Project : Titanic - Who will survive? - Hyeondeok Cho, Yongjun Cho

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score, recall_score, precision_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics


#Read train and test csv files and convert them to dataframe
titanic_train = pd.read_csv('train.csv')
titanic_test = pd.read_csv('test.csv')
```

In [2]:
```
titanic_train
```

Out[2]:

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	(
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	

891 rows × 12 columns

In [3]:

```
#Check null values
titanic_train.isnull().sum()
```

Out[3]:

```
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
```

It shows that there are 177 rows who have missing values in the column "Age" out of 891 rows, and we decided not to use mean value imputation because there are about 20 % missing values and it may result inaccurate analysis. We are going to use KNN(K-Nearest Neighbor) imputation instead.

In [4]:

```
#clean the dataset, remove the outliers, before any data analysis
#KNN imputation
imputer = KNNImputer(n_neighbors = 2, weights="uniform")
titanic_train['Age'] = imputer.fit_transform(titanic_train[['Age']])
titanic_test['Age'] = imputer.fit_transform(titanic_test[['Age']])
titanic_train
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.92
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.10
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.05
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376	7.75

891 rows × 12 columns

In [5]:

```
#Convert values in the Age column to integer
titanic_train['Age']= round(titanic_train['Age'])
titanic_test['Age']= round(titanic_test['Age'])
titanic_train
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	1	0	3	Braund, Mr. Owen	male	22.0	1	0	A/5 21171	7.2500	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
				Harris							
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	(
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	30.0	1	2	W./C. 6607	23.4500	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	(
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	

891 rows × 12 columns

In [6]:
```
titanic_train.isnull().sum()
```

Out[6]:
```
PassengerId     0
Survived        0
Pclass          0
Name            0
Sex             0
Age             0
SibSp           0
Parch           0
Ticket          0
```

```
Fare            0
Cabin         687
Embarked        2
dtype: int64
```

We don't think that handling missing values in the column "Cabin" is necessary since there are a large amount of missing values and we don't specifically know about its seating chart.

In [7]:
```
#remove the outlier using interquartile range
q1, q3 = np.percentile(titanic_train['Age'],[25,75])
```

In [8]:
```
iqr = q3 -q1
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
lower_bound
```

Out[8]:
```
2.5
```

In [9]:
```
upper_bound
```

Out[9]:
```
54.5
```

In [10]:
```
#we can safely remove the outlier which is greater than upper bound and less tha
clean_titanic_train = titanic_train[titanic_train['Age'] <= upper_bound]
clean_titanic_train = clean_titanic_train[titanic_train['Age'] >= lower_bound]
```

In [11]:
```
clean_titanic_train
```

Out[11]:

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	30.0	1	2	W./C. 6607	23.4500	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	(
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	

825 rows × 12 columns

In [12]:

```
#Extract information from the non-numerical features
#Add the column "Gender" (male = 1, female = 0)
#Drop the column "Sex"

clean_titanic_train['Gender'] = clean_titanic_train["Sex"].apply(lambda x: 1 if
clean_titanic_train.drop("Sex", axis=1, inplace=True)

titanic_test['Gender'] = titanic_test["Sex"].apply(lambda x: 1 if x=="male" else
titanic_test.drop("Sex", axis=1, inplace=True)
```

In [13]:

```
clean_titanic_train.describe()
```

Out[13]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	
count	825.000000	825.000000	825.000000	825.000000	825.000000	825.000000	825.000000	{
mean	447.369697	0.380606	2.341818	28.938182	0.507879	0.357576	31.483615	
std	257.088865	0.485830	0.824096	10.189458	1.090670	0.798599	49.956429	
min	1.000000	0.000000	1.000000	3.000000	0.000000	0.000000	0.000000	
25%	226.000000	0.000000	2.000000	22.000000	0.000000	0.000000	7.895800	
50%	445.000000	0.000000	3.000000	30.000000	0.000000	0.000000	13.416700	
75%	671.000000	1.000000	3.000000	34.000000	1.000000	0.000000	30.070800	
max	891.000000	1.000000	3.000000	54.000000	8.000000	6.000000	512.329200	

In [14]:

