1454	2915	92020.874720
1455	2916	97106.062563
1456	2917	192688.359392
1457	2918	99234.950627
1458	2919	218286.205289

1459 rows × 2 columns

In [60]:

```
my_submission.to_csv('Assignment2home.csv', index=False)
##1459
```

Assignment 2 - Part 2 | Titanic

In [61]:

```
##Importing Data
titanic_trainDat = pd.read_csv('Titanic/train.csv')
titanic_testDat =  pd.read_csv('Titanic/test.csv')
```

In [62]:

```
titanic_trainDat.shape
titanic_testDat.shape
```

Out[62]:

```
(891, 12)
```

Out[62]:

```
(418, 11)
```

In [63]:

```
# checking variables to make sure survivied is the y variable
set(titanic_trainDat.columns).difference(set(titanic_testDat.columns))
```

Out[63]:

```
{'Survived'}
```

In [64]:

```
# Are the features in the test data a subset of the train data features?
set(titanic_testDat.columns).issubset(set(titanic_trainDat.columns))
```

Out[64]:

```
True
```

In [65]:

```
## Understanding the data
titanic_trainDat.info()
titanic_trainDat.head(3)
titanic_trainDat.describe()
```

```
<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
```

Out[65]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

Out[65]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [67]:

```
## Isolating numerical and categorical variables for potential use later
titanic_categoricals = titanic_trainDat[['Survived','Pclass','Sex',\
                                         'Ticket','Cabin','Embarked']]
titanic_numericals = titanic_trainDat[['Age','SibSp','Parch','Fare']]
```

In [68]:

```
#Commented for cleaner PDF export

# Understanding data distributions
#import warnings
#warnings.filterwarnings("ignore")

#for i in titanic_numericals:
    #plt.hist(titanic_numericals[i])
    #plt.title(i)
    #plt.show()
```

In [69]:

```
#Commeneted for cleaner PDF export

## Understanding data distributions of categoricals
#for i in titanic_categoricals:
    #sns.barplot(titanic_categoricals[i].value_counts().index,\
                #titanic_categoricals[i].value_counts()).set_title(i)
    #plt.show()
```

EDA

- After some initial analysis of the data, below is my plan.
    - Drop unnecessary columns (PassengerID, Name)
    - I am also going to drop ticket as I don't think it has any revalence. Data seems rather sporatic.
    - I am going to feature engineer Cabin section (letter)
    - Going to impute numerical data with either mean/median
    - Going to impute most categorical with mode

In [70]:

```
total = titanic_trainDat.isnull().sum().sort_values(ascending = False)
pcg = (total / titanic_trainDat.isnull().count()).sort_values\
(ascending = False)
miss_val = pd.concat([total, pcg], axis = 1, keys = ['Total', 'Percentage'])
miss_val.head(20)
```

Out[70]:

	Total	Percentage
Cabin	687	0.771044
Age	177	0.198653
Embarked	2	0.002245
PassengerId	0	0.000000
Survived	0	0.000000
Pclass	0	0.000000
Name	0	0.000000
Sex	0	0.000000
SibSp	0	0.000000
Parch	0	0.000000
Ticket	0	0.000000
Fare	0	0.000000

In [71]:

```
## Created a function for easier use later with test data
```

```
def titanic_eda(df):
    df['Cabin']=df['Cabin'].fillna('NA')
    #Feature Engineer a new variable with only cabin area(letter)
    df['cabinLetter'] = df['Cabin'].str[:1]
    df['Embarked']=df['Embarked'].fillna(df['Embarked'].mode()[0])
    df['Pclass']=df['Pclass'].fillna(df['Pclass'].mode()[0])
    df['Sex']=df['Sex'].fillna(df['Sex'].mode()[0])
    df['SibSp']=df['SibSp'].fillna(df['SibSp'].median())
    df['Parch']=df['Parch'].fillna(df['Parch'].median())
    df['Age']=df['Age'].fillna(df['Age'].median())
    df['Fare']=df['Fare'].fillna(df['Fare'].mean())
    df.drop(['PassengerId','Ticket','Name','Cabin'],axis=1,inplace=True)
    return(df)
```

In [72]:

```
titanic_trainDat = titanic_eda(titanic_trainDat)
```

In [73]:

```
sum(titanic_trainDat.isnull().sum())
```

Out[73]:

```
0
```

In [74]:

```
titanic_trainDat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Survived     891 non-null    int64  
 1   Pclass       891 non-null    int64  
 2   Sex          891 non-null    object 
 3   Age          891 non-null    float64
 4   SibSp        891 non-null    int64  
 5   Parch        891 non-null    int64  
 6   Fare         891 non-null    float64
 7   Embarked     891 non-null    object 
 8   cabinLetter  891 non-null    object 
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

Transformation

- Similar to above, I have selected the appropriate numericals and categoricals to standardize and encode

In [75]:

```
### Encoding
# Col transform specs

titanic_ct = ColumnTransformer([('standardized',\
                                 preprocessing.StandardScaler(),[2,3,4,5]),
       ('oneHotter', preprocessing.OneHotEncoder\
        (handle_unknown='ignore'),[0,1,6,7])])

titanic_ct
```

Out[75]:

