1.Write a method for data preprocessing

In the data preprocessing stage, I did not use the Age filling method which Proferssor Hao provided in his example code. Professor Hao seperately fill the NAN values for survived samples and not survived samples, namely use the information about the label we want to classify in feature data preprocssing, which I believe will cause target leakage since it would be using information that is only available after the event has occurred (i.e., the passenger survived). This could lead to overfitting and inflated performance estimates, since the model is effectively using future information to make predictions.Thus, I simply use the mean of age of all samples(i.e., no matter sample survives or not) to fill Age Nan values.

In [81]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn import svm
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.neighbors import KNeighborsClassifier

def getNumber(str):
    if str=="male":
        return 1
    else:
        return 2

def getS(str):
    if str=='S':
        return 1
    else:
        return 0

def getQ(str):
        if str=='Q':
            return 1
        else:
            return 0

def getC(str):
        if str=='C':
            return 1
        else:
            return 0
```

In [45]:
```
def preprocess(train):
    del train['Name']
    del train['Cabin']
    del train['Fare']
    del train['Ticket']

    train["Gender"]=train["Sex"].apply(getNumber)
    del train['Sex']
    samplemean=train.Age.mean()
```

```
    train['age']=np.where(pd.isnull(train.Age), samplemean, train.Age)

    del train['Age']
    train.rename(columns={'age':'Age'}, inplace=True)
    train.dropna(inplace=True)

    train['S']=train['Embarked'].apply(getS)
    train['Q']=train['Embarked'].apply(getQ)
    train['C']=train['Embarked'].apply(getC)
    del train['Embarked']
    del train['PassengerId']
    return train
```

2.Read and process the data for training

In [46]:
```
train=pd.read_csv('train.csv',header = 0, dtype={'Age': np.float64})
train=preprocess(train)
```

In [47]:
```
train.head()
```

Out[47]:

	Survived	Pclass	SibSp	Parch	Gender	Age	S	Q	C
0	0	3	1	0	1	22.0	1	0	0
1	1	1	1	0	2	38.0	0	0	1
2	1	3	0	0	2	26.0	1	0	0
3	1	1	1	0	2	35.0	1	0	0
4	0	3	0	0	1	35.0	1	0	0

3.Read and process the data for testing

In [48]:
```
test=pd.read_csv('test.csv',header = 0, dtype={'Age': np.float64})
test=preprocess(test)
```

In [49]:
```
test.head()
```

Out[49]:

	Pclass	SibSp	Parch	Gender	Age	S	Q	C
0	3	0	0	1	34.5	0	1	0
1	3	1	0	2	47.0	1	0	0
2	2	0	0	1	62.0	0	1	0
3	3	0	0	1	27.0	1	0	0
4	3	1	1	2	22.0	1	0	0

4.Models

4.1 Logistic Regression

In [78]:
```
X = train[['Pclass', 'SibSp', 'Parch', 'Gender', 'Age', 'S', 'Q', 'C']]

y = train['Survived']

# create a logistic regression model
Logisticmodel = LogisticRegression(max_iter=1000, C=1.0)
y_pred = cross_val_predict(Logisticmodel, X, y, cv=5)

# Compute the performance metrics for the predicted target values and the true target va
cm = confusion_matrix(y, y_pred)
report = classification_report(y, y_pred)
auc = roc_auc_score(y, y_pred)

# Print out the performance metrics
print("Confusion matrix:")
print(cm)
print('\n')
print("AUC:", auc)
print('\n')
print("Classification report:")
print(report)
```

```
Confusion matrix:
[[468  81]
 [104 236]]


AUC: 0.7732883317261331


Classification report:
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       549
           1       0.74      0.69      0.72       340

    accuracy                           0.79       889
   macro avg       0.78      0.77      0.78       889
weighted avg       0.79      0.79      0.79       889
```

4.2 SVM

In [80]:
```
# create a logistic regression model
SVMmodel = svm.SVC()
y_pred = cross_val_predict(SVMmodel, X, y, cv=5)

# Compute the performance metrics for the predicted target values and the true target va
cm = confusion_matrix(y, y_pred)
report = classification_report(y, y_pred)
auc = roc_auc_score(y, y_pred)

# Print out the performance metrics
print("Confusion matrix:")
print(cm)
print('\n')
```

```
print("AUC:", auc)
print('\n')
print("Classification report:")
print(report)
```

```
Confusion matrix:
[[535  14]
 [304  36]]


AUC: 0.540190721097182


Classification report:
              precision    recall  f1-score   support

           0       0.64      0.97      0.77       549
           1       0.72      0.11      0.18       340

    accuracy                           0.64       889
   macro avg       0.68      0.54      0.48       889
weighted avg       0.67      0.64      0.55       889
```

4.3 KNN

In [88]:
```
# Create a KNN classifier and find best K
def knnresult():
    maxauc=0
    bestk=1
    for i in range(1,20):

        KNNmodel = KNeighborsClassifier(n_neighbors=i)
        y_pred = cross_val_predict(KNNmodel, X, y, cv=5)
        auc = roc_auc_score(y, y_pred)

        if auc>maxauc:
            maxauc=auc
            bestk=i


    KNNmodel = KNeighborsClassifier(n_neighbors=bestk)
    y_pred = cross_val_predict(KNNmodel, X, y, cv=5)

    # Compute the performance metrics for the predicted target values and the true targ
    cm = confusion_matrix(y, y_pred)
    report = classification_report(y, y_pred)
    auc = roc_auc_score(y, y_pred)
    print('the best k is',bestk,'\n')
    # Print out the performance metrics
    print("Confusion matrix:")
    print(cm)
    print('\n')
    print("AUC:", auc)
    print('\n')
    print("Classification report:")
    print(report)

knnresult()
```

```
the best k is 3

Confusion matrix:
[[465  84]
 [111 229]]


AUC: 0.7602619736419158


Classification report:
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       549
           1       0.73      0.67      0.70       340

    accuracy                           0.78       889
   macro avg       0.77      0.76      0.76       889
weighted avg       0.78      0.78      0.78       889
```

For all 3 models, we are using the same 8 features: 'Pclass', 'SibSp', 'Parch', 'Gender', 'Age', 'S', 'Q', 'C'. By comparing these three models we can see that logistic regression model performs the best. Thus we use it to predict on our testing set.

5.Prediction for Testing Set

In [90]:
```
X_test = test[['Pclass', 'SibSp', 'Parch', 'Gender', 'Age', 'S', 'Q', 'C']]
#Fit the Model
X = train[['Pclass', 'SibSp', 'Parch', 'Gender', 'Age', 'S', 'Q', 'C']]
y = train['Survived']
Logisticmodel.fit(X, y)
# Make predictions on the test data
y_test_pred = Logisticmodel.predict(X_test)
```

In [93]:
```
#Add predictions to df and export as xlsx
result=pd.read_csv('test.csv',header = 0, dtype={'Age': np.float64})
result.insert(1, 'Survived', y_test_pred)
result.to_excel('result.xlsx', index=False)
```