```
#check duplicated Passenger ID becasue it must be unique.
clean_titanic_train[clean_titanic_train.duplicated(subset=['PassengerId'])]
```

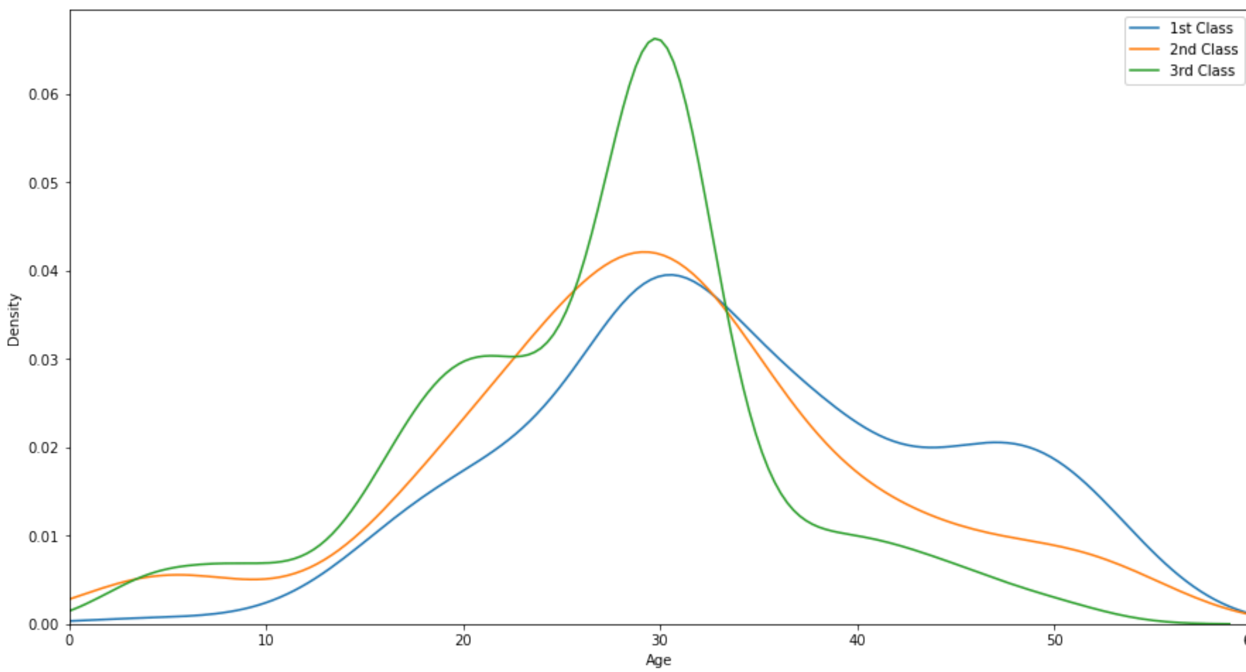
Out[14]:

PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Gend

There is no duplicated passenger ID, and now we finshed cleaning the data and removing the outlier.

In [15]:

```
#Explore the socio-economic status of the passenger, is there any relationship b
# other features, such as age, gender, number of family members on board, etc.

#The one way we can distinguish a passenger's socio-ecnomic status can be ticket
#pay more to reserve good seats, which means that passengers who are in the firs
#social status than others.


#1.relationship between pclass(socio-economic status) and age
#Using KDE(Kernel Density Estimator) plot,
plt.figure(figsize=(15,8))
sns.kdeplot(clean_titanic_train.Age[clean_titanic_train.Pclass == 1], shade=Fals
sns.kdeplot(clean_titanic_train.Age[clean_titanic_train.Pclass == 2], shade=Fals
sns.kdeplot(clean_titanic_train.Age[clean_titanic_train.Pclass == 3], shade=Fals
plt.legend(('1st Class', '2nd Class','3rd Class'),loc='best')
plt.xlim([0, 60])
plt.show()
```

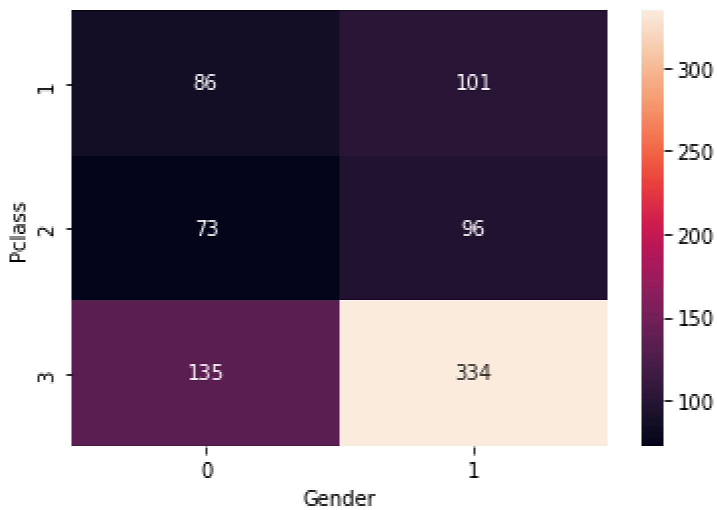


The plot shows that passengers in the age group of 30's had the lowest proportion of first class seats while passengers in the age group opf 40's and 50's had the highest proportion.

In [16]:

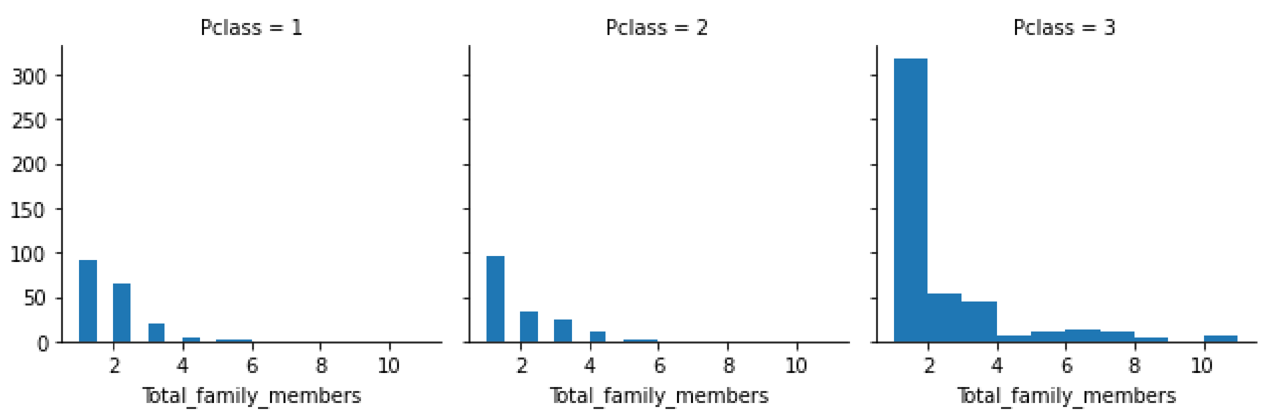
```
#2.relationship between pclass(socio-economic status) and gender
#Using heatmap,

heatmap_df = clean_titanic_train.groupby(['Pclass', 'Gender'])
pclass_gender = heatmap_df.size().unstack()

sns.heatmap(pclass_gender, annot = True, fmt ="d")
```

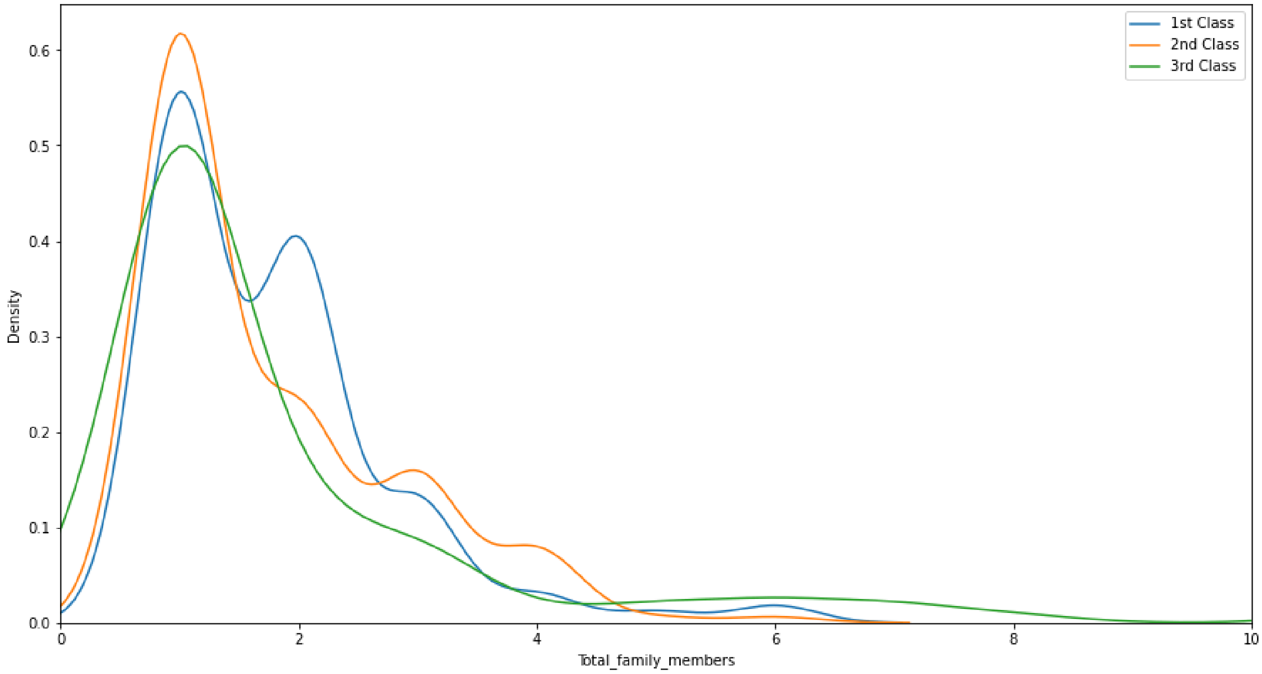
Out[16]: <AxesSubplot:xlabel='Gender', ylabel='Pclass'>



In [17]:

```
#3.relationship between pclass(socio-economic status) and family member
#Add the column "Total family members"
#Total family members = Parch +SibSp + 1(including oneself)
for dt in clean_titanic_train:
    clean_titanic_train['Total_family_members'] = 1 + clean_titanic_train['Parch