```
ColumnTransformer(transformers=[('standardized', StandardScaler(),
                                 [2, 3, 4, 5]),
                                ('oneHotter',
                                 OneHotEncoder(handle_unknown='ignore'),
                                 [0, 1, 6, 7])])
```

In [76]:

```
titanic_y=titanic_trainDat.Survived.to_numpy(copy=True)
titanic_X=titanic_trainDat.loc[:,titanic_trainDat.columns!='Survived']\
.to_numpy(copy=True)
titanic_X.shape   # size
titanic_y.shape   # size
#titanic_X
```

Out[76]:

```
(891, 8)
```

Out[76]:

```
(891,)
```

In [77]:

```
#For easier accuracy calcuations.
#Selected this metric for performance measurement
from sklearn.metrics import accuracy_score
```

Modeling Summary

- **Logistic Regression**
  - Using K-Fold Cross Validation
  - Accuracy: 0.811
  - Use the same explicit approach shown in sync session.
  - This seems to make sense and more desirable than pipeline although technically not was efficient
- **LDA**
  - Used split/train/test 80-20 Hold Out Cross Validation
  - Accuracy: 0.777
- **KNN**
  - Also used 80/20 Hold-Out Cross Validation
  - Neighbors = 5
  - Accuracy: 0.782

In [78]:

```
resListofDicts=[]                        # a list of results in dicts
  # Outer processing loop
fold=0          # fold counter
for trainNdx, testNdx in kf.split(titanic_X):# cv loop. should do it 10 times.
    fold+=1
    Xtr = titanic_ct.fit_transform(titanic_X[trainNdx])\
    # fit & transform X training fold
    Xval = titanic_ct.transform(titanic_X[testNdx])   \
    # transform X test fold
    logMod=LogisticRegression(\
                 max_iter=2000)  # instantiate regressor
    fitMod=logMod.fit(Xtr,titanic_y[trainNdx])           # fitted
    predtr = fitMod.predict(Xtr)            # training pred values
    predval = fitMod.predict(Xval)          # test pred values
    msetr = accuracy_score(titanic_y[trainNdx],predtr)
    #mseval = metrics.mean_squared_error(y[testNdx],predval)
    resDict={'fold': fold,
            'Accuracy':msetr}
    resListofDicts.append(resDict)
```

In [79]:

```
resultsDF=pd.DataFrame(resListofDicts)
#resultsDF.shape
#resultsDF.columns
resultsDF
```

Out[79]:

	fold	Accuracy
0	1	0.821473
1	2	0.814214
2	3	0.820449
3	4	0.817955
4	5	0.820449
5	6	0.824190
6	7	0.817955
7	8	0.820449
8	9	0.811721
9	10	0.812968

In [80]:

```
# Random split using a scikit-learn preprocessing method
Xtrain, Xtest, ytrain, ytest = train_test_split(titanic_X, titanic_y, \
        train_size=0.8, random_state=9)

Xtrain.shape
Xtest.shape
ytrain.shape
ytest.shape
```

Out[80]:

```
(712, 8)
```

Out[80]:

```
(179, 8)
```

Out[80]:

```
(712,)
```

Out[80]:

```
(179,)
```

In [81]:

```
lda = LinearDiscriminantAnalysis(n_components=1)

#X_train = lda.fit_transform(Xtrain, ytrain)
```

```
titanic_lda_trained = lda.fit(titanic_ct.fit_transform(Xtrain),ytrain)

y_pred = titanic_lda_trained.predict(titanic_ct.transform(Xtest))
accuracy_score(ytest,y_pred)
```

Out[81]:

```
0.776536312849162
```

In [82]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
titanic_knn_trained = knn_classifier.fit\
(titanic_ct.fit_transform(Xtrain),ytrain)

y_pred = titanic_knn_trained.predict(titanic_ct.transform(Xtest))
accuracy_score(ytest,y_pred)
#from sklearn.metrics import classification_report, confusion_matrix
#print(confusion_matrix(ytest,y_pred))
#print(classification_report(ytest,y_pred))
```

Out[82]:

```
0.7821229050279329
```

Implement/Submission

I am going to use logistic regresssion approach for the submission because it provided the highest accuracy

In [83]:

```
titanic_testDat =  pd.read_csv('Titanic/test.csv')
```

In [84]:

```
# Called the EDA function above before transformations
titanic_testDat = titanic_eda(titanic_testDat)
```

In [85]:

```
sum(titanic_testDat.isnull().sum())
```

Out[85]:

```
0
```

In [86]:

```
titanic_final_exam=titanic_testDat.to_numpy(copy=True)
```

In [87]:

```
lg = LogisticRegression(max_iter = 2000)
#titanic_log_trained = lg.fit(titanic_ct.fit_transform(Xtrain),ytrain)

y_pred = fitMod.predict(titanic_ct.transform(titanic_final_exam))
```

In [88]:

```
df_test = pd.read_csv('Titanic/test.csv')
Regresult = pd.DataFrame(y_pred, columns=['Survived'])
df_test['PassengerId'].shape, Regresult.shape
test_t = pd.DataFrame(df_test["PassengerId"])
```

Out[88]:

```
((418,), (418, 1))
```

In [89]:

```
my_submission = pd.concat([test_t, Regresult ], axis=1)
```

In [90]:

```
my_submission
```

Out[90]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [91]:

```
my_submission.to_csv('Assignment2-Titanicc.csv', index=False)
```

In [ ]: