgboost as xg
xgboost=xg.XGBClassifier(n_estimators=900,learning_rate=0.1)
result=cross_val_score(xgboost,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for XGBoost is:',result.mean())
```

```
The cross validated score for XGBoost is: 0.8160299625468165
```

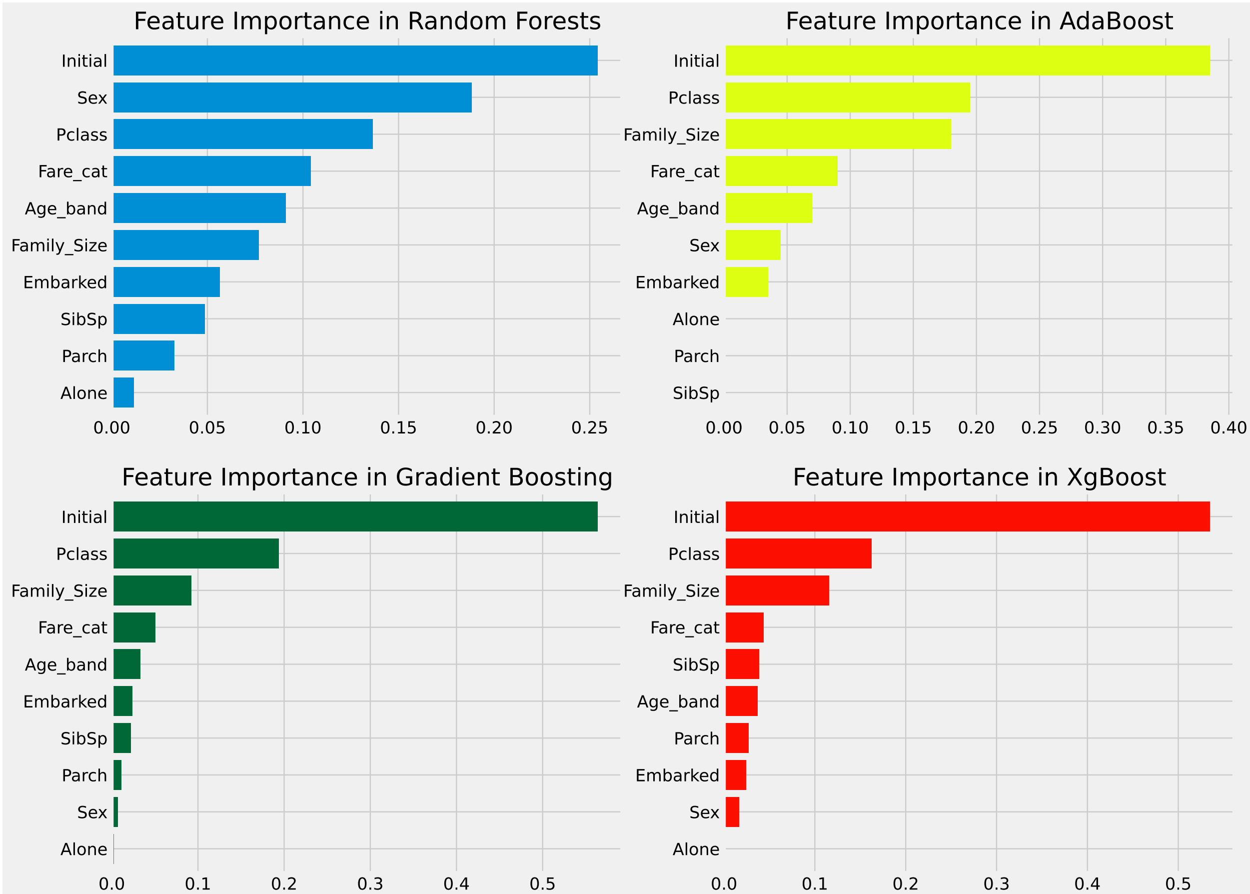
Confusion Matrix for the Best Model

```
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.05)
result=cross_val_predict(ada,X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,result),cmap='winter',annot=True,fmt='2.0f')
plt.show()
```



Feature Importance

```
f,ax=plt.subplots(2,2,figsize=(15,12))
model=RandomForestClassifier(n_estimators=500,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[0
ax[0,0].set_title('Feature Importance in Random Forests')
model=AdaBoostClassifier(n_estimators=200,learning_rate=0.05,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[0
ax[0,1].set_title('Feature Importance in AdaBoost')
model=GradientBoostingClassifier(n_estimators=500,learning_rate=0.1,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[1
ax[1,0].set_title('Feature Importance in Gradient Boosting')
model=xg.XGBClassifier(n_estimators=900,learning_rate=0.1)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,ax=ax[1
ax[1,1].set_title('Feature Importance in XgBoost')
plt.show()
```



Feature Importance in Random Forests
Initial
Sex
Pclass
Fare_cat
Age_band
Family_Size
Embarked
SibSp
Parch
Alone
0.00
0.05
0.10
0.15
0.20
0.25
Feature Importance in AdaBoost
Initial
Pclass
Family_Size
Fare_cat
Age_band
Sex
Embarked
Alone
Parch
SibSp
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Feature Importance in Gradient Boosting
Initial
Pclass
Family_Size
Fare_cat
Age_band
Embarked
SibSp
Parch
Sex
Alone
0.0
0.1
0.2
0.3
0.4
0.5
Feature Importance in XgBoost
Initial
Pclass
Family_Size
Fare_cat
SibSp
Age_band
Parch
Embarked
Sex
Alone
0.0
0.1
0.2
0.3
0.4
0.5