S = sns.FacetGrid(data = clean_titanic_train[clean_titanic_train['Total_family_m
S.map(plt.hist, "Total_family_members")

plt.figure(figsize=(15,8))
sns.kdeplot(clean_titanic_train.Total_family_members[clean_titanic_train.Pclass
sns.kdeplot(clean_titanic_train.Total_family_members[clean_titanic_train.Pclass
sns.kdeplot(clean_titanic_train.Total_family_members[clean_titanic_train.Pclass
plt.legend(('1st Class', '2nd Class','3rd Class'),loc='best')
plt.xlim([0, 10])
plt.show()
```





According to the figure above, a passenger whose total family member is 2 has the highest proportion in the first class seats. We guess the reason is that couples came to celebrate for their anniversary and they want better seats for that.

In [18]:
```
# Now we are going to investigate how many couples are in the first class
couples = clean_titanic_train[clean_titanic_train['SibSp'] == 1]
couples = couples[couples['Parch'] == 0]
couples = couples[couples['Pclass'] == 1]
num_of_first_couples = len(couples)
num_of_first_couples
```

Out[18]: 49

In [19]:
```
num_of_first_passengers = clean_titanic_train[clean_titanic_train['Pclass']==1]
num_of_first_passengers = len(num_of_first_passengers)
num_of_first_passengers
```

Out[19]: 187

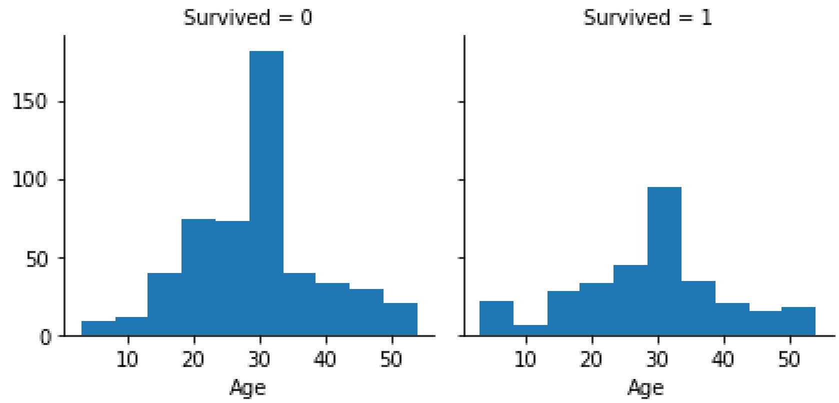
About 30 percent of total number of passengers in the first class are couples, but it is hard to explain the relationship between socio-economic status and number of family members because the number of couples doesn't necessarily tell that they must have higher socio-economic status than others.

Below graphs represents that distribution of survival victims regard to age, gender, Pclass, SibSp, and Parch

In [20]:
```
# Explore the distribution of survival victims in relation to age, gender, socic
# class, etc.

# 1. relationship between Survived and Age
```

```
A = sns.FacetGrid(data = clean_titanic_train[clean_titanic_train['Age'].notna()]
A.map(plt.hist, "Age")
```

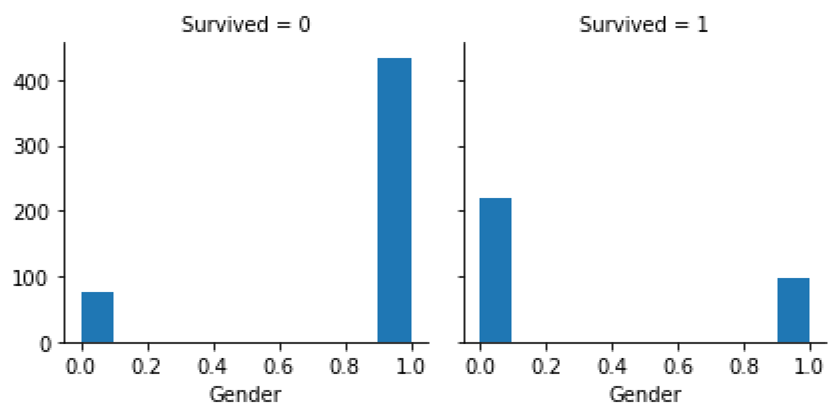
Out[20]: <seaborn.axisgrid.FacetGrid at 0x7fbd3279eaf0>



The number of passengers whose age group is in their 30s represents big portion of total passengers. So, there are a lot of passengers in their 30s who died or survived compared to other age groups.

In [21]:
```
# 2. relationship between Survived and Gender

G = sns.FacetGrid(data = clean_titanic_train[clean_titanic_train['Gender'].notna
G.map(plt.hist, "Gender")
```

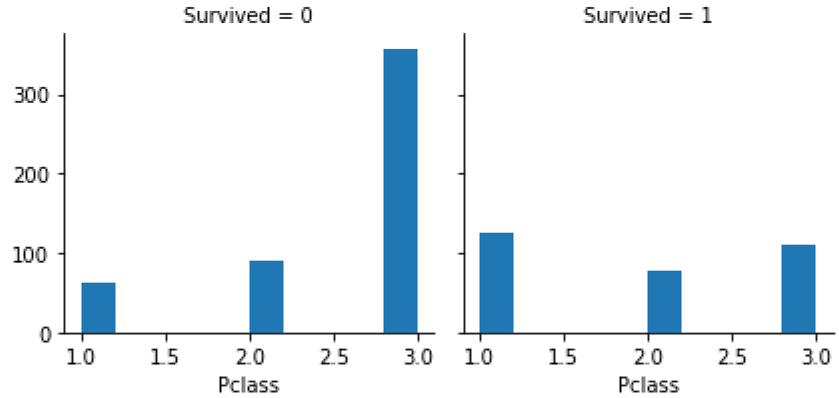
Out[21]: <seaborn.axisgrid.FacetGrid at 0x7fbd3279eca0>



1 represents the male, and we can see that lots of male passengers died. Otherwise, survived rate of female passengers is higher than male passengers.

In [22]:
```
# 3. relationship between Survived and Pclass

C = sns.FacetGrid(data = clean_titanic_train[clean_titanic_train['Pclass'].notna
C.map(plt.hist, "Pclass")
```

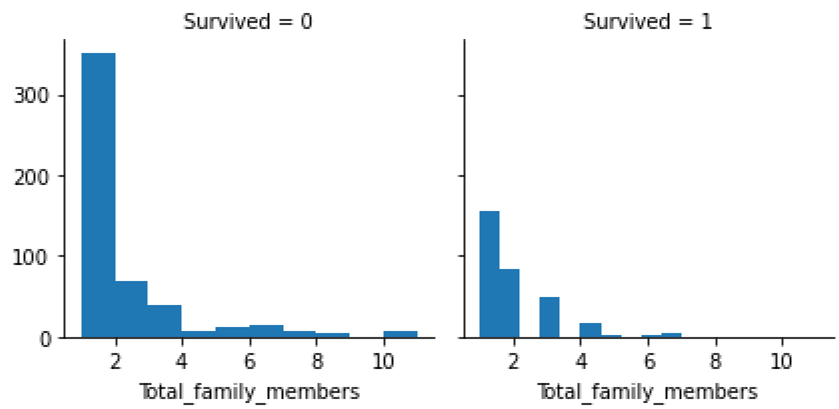
Out[22]: <seaborn.axisgrid.FacetGrid at 0x7fbd4f48e220>



The passengers in first class died less than other classes and survived more compared to passengers in other classes.

In [23]:
```
# 4. relationship between Survived and Total family members

F = sns.FacetGrid(data = clean_titanic_train[clean_titanic_train['Total_family_m
F.map(plt.hist, "Total_family_members")
```

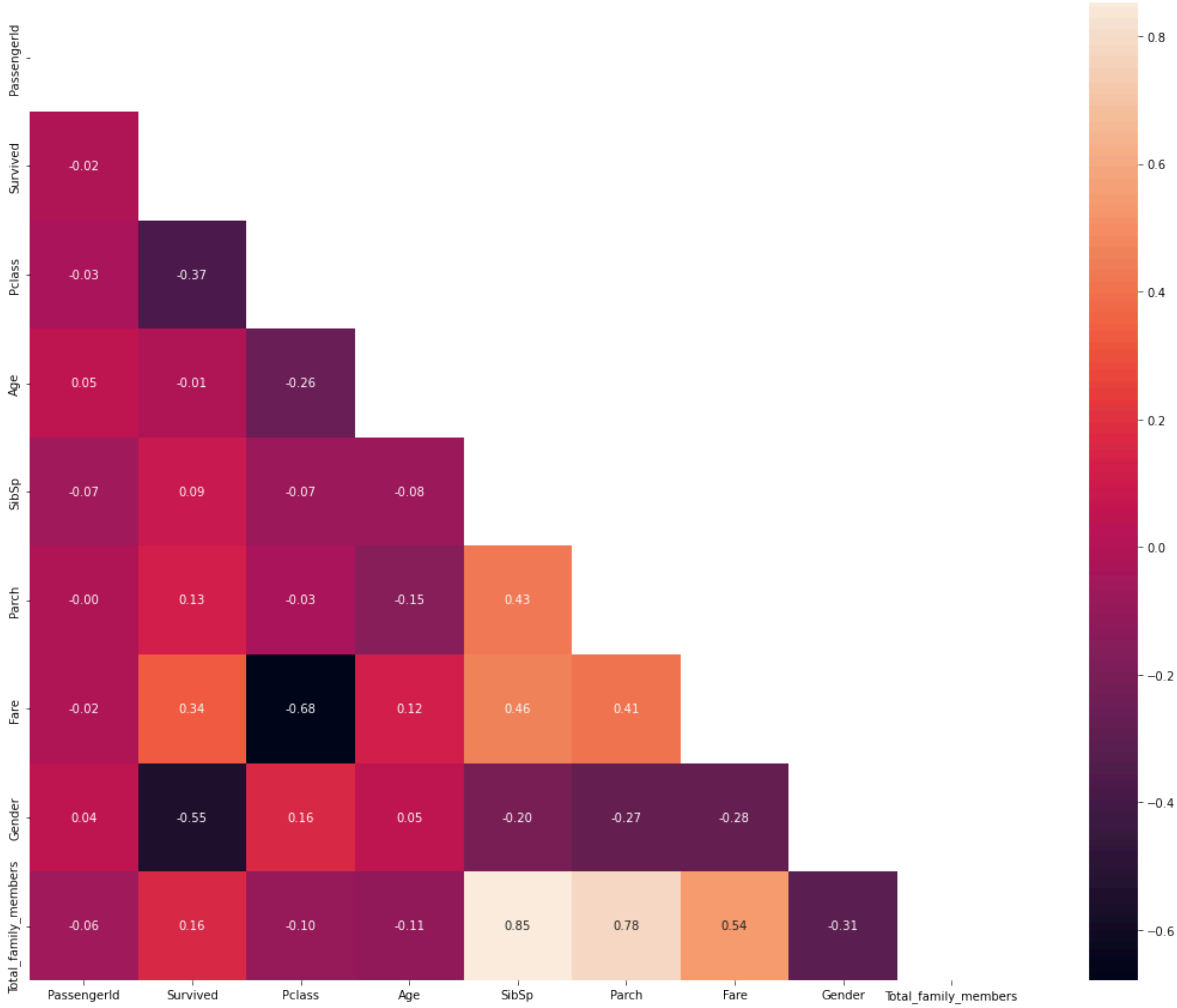
Out[23]: <seaborn.axisgrid.FacetGrid at 0x7fbd32d944c0>



We can see that the number of passengers on board alone accounted for a large portion of total passengers. So, the passengers who boarded alone died or survived more compared to others.

In [24]:
```
# What features seem to be the most important ones? Perform a correlation analys
# before your prediction task.

correlation = clean_titanic_train.corr(method = 'spearman')
mask = np.zeros_like(correlation)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize = (25,15))
sns.heatmap(correlation, annot = True, mask = mask, fmt=".2f", square = True,lir
```

Out[24]: <AxesSubplot:>



The heatmap shows that "Gender", "Pclass" and "Fare" are highly correlated with survival. Specifically, "Gender" is the most related factor to survive, and the correlation -0.55 means that survival rate of female is higher than male's. So, we can use those factors as training and testing features.

In [25]:
```
#Build three models, train them on the training set, and predict the outcome on
#dropping the survival column in the test set). Explain how each model works (br
#the machine learning algorithms behind them).
#Evaluate the performance of each model based on the original outcome in the tes
#not so accurate, what do you think is the reason? Use other evaluation metrics
#models (Precision, Recall, Fscore). Split the data further to include a cross v
#Did this improve your model's performance on the test set?

#1. Build a logistic regression model.
logistic_train_data = clean_titanic_train[70:]
logistic_training_features = logistic_train_data[['Gender', 'Pclass', 'Fare']]
logistic_training_label = logistic_train_data[['Survived']]

#70 testing set
logistic_test_data = clean_titanic_train[:70]
logistic_testing_features = logistic_test_data[['Gender', 'Pclass', 'Fare']]
logistic_testing_label = logistic_test_data[['Survived']]
```

In [26]:
```
#scaling the training and testing features
scaler = StandardScaler()
logistic_training_features = scaler.fit_transform(logistic_training_features)
logistic_testing_features = scaler.transform(logistic_testing_features)
```

In [27]:
```
#build logistic regression model
logisticReg = LogisticRegression()
logisticReg.fit(logistic_training_features, logistic_training_label.values.ravel
logistic_predicted = logisticReg.predict(logistic_testing_features)
```

In [28]:
```
#calculate the acurracy score of the logistic regression model
accuracy_score(logistic_testing_label, logistic_predicted)
```

Out[28]: 0.7714285714285715

In [29]:
```
#calculate the F1 score of the logistic regression model
f1_score(logistic_testing_label, logistic_predicted)
```

Out[29]: 0.7419354838709677

In [30]:
```
#calculate the recall score of the logistic regression model
recall_score(logistic_testing_label, logistic_predicted)
```

Out[30]: 0.7666666666666667

In [31]:
```
#calculate the precision score of the logistic regression model
precision_score(logistic_testing_label, logistic_predicted)
```

Out[31]: 0.71875

In [32]:
```
predicted_df = logistic_testing_label.copy()
predicted_df['Survived'] = logistic_predicted
predicted_df
```

Out[32]:

	Survived
0	0
1	1
2	1
3	1
4	0
...	...
71	1
72	0
73	0

	Survived
74	0
75	0

70 rows × 1 columns

In [33]:
```
#2. Build a KNN model.
knn_train_data = clean_titanic_train[70:]
knn_training_features = knn_train_data[['Gender', 'Pclass', 'Fare']]
knn_training_label = knn_train_data[['Survived']]

#70 testing set
knn_test_data = clean_titanic_train[:70]
knn_testing_features = knn_test_data[['Gender', 'Pclass', 'Fare']]
knn_testing_label = logistic_test_data[['Survived']]
```

In [34]:
```
scaler = StandardScaler()
knn_training_features = scaler.fit_transform(knn_training_features)
knn_testing_features = scaler.transform(knn_testing_features)
```

In [35]:
```
KNN = KNeighborsClassifier(n_neighbors=5)
KNN.fit(knn_training_features, knn_training_label)
```

Out[35]:
```
▾ KNeighborsClassifier ⓘ ?

KNeighborsClassifier()
```

In [36]:
```
knn_predicted = KNN.predict(knn_testing_features)
knn_predicted = knn_predicted.round()
```

In [37]:
```
#calculate the acurracy score of the Knn model
accuracy_score(knn_testing_label, knn_predicted)
```

Out[37]:
```
0.7714285714285715
```

In [38]:
```
#calculate the F1 score of the Knn model
f1_score(knn_testing_label, knn_predicted)
```

Out[38]:
```
0.7037037037037037
```

In [39]:
```
#calculate the recall score of the Knn model
recall_score(knn_testing_label, knn_predicted)
```

Out[39]:
```
0.6333333333333333
```

In [40]:
```
#calculate the precision score of the Knn model
precision_score(knn_testing_label, knn_predicted)
```

Out[40]: 0.7916666666666666

In [41]:
```
prediction_df = knn_testing_label.copy()
prediction_df['Survived'] = knn_predicted
prediction_df
```

Out[41]:

	Survived
0	0
1	1
2	0
3	1
4	0
...	...
71	0
72	0
73	0
74	1
75	0

70 rows × 1 columns

In [42]:
```
#3. Build a Random Forest model.

rfc_train_data = clean_titanic_train[70:]
rfc_training_features =  rfc_train_data[['Gender', 'Pclass', 'Fare']]
rfc_training_label = rfc_train_data[['Survived']]

rfc_test_data = clean_titanic_train[:70]
rfc_testing_features = rfc_test_data[['Gender', 'Pclass', 'Fare']]
rfc_testing_label = rfc_test_data[['Survived']]

rfc = RandomForestClassifier(n_estimators=150)
rfc.fit(rfc_training_features, rfc_training_label)
rfc_predicted = rfc.predict(rfc_testing_features)

rfc_predicted_df = rfc_testing_label.copy()
rfc_predicted_df['Survived'] = rfc_predicted
rfc_predicted_df
```

Out[42]:

	Survived
0	0
1	1

	Survived
2	0
3	1
4	0
...	...
71	0
72	0
73	0
74	1
75	0

70 rows × 1 columns

In [43]:
```
#calculate the acurracy score of the rfc model
accuracy_score(rfc_testing_label, rfc_predicted)
```
Out[43]: 0.7428571428571429

In [44]:
```
#calculate the f1 score of the rfc model
f1_score(rfc_testing_label, rfc_predicted)
```
Out[44]: 0.6896551724137931

In [45]:
```
#calculate the recall score of the rfc model
recall_score(rfc_testing_label, rfc_predicted)
```
Out[45]: 0.6666666666666666

In [46]:
```
#calculate the precision score of the rfc model
precision_score(rfc_testing_label, rfc_predicted)
```
Out[46]: 0.7142857142857143

In [47]:
```
#cross validation (logistic regression model)
cv_scores_logistic = cross_val_score(logisticReg, logistic_training_features, lo
cv_scores_logistic.mean()
```
Out[47]: 0.7867549668874172

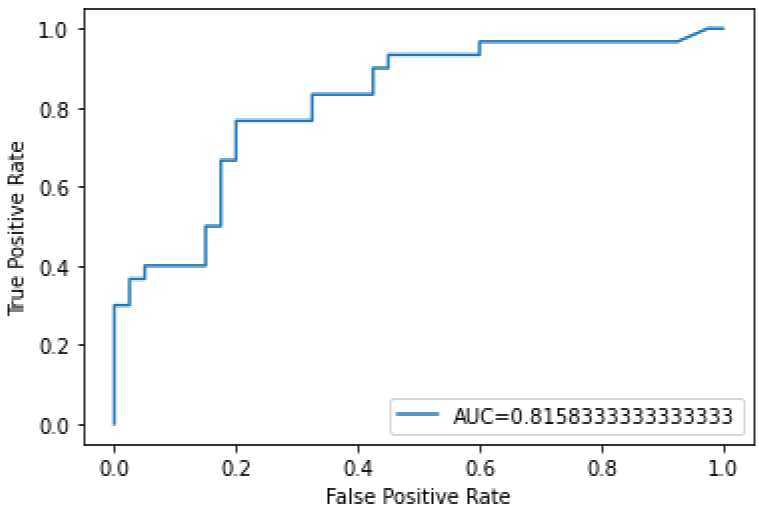
In [48]:
```
#cross validation (knn model)
cv_scores_knn = cross_val_score(KNN, knn_training_features, knn_training_label)
cv_scores_knn.mean()
```
Out[48]: 0.8251655629139073

In [49]:
```
#cross validation (rfc model)
cv_scores_rfc = cross_val_score(rfc, rfc_training_features, rfc_training_label)
cv_scores_rfc.mean()
```

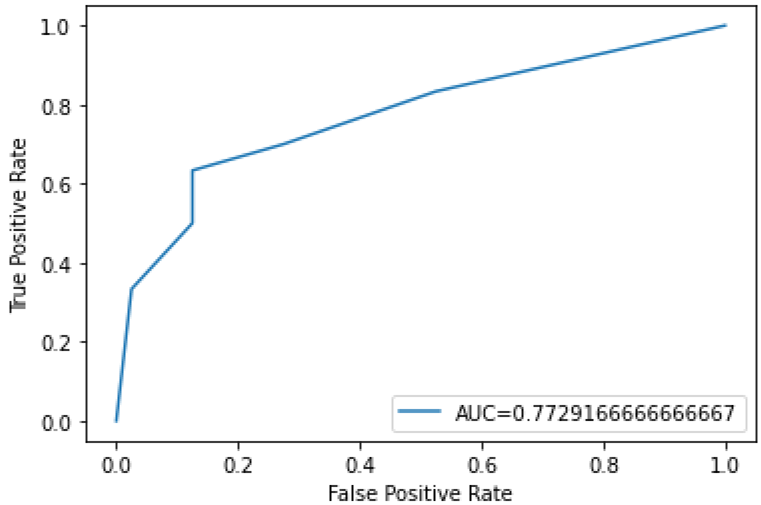
Out[49]: 0.8211920529801324

In [50]:
```
#Compare models
```

In [51]:
```
# ROC curve (logstic model)
label_logistic_predicted = logisticReg.predict_proba(logistic_testing_features)[
fpr, tpr, _ = metrics.roc_curve(logistic_testing_label, label_logistic_predictec
auc = metrics.roc_auc_score(logistic_testing_label, label_logistic_predicted)

plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

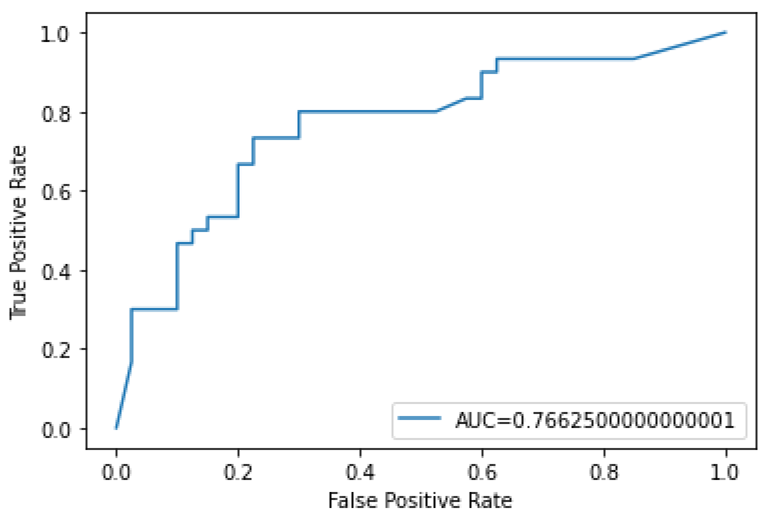


In [52]:
```
# ROC curve (KNN model)
label_knn_predicted = KNN.predict_proba(knn_testing_features)[::,1]
fpr, tpr, _ = metrics.roc_curve(knn_testing_label, label_knn_predicted)
auc = metrics.roc_auc_score(knn_testing_label, label_knn_predicted)

plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



In [53]:

```
# ROC curve (rfc model)
label_rfc_predicted = rfc.predict_proba(rfc_testing_features)[::,1]
fpr, tpr, _ = metrics.roc_curve(rfc_testing_label, label_rfc_predicted)
auc = metrics.roc_auc_score(rfc_testing_label, label_rfc_predicted)

plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



We used ROC curve to compare performance of each model. Since logistic regression model has the biggest AUC(Area Under Curve) value, so we can say that logistic regression model is the best to use out of three models.

In [ ]: