In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV files
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
training = pd.read_csv('titanic/train.csv')
test = pd.read_csv('titanic/test.csv')

training['train_test'] = 1
test['train_test'] = 0
test['Survived'] = np.NaN
all_data = pd.concat([training,test])

%matplotlib inline
all_data.columns
```

Out[2]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'train_test'],
      dtype='object')
```

In [3]:
```
# Understand nature of the data .info() .describe()
# Histogram and boxplots
# Value counts
# Missing data
# Correlation between the metrics
# Eplore interesting themes
        # Wealthy survive?
        # By location
        # Age scatterplot with ticket price
        # Young and wealthy Variable?
        # Total spent?
# Feature engineering
# preprocess data together or use a transformer?
        #use label for train and test
# Scaling

# Model Baseline
# Model comparison with CV
```

**Data Exploration

1) For numeric data

```
- Made histograms to understand distribution
- Corrplot
- Pivot table comparing survival rate across numeric variables
```

2) For categorical data

```
- Made bar charts to understand balance of classes
- Made pivot tables to understand relationship with survival
```

In [4]:
```
# Look into data types & null counts - Understand types of datas and null values
training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 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
 12  train_test   891 non-null    int64
dtypes: float64(2), int64(6), object(5)
memory usage: 90.6+ KB
```

In [5]:

```
# Understand the numeric data. Use the .describe() method. This gives an understanding
training.describe()
```

Out[5]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	train_test
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000	891.0
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208	1.0
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429	0.0
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	1.0
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400	1.0
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200	1.0
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000	1.0
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	1.0

In [6]:

```
# Seperate the numeric columns
training.describe().columns
```

Out[6]:

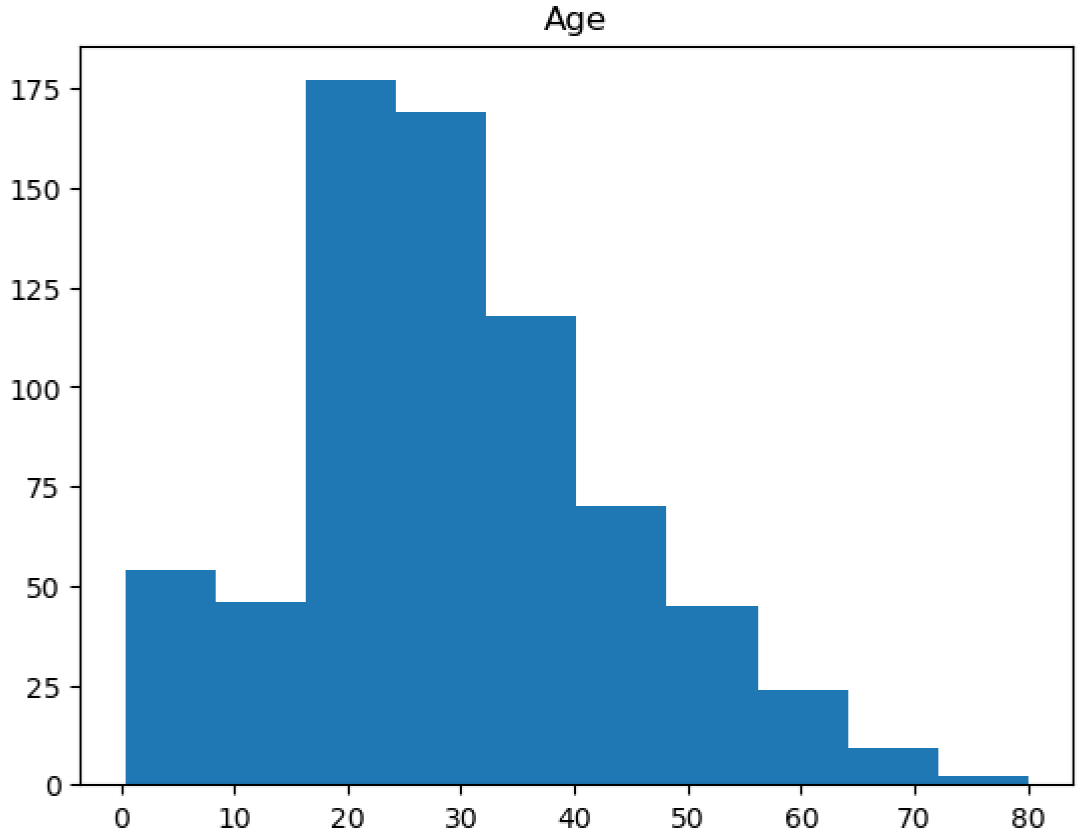
```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
       'train_test'],
      dtype='object')
```

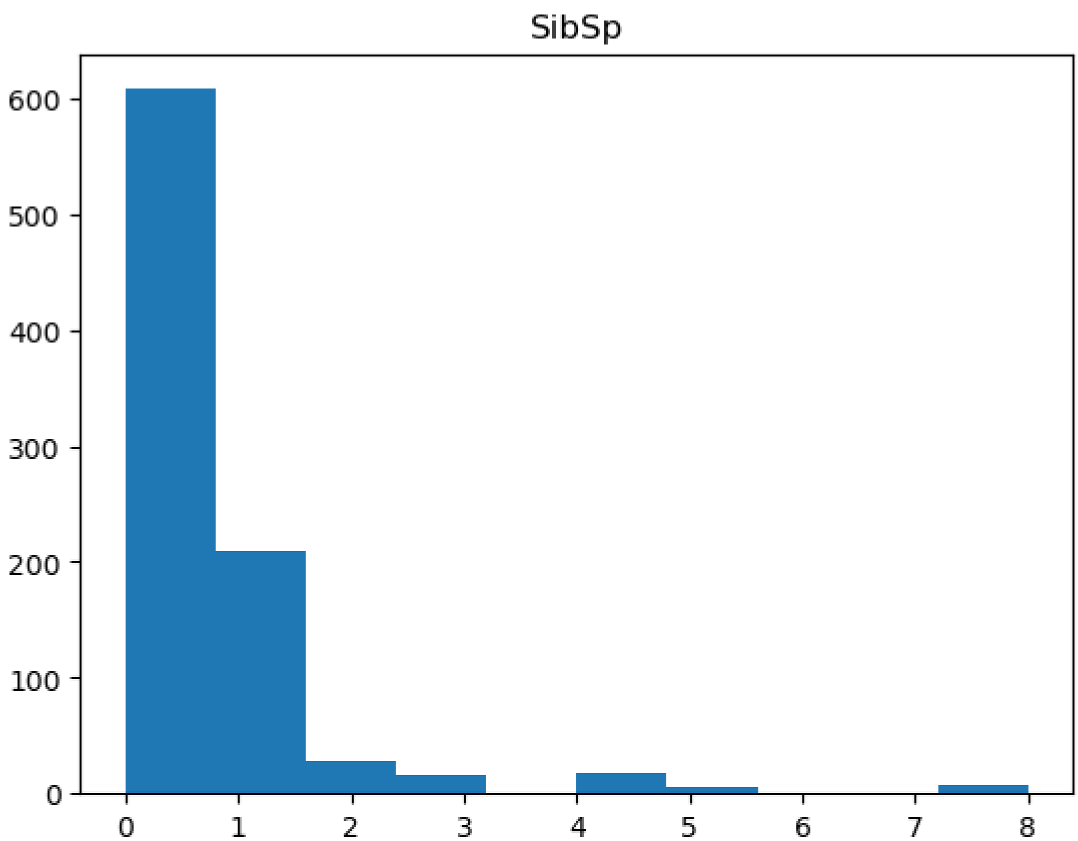
In [7]:

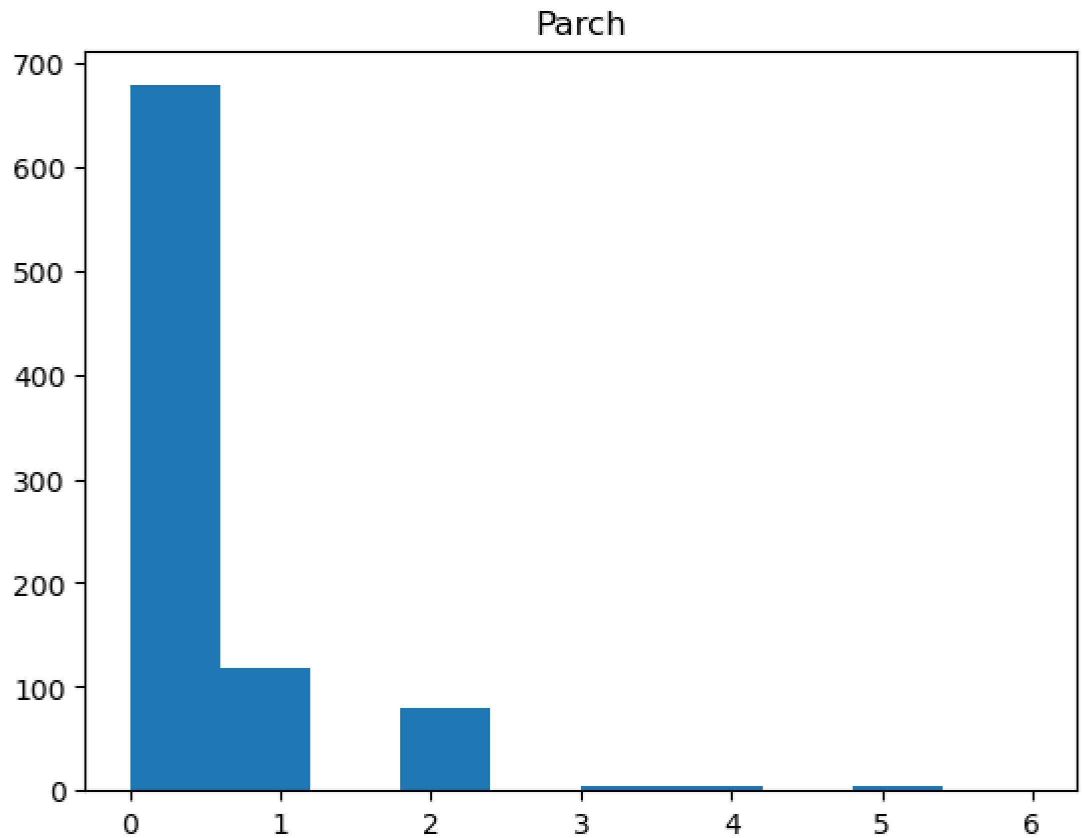
```
# df_num. Seperate values into numeric variables (For histograms)
# df_cat. Seperate values into categorical variables (For values counts)
df_num = training[['Age','SibSp','Parch','Fare']]
df_cat = training[['Survived','Pclass','Sex','Ticket','Cabin','Embarked']]
```

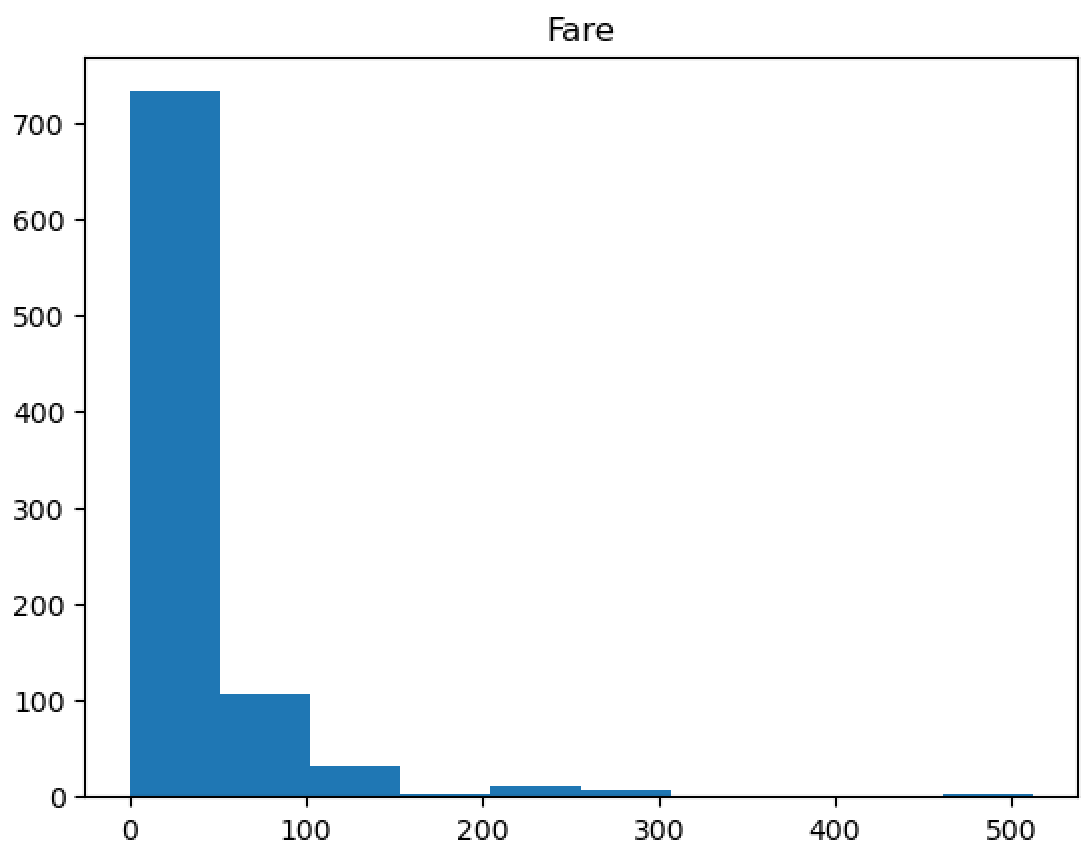
In [8]:

```
# Take a look at all the numeric variables
for i in df_num.columns:
    plt.hist(df_num[i])
    plt.title(i)
    plt.show()
```


Age
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70
80


SibSp
600
500
400
300
200
100
0
0
1
2
3
4
5
6
7
8





Age shows a normal distribution. Survival highest between 20 to 40 year olds.

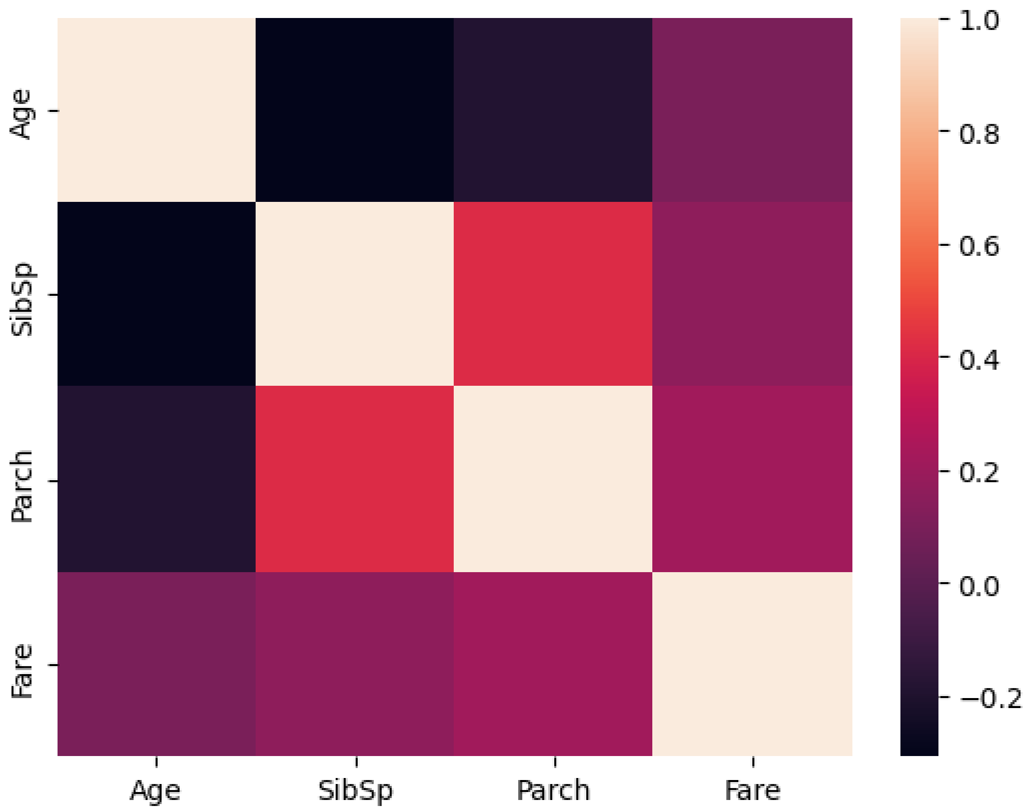
SibSp, Parch, and Fare didn't.

```
- Normalize and scaling
```

In [9]:
```
print(df_num.corr())
sns.heatmap(df_num.corr())
```

```
            Age     SibSp     Parch      Fare
Age    1.000000 -0.308247 -0.189119  0.096067
SibSp -0.308247  1.000000  0.414838  0.159651
Parch -0.189119  0.414838  1.000000  0.216225
Fare   0.096067  0.159651  0.216225  1.000000
```

Out[9]: <AxesSubplot:>



Heatmap shows the correlation between variables.

```
- Purple didicate lower correlation.
- SibSp/Parch have medium relationship
```

In [10]:
```
# Compare survival rate across Age, SibSp, Parch, and Fare
pd.pivot_table(training, index = 'Survived', values =['Age','SibSp','Parch','Fare'])
```

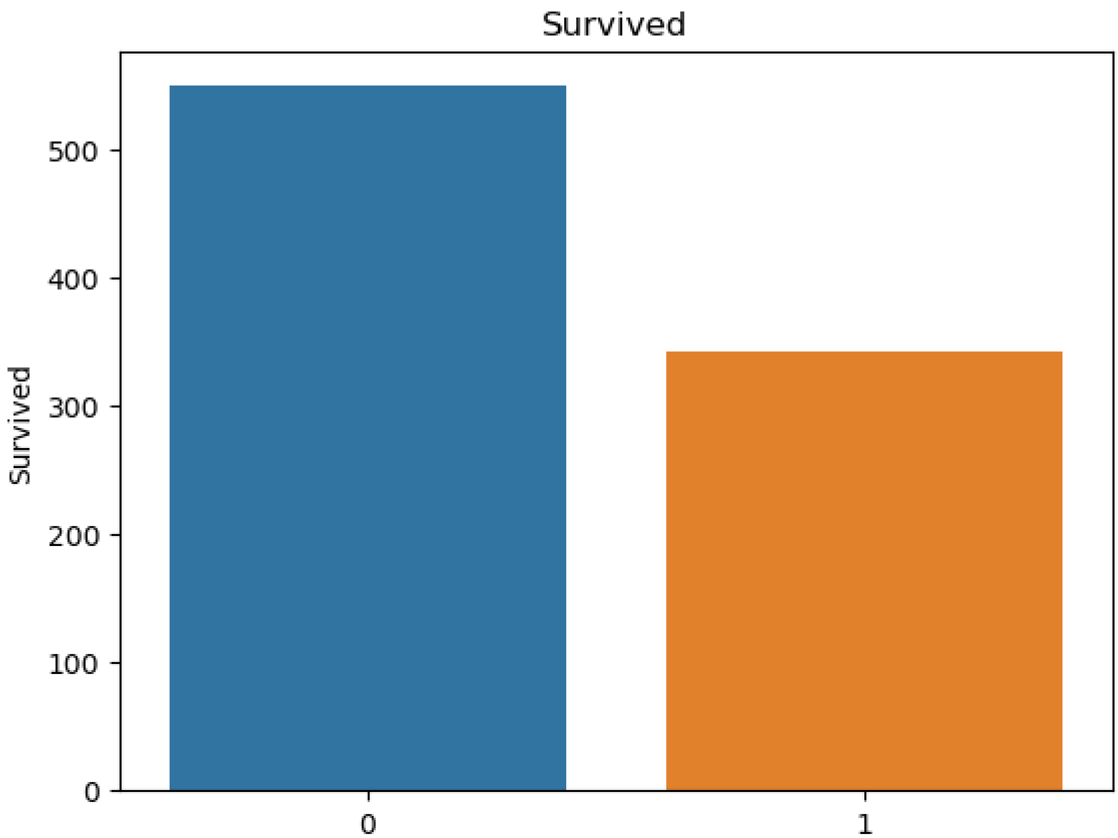
Out[10]:

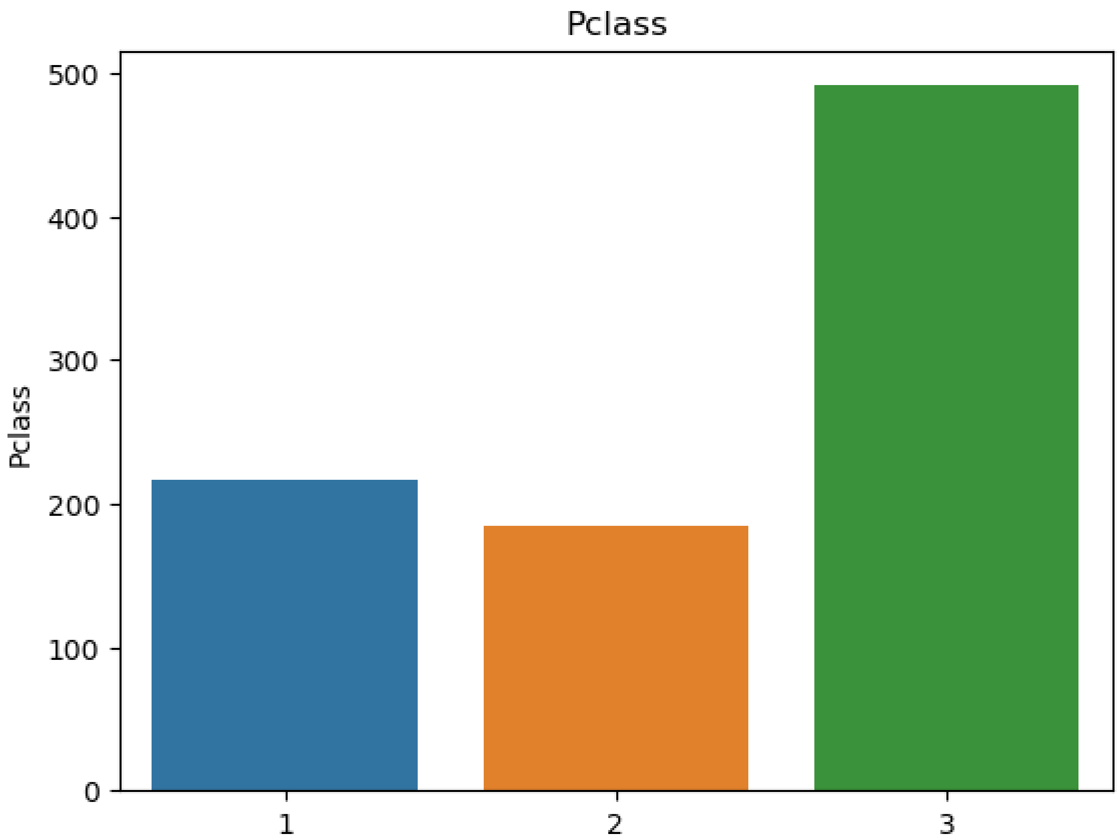
	Age	**Fare**	**Parch**	**SibSp**
Survived				
0	30.626179	22.117887	0.329690	0.553734
1	28.343690	48.395408	0.464912	0.473684

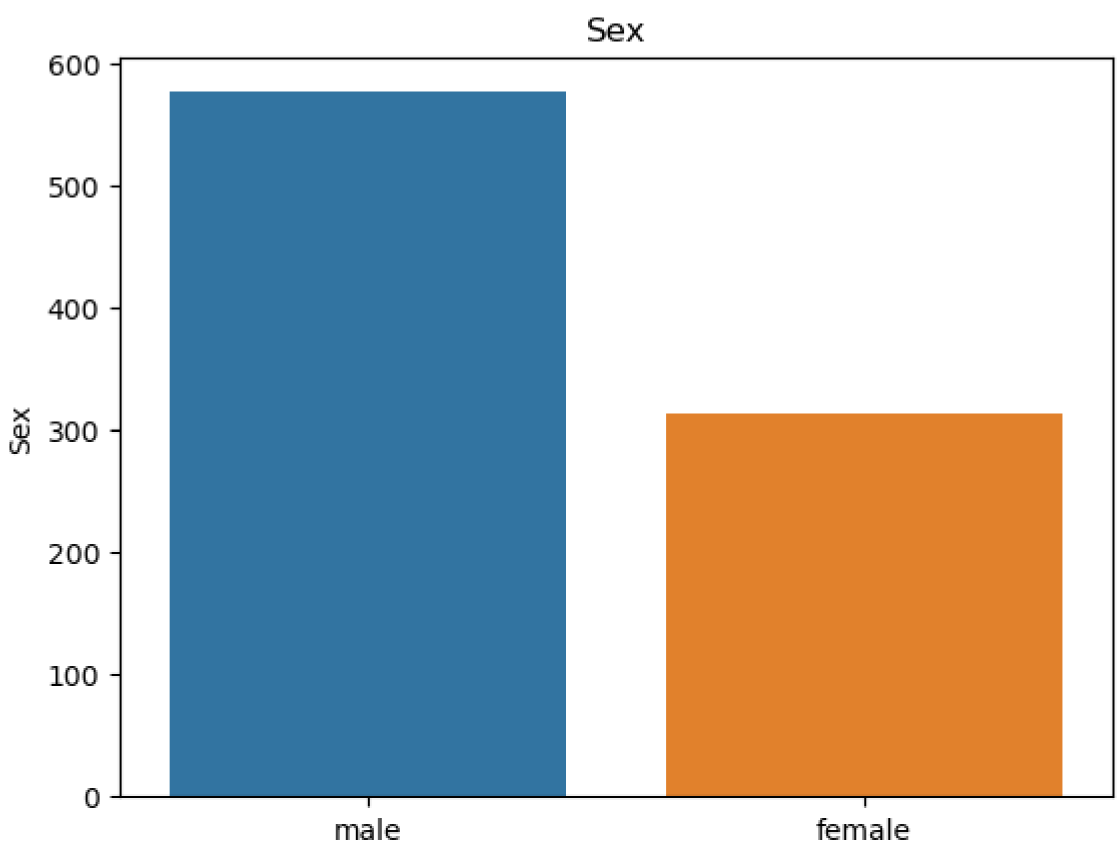
Survival rate across indexed groups.

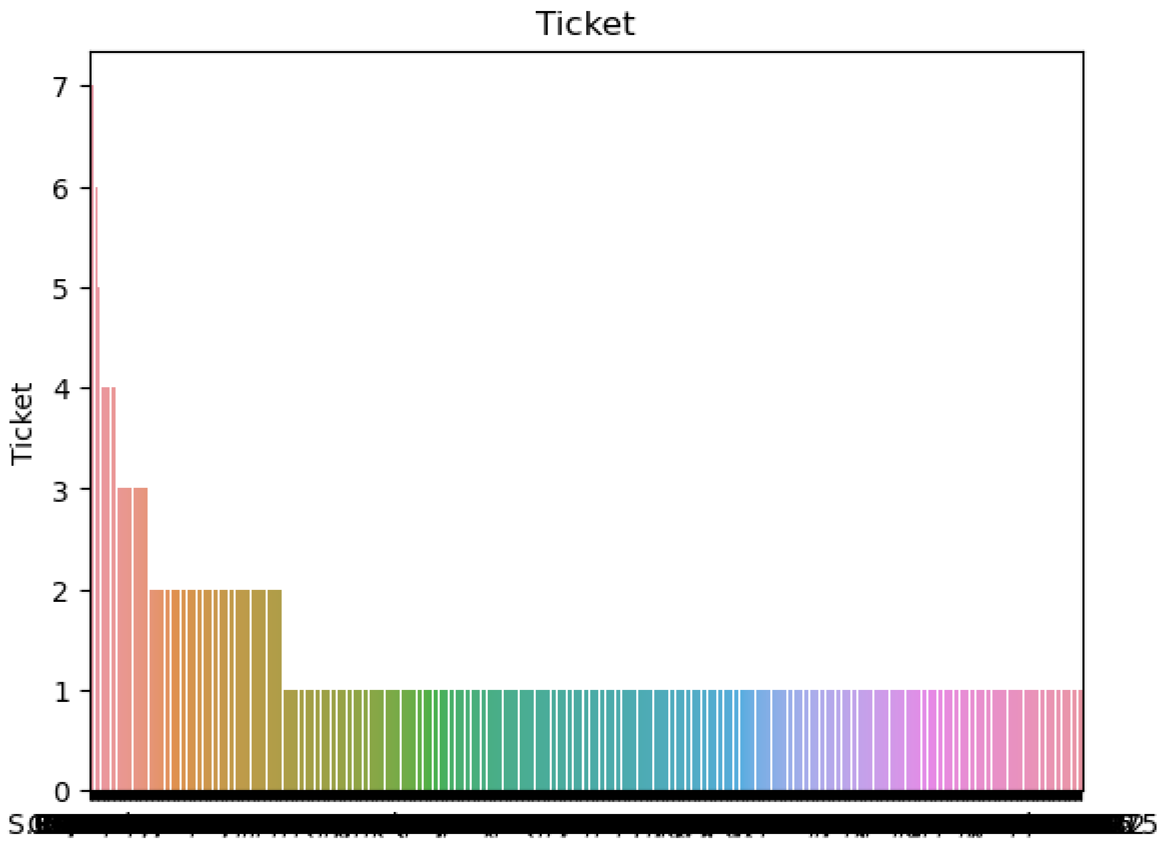
```
- Younger people survived. Avg, 28
- Higher fares surived. Avg, $48
- Younger children higher chance survived
- Older passenger less change survived
```

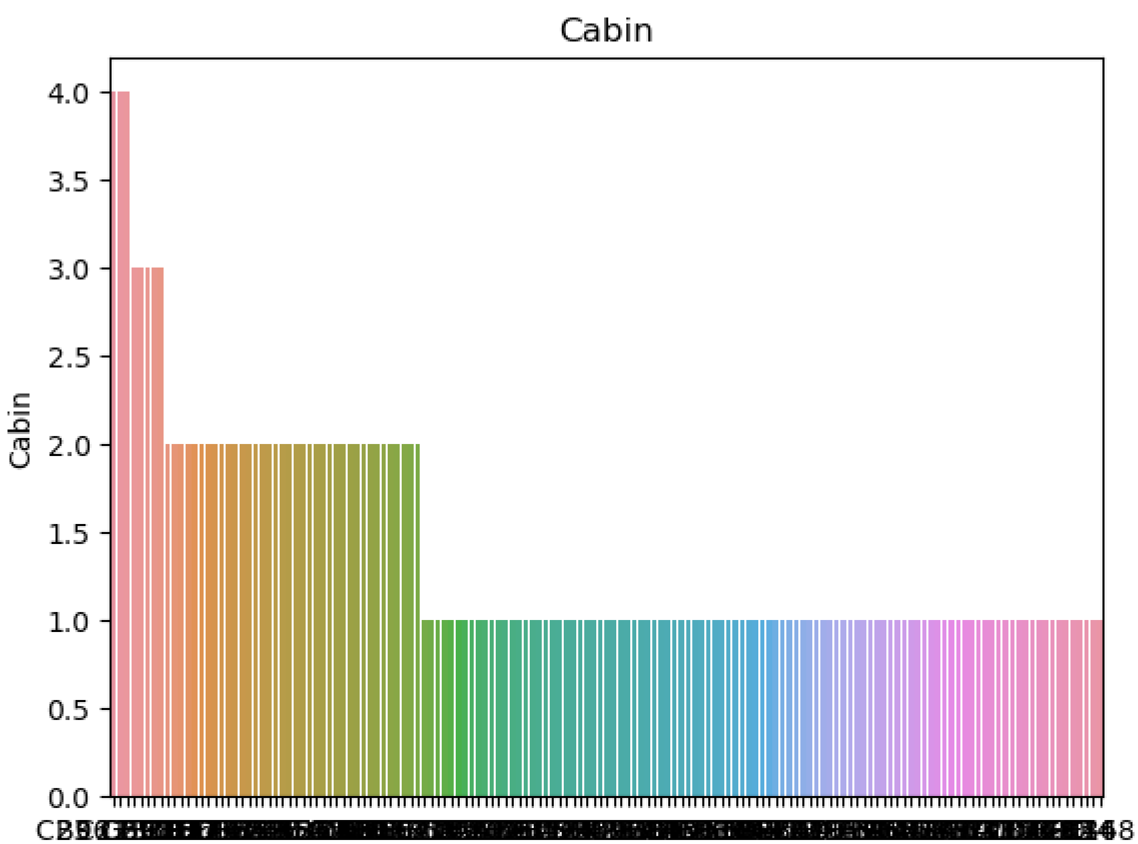
In [11]:
```
for i in df_cat.columns:
    sns.barplot(x=df_cat[i].value_counts().index, y=df_cat[i].value_counts()).set_titl
    plt.show()
```

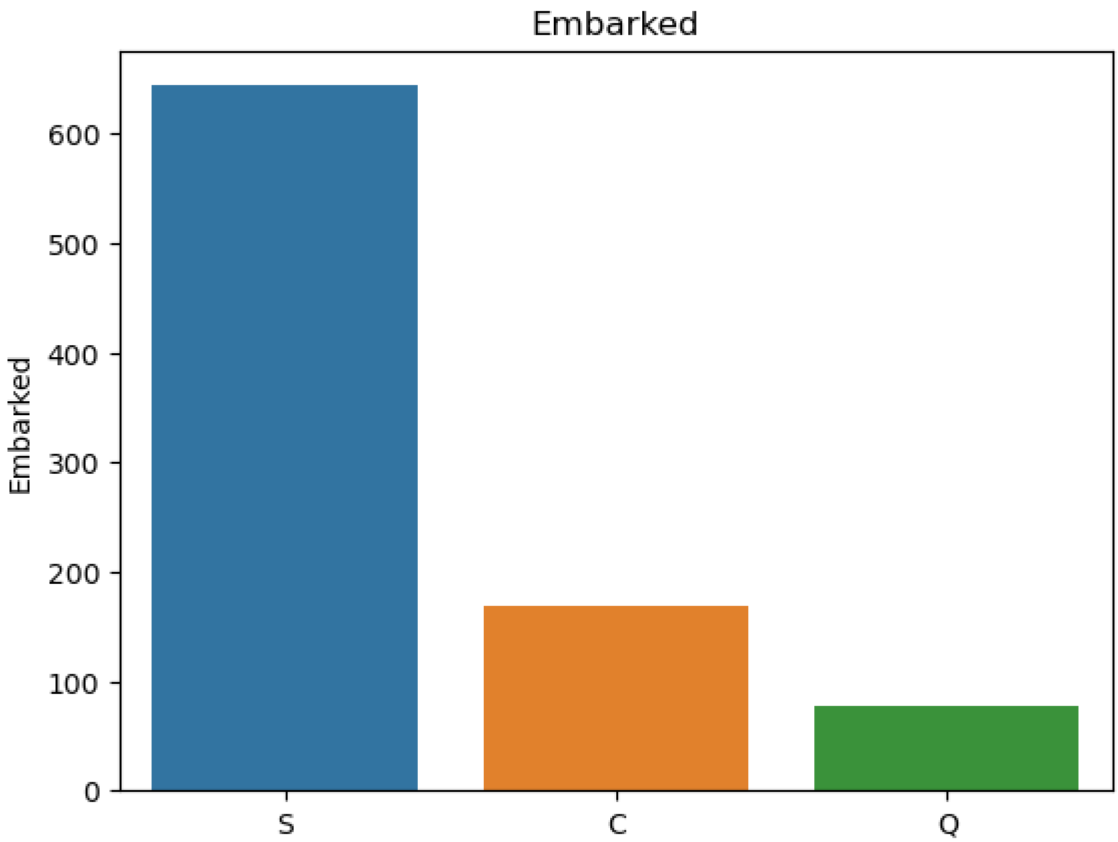



Pclass
500
400
300
200
100
0
Pclass
1
2
3


Sex
600
500
400
300
200
100
0
Sex
male
female


Ticket
Ticket
0
1
2
3
4
5
6
7


Cabin
Cabin
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



In [12]:
```
# Comparing survivaland each of these categorical variables

print(pd.pivot_table(training, index = 'Survived', columns = 'Pclass', values = 'Ticke
print()

print(pd.pivot_table(training,index = 'Survived', columns = 'Sex', values = 'Ticket',
print()

print(pd.pivot_table(training,index = 'Survived', columns = 'Embarked', values = 'Tick
```

```
Pclass      1   2    3
Survived
0          80  97  372
1         136  87  119

Sex       female  male
Survived
0             81   468
1            233   109

Embarked   C   Q    S
Survived
0         75  47  427
1         93  30  217
```

Feature Engineering

1. Cabin - Simplify cabins (evaluated if cabin letter (cabin_adv) or the purchase of tickets across multiple cabins (cabin_multiple) impacted survival)

1. Tickets - Do different ticket types impact survival rates?

1. Does a person's title relate to survival rates?

In [13]:
```
df_cat.Cabin
training['cabin_multiple'] = training.Cabin.apply(lambda x: 0 if pd.isna(x) else len(x
# after looking at this, we may want to look at cabin by letter or by number. Let's cr
# letters
# multiple letters
training['cabin_multiple'].value_counts()
```

Out[13]:
```
0    687
1    180
2     16
3      6
4      2
Name: cabin_multiple, dtype: int64
```

In [14]:
```
pd.pivot_table(training, index = 'Survived', columns = 'cabin_multiple', values = 'Tic
```

Out[14]:

cabin_multiple	0	1	2	3	4
Survived					
0	481.0	58.0	7.0	3.0	NaN
1	206.0	122.0	9.0	3.0	2.0

In [15]:
```
# creates categories based on the cabin letter (n stands for null)
# in this case we will treat nulll values like it's own category

training['cabin_adv'] = training.Cabin.apply(lambda x: str(x)[0])
```

In [16]:
```
#comparing survival rate by cabin
print(training.cabin_adv.value_counts())
pd.pivot_table(training,index='Survived',columns='cabin_adv',values = 'Name', aggfunc=
```

```
n    687
C     59
B     47
D     33
E     32
A     15
F     13
G      4
T      1
Name: cabin_adv, dtype: int64
```

Out[16]:

cabin_adv	A	B	C	D	E	F	G	T	n
Survived									
0	8.0	12.0	24.0	8.0	8.0	5.0	2.0	1.0	481.0
1	7.0	35.0	35.0	25.0	24.0	8.0	2.0	NaN	206.0

In [17]:
```
#understand ticket values better
#numeric vs non numeric

training['numeric_ticket'] = training.Ticket.apply(lambda x: 1 if x.isnumeric() else 0
```

```
training['ticket_letters'] = training.Ticket.apply(lambda x: ''.join(x.split(' ')[:-1]
('.','').replace('/','').lower() if len(x.split(' ')[: -1]) >0 else 0)
```

In [18]:
```
training['numeric_ticket'].value_counts()
```

Out[18]:
```
1    661
0    230
Name: numeric_ticket, dtype: int64
```

In [19]:
```
# View all rows in dataframe through scrolling

pd.set_option("display.max_rows", None)
training['ticket_letters'].value_counts()
```

Out[19]:
```
0            665
pc            60
ca            41
a5            21
stono2        18
sotonoq       15
scparis       11
wc            10
a4             7
soc            6
fcc            5
c              5
sopp           3
pp             3
wep            3
ppp            2
scah           2
sotono2        2
swpp           2
fc             1
scahbasle      1
as             1
sp             1
sc             1
scow           1
fa             1
sop            1
sca4           1
casoton        1
Name: ticket_letters, dtype: int64
```

In [20]:
```
#Difference in numeric vs non-numeric tickets in survival rate

pd.pivot_table(training,index = 'Survived', columns = 'numeric_ticket', values = 'Tick
```

Out[20]:

numeric_ticket	0	1
Survived		
0	142	407
1	88	254

In [21]:
```
# Survival rate across different ticket types

pd.pivot_table(training,index = 'Survived', columns = 'ticket_letters', values = 'Tick
```

Out[21]:

ticket_letters	0	a4	a5	as	c	ca	casoton	fa	fc	fcc	...	soc	sop	sopp	sotonc
Survived															
0	410.0	7.0	19.0	1.0	3.0	27.0	1.0	1.0	1.0	1.0	...	5.0	1.0	3.0	2
1	255.0	NaN	2.0	NaN	2.0	14.0	NaN	NaN	NaN	4.0	...	1.0	NaN	NaN	Na

2 rows × 29 columns

In [22]:

```
# Feature engineering on person's title

training.Name.head(50)
training['name_title'] = training.Name.apply(lambda x: x.split(',')[1].split('.')[0].s

#mr., ms., master. etc
```

In [23]:

```
training['name_title'].value_counts()

#There aren't many special surnames
```

Out[23]:

```
Mr              517
Miss            182
Mrs             125
Master           40
Dr                7
Rev               6
Mlle              2
Major             2
Col               2
the Countess      1
Capt              1
Ms                1
Sir               1
Lady              1
Mme               1
Don               1
Jonkheer          1
Name: name_title, dtype: int64
```

Data Preprocessing for Model

1) Drop null values from Embarked (only 2)

2) Include only relevant variables (Since we have limited data, I wanted to exclude things like name and passanger ID so that we could have a reasonable number of features for our models to deal with)

Variables: Pclass, Sex, Age, SibSp, Parch, Fare, Embarked, cabin_adv, cabin_multiple, numeric_ticket, name_title

3) Do categorical transforms on all data. Usually we can use a transformer, but this approach we can just make sure the training and test data have the same columns. We can also infer

something about the shake of the test data through this method.

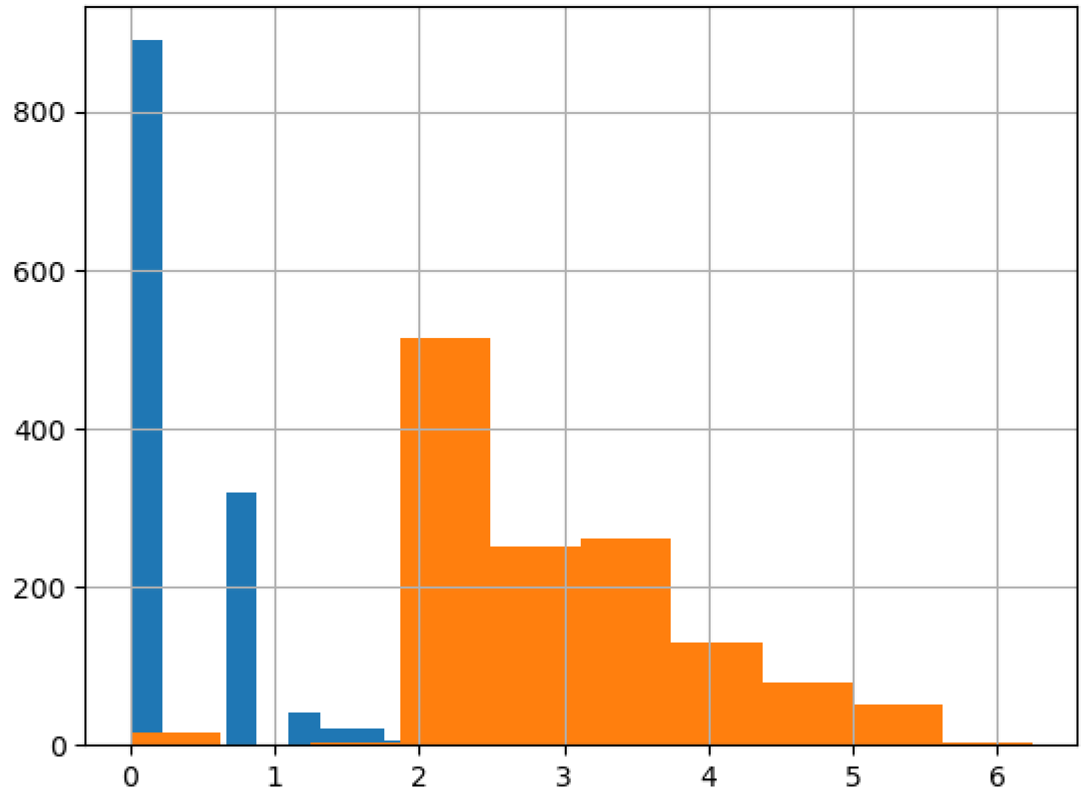
4) Impute data with mean for fare and age (Can also experiment with median)

5) Normalized fare using logarithm to give more semblance of a normal distribution

6) Scaled data 0-1 with standard scaler

In [24]:
```
# Create all categorical variables that we did above for both training and test sets

all_data['cabin_multiple'] = all_data.Cabin.apply(lambda x: 0 if pd.isna(x) else len(x
all_data['cabin_adv'] = all_data.Cabin.apply(lambda x: str(x)[0])
all_data['numeric_ticket'] = all_data.Ticket.apply(lambda x: 1 if x.isnumeric() else 0
all_data['ticket_letters'] = all_data.Ticket.apply(lambda x: ''.join(x.split(' ')[:-1]
                                                   ('/','').lower() if len(x.split(' '
all_data['name_title'] = all_data.Name.apply(lambda x: x.split(',')[1].split('.')[0].s

#impute nulls for continuous data
all_data.Age = all_data.Age.fillna(training.Age.mean())
all_data.Fare = all_data.Fare.fillna(training.Fare.mean())

#drop null embarked rows. Only 2 instances of this in training and 0 in test
all_data.dropna(subset = ['Embarked'],inplace = True)

#tried log norm of sibsp (not used)
all_data['norm_sibsp'] = np.log(all_data.SibSp+1)
all_data['norm_sibsp'].hist()

#log norm of fare (used)
all_data['norm_fare'] = np.log(all_data.Fare+1)
all_data['norm_fare'].hist()

#converted fare to category for pd.get_dummies()
all_data.Pclass = all_data.Pclass.astype(str)

#created dummy variables from categories (also can use OneHotEncoder)
all_dummies = pd.get_dummies(all_data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'norm
                                       , 'cabin_multiple', 'numeric_ticket', 'name_tit

#split to train test again
X_train = all_dummies[all_dummies.train_test == 1].drop(['train_test'], axis = 1)
X_test = all_dummies[all_dummies.train_test == 0].drop(['train_test'], axis = 1)

y_train = all_data[all_data.train_test == 1].Survived
y_train.shape
```

Out[24]:
```
(889,)
```



In [25]:
```
# Scale data

from sklearn.preprocessing import StandardScaler
scale = StandardScaler ()
all_dummies_scaled = all_dummies.copy()
all_dummies_scaled[['Age','SibSp','Parch','norm_fare']] = scale.fit_transform(all_dumm
all_dummies_scaled

X_train_scaled = all_dummies_scaled[all_dummies_scaled.train_test == 1].drop(['train_t
X_test_scaled = all_dummies_scaled[all_dummies_scaled.train_test == 0].drop(['train_te

y_train = all_data[all_data.train_test ==1].Survived
```

Model Building (Baseline Validation Performance)

Let see how various different models perform with default parameters. The following models using cross validation to get a baseline. With a validation set baseline, we can see how much tuning improves each of the mdoels. Just because a model has a higher baseline on this validation set doesn't mean that it will do better on the eventual test set.

- Naive Bayes 72%
- Logistic Regression 82%
- Decision Tree 77%
- K Nearest Neighbor 81%
- Random Forest 80%

- Support Vector Classifier **83%**
- Xtreme Gradient Boosting 81%
- Soft Voting Classifier - All Models 82%

In [26]:
```
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [27]:
```
# Naive Bayes as as baseline for classification tasks

gnb = GaussianNB()
cv = cross_val_score(gnb,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.66853933 0.70224719 0.75842697 0.74719101 0.73446328]
0.7221735542436362
```

In [28]:
```
lr = LogisticRegression(max_iter = 2000)
cv = cross_val_score(lr,X_train,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.82022472 0.80898876 0.80337079 0.82022472 0.85310734]
0.8211832666793626
```

In [29]:
```
lr = LogisticRegression(max_iter = 2000)
cv = cross_val_score(lr,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.82022472 0.80898876 0.80337079 0.82022472 0.85310734]
0.8211832666793626
```

In [30]:
```
dt = tree.DecisionTreeClassifier(random_state = 1)
cv = cross_val_score(dt,X_train,y_train, cv=5)
print(cv)
print(cv.mean())
```

```
[0.74719101 0.74157303 0.80898876 0.75842697 0.82485876]
0.776207706468609
```

In [31]:
```
dt = tree.DecisionTreeClassifier(random_state = 1)
cv = cross_val_score(dt,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.74719101 0.74157303 0.80898876 0.75280899 0.81920904]
0.7739541674601662
```

In [32]:
```
knn = KNeighborsClassifier()
cv = cross_val_score(knn,X_train,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.76966292 0.80898876 0.80337079 0.81460674 0.83615819]
0.8065574811147084
```

```
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [33]:

```
knn = KNeighborsClassifier()
cv = cross_val_score(knn,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.79775281 0.79213483 0.83146067 0.80337079 0.85310734]
0.8155652891512728
```

```
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\mnDen\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the defaul
t behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, thi
s behavior will change: the default value of `keepdims` will become False, the `axis`
over which the statistic is taken will be eliminated, and the value None will no long
er be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [34]:

```
rf = RandomForestClassifier(random_state = 1)
cv = cross_val_score(rf,X_train,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.82022472 0.78651685 0.85393258 0.73033708 0.84180791]
0.8065638291119152
```

In [35]:

```
rf = RandomForestClassifier(random_state = 1)
cv = cross_val_score(rf,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.81460674 0.78651685 0.85393258 0.73033708 0.84180791]
0.8054402336062972
```

In [36]:

```
svc = SVC(probability = True)
cv = cross_val_score(svc,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.84831461 0.82022472 0.8258427  0.80337079 0.86440678]
0.8324319177299563
```

In [37]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(random_state =1)
cv = cross_val_score(xgb,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.8258427  0.80898876 0.84831461 0.78651685 0.81920904]
0.8177743921792675
```

In [38]:
```
# Votiong classifier take all of the inputers and average the results. For a "Hard" vo
# A "Soft" classifier averages the confidence of each of the models. If the average co

from sklearn.ensemble import VotingClassifier
voting_clf = VotingClassifier(estimators = [('lr',lr),('knn',knn),('rf',rf),('gnb',gnb
```

In [39]:
```
cv = cross_val_score(voting_clf,X_train_scaled,y_train,cv=5)
print(cv)
print(cv.mean())
```

```
[0.83146067 0.81460674 0.83146067 0.80337079 0.85875706]
0.8279311877102774
```

In [40]:
```
voting_clf.fit(X_train_scaled,y_train)
y_hat_base_vc = voting_clf.predict(X_test_scaled)
basic_submission = {'PassengerId': test.PassengerId,'Survived': y_hat_base_vc}
base_submission = pd.DataFrame(data=basic_submission)
base_submission.to_csv('base_submission.csv', index=False)
```

Model Tuned Performance

After getting the baselines, we can improve on the individual model results! Mainly using grid search to tune the models. We can also use Randomized Search for the Random Forst and XG boosted model to simplify testing time.

Model	Baseline	Tuned Performance
Naive Bayes	72%	0
Logistic Regression	82%	83%
Decision Tree	77%	0
K Nearest Neighbor	81%	83%
Random Forest	80%	84%
Support Vector Classifier	83%	83%
Xtreme Gradient Boosting	81%	85%

In [41]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [42]:
```
# Simple performance reporting function

def clf_performance(classifier, model_name):
    print(model_name)
```

```
    print('Best Score: ' + str(classifier.best_score_))
    print('Best Parameters: ' + str(classifier.best_params_))
```

In [43]:

```
lr = LogisticRegression()
param_grid = {'max_iter' : [2000],
              'penalty' : ['l1', 'l2'],
              'C' : np.logspace(-4, 4, 20),
              'solver' : ['liblinear']}

clf_lr = GridSearchCV(lr, param_grid = param_grid, cv = 5, verbose = True, n_jobs = -1
best_clf_lr = clf_lr.fit(X_train_scaled,y_train)
clf_performance(best_clf_lr,'Logistic Regression')
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Logistic Regression
Best Score: 0.8268075922046594
Best Parameters: {'C': 1.623776739188721, 'max_iter': 2000, 'penalty': 'l1', 'solve
r': 'liblinear'}
```

In [44]:

```
knn = KNeighborsClassifier()
param_grid = {
    'n_neighbors' : [3,5,7,9],
    'weights' : ['uniform', 'distance'],
    'algorithm' : ['auto','ball_tree','kd_tree'],
    'p' : [1,2]}

clf_knn = GridSearchCV(knn, param_grid = param_grid, cv = 5, verbose = True, n_jobs =
best_clf_knn = clf_knn.fit(X_train_scaled, y_train)
clf_performance(best_clf_knn, 'KNN')
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
KNN
Best Score: 0.8301720307243065
Best Parameters: {'algorithm': 'auto', 'n_neighbors': 7, 'p': 2, 'weights': 'unifor
m'}
```

In [ ]:

```
svc = SVC(probability = True)
param_grid = tuned_parameters = [{'kernel': ['rbf'], 'gamma': [.1,.5,1,2,5,10],
                                  'C': [.1, 1, 10, 100, 1000]},
                                 {'kernel': ['linear'], 'C': [.1, 1, 10, 100, 1000]},
                                 {'kernel': ['poly'], 'degree' : [2,3,4,5], 'C': [.1,

clf_svc = GridSearchCV(svc, param_grid = param_grid, cv = 5, verbose = True, n_jobs =
best_clf_svc = clf_svc.fit(X_train_scaled,y_train)
clf_performance(best_clf_svc,'SVC')
```

```
Fitting 5 folds for each of 55 candidates, totalling 275 fits
```

In [ ]:

```
# The total feature space is so large. Used a randomized search to narrow down the par

"""
rf = RandomForestClassifier(random_state = 1)
param_grid = {'n_estimators': [100, 500, 1000],
                                'bootstrap': [True, Flase],
                                'max_depth': [3,5,10,20,50,75,100,None],
                                'max_features': ['auto','sqrt'],
                                'min_samples_leaf': [1,2,4,10],
                                'min_samples_split': [2,5,10]}

clf_rf_rnd = RandomizedSearchCV(rf, param_distributions = param_grid, n_iter = 100, cv
```

```
best_clf_rf_rnd = clf_rf_rnd.fit(X_train_scaled,y_train)
clf_performance(best_clf_rf_rnd,'Random Forsest')
"""
```

In [ ]:
```
rf = RandomForestClassifier(random_state = 1)
param_grid = {'n_estimators': [400,450,500,550],
              'criterion': ['gini','entropy'],
              'bootstrap': [True],
              'max_depth': [15, 20, 25],
              'max_features':['auto','sqrt', 10],
              'min_samples_leaf': [2,3],
              'min_samples_split': [2,3]}

clf_rf = GridSearchCV(rf, param_grid = param_grid, cv = 5, verbose = True, n_jobs = -1
best_clf_rf = clf_rf.fit(X_train_scaled, y_train)
clf_performance(best_clf_rf,'Random Forest')
```

In [ ]:
```
best_rf = best_clf_rf.best_estimator_.fit(X_train_scaled,y_train)
feat_importances = pd.Series(best_rf.feature_importances_, index=X_train_scaled.column
feat_importances.nlargest(20).plot(kind='barh')
```

In [ ]:
```
"""xgb = XGBClassifier(random_state = 1)

param_grid = {
    'n_estimators': [20, 50, 100, 250, 500, 1000],
    'colsample_bytree': [0.2, 0.5, 0.7, 0.8, 1],
    'max_depth': [2, 5, 10, 15, 20, 25, None],
    'reg_alpha': [0, 0.5, 1],
    'reg_lambda': [1, 1.5, 2],
    'subsample': [0.5, 0.6, 0.7, 0.8, 0.9],
    'learning_rate': [.01, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9],
    'gamma': [0, .01, 0.1, 10, 100],
    'min_child_weight': [0, .01, 0.1, 1, 10, 100],
    'sampling_method': ['uniform', 'gradient_based']
}

#clf_xgb = GridSearchCV(xgb, param_grid = param_grid, cv = 5, verbose = True, n_jobs =
#best_clf_xgb = clf_xgb.fit(X_train_scaled, y_train)
#clf_performance(best_clf_xgb, 'XGB')
clf_xgb_rnd = RandomizedSearchCV(xgb, param_distributions = param_grid, n_iter = 1000,
best_clf_xgb_rnd = clf_xgb_rnd.fit(X_train_scaled,y_train)
clf_performance(best_clf_xgb_rnd,'XGB')
"""
```

In [ ]:
```
xgb = XGBClassifier(random_state = 1)

param_grid = {
    'n_estimators': [450,500,550],
    'colsample_bytree': [0.75, 0.8, 0.85],
    'max_depth': [None],
    'reg_alpha': [1],
    'reg_lambda': [2, 5, 10],
    'subsample': [0.55, 0.6, 0.65],
    'learning_rate': [0.5],
    'gamma': [0.5, 1, 2],
    'sampling_method': ['uniform']
}
```

```
clf_xgb = GridSearchCV(xgb, param_grid = param_grid, cv = 5, verbose = True, n_jobs =
best_clf_xgb = clf_xgb.fit(X_train_scaled,y_train)
clf_performance(best_clf_xgb,'XGB')
```

In [ ]:
```
y_hat_xgb = best_clf_xgb.best_estimator_.predict(X_test_scaled)
xgb_submission = {'PassengerID': test.PassengerId, 'Survived': y_hat_xgb}
submission_xgb = pd.DataFrame(data=xgb_submission)
submission_xgb.to_csv('xgb_submission3.csv',index=False)
```

Model Additional Ensemble Approaches

1) Experimented with a hard voting classifier of three estimators (KNN, SVM, RF) (81.6%)

2) Experimented with a soft voting classifier of three estimators (KNN, SVM, RF) (82.3%)

3) Experimented with a soft voting on all estimators performing better than 80% except xgb (KNN, RF, LR, SVC) (82.9%)

4) Experimented with a soft voting on all estimators including XGB (KNN, SVM, RF, LR, XGB) (83.5%)

In [ ]:
```
best_lr = best_clf_lr.best_estimator_
best_knn = best_clf_knn.best_estimator_
best_svc = best_clf_svc.best_estimator_
best_rf = best_clf_rf.best_estimator_
best_xgb = best_clf_xgb.best_estimator_

voting_clf_hard = VotingClassifier(estimators = [('knn',best_knn),('rf',best_rf),('svc
voting_clf_soft = VotingClassifier(estimators = [('knn',best_knn),('rf',best_rf),('svc
voting_clf_all = VotingClassifier(estimators = [('knn',best_knn),('rf',best_rf),('svc'
voting_clf_xgb = VotingClassifier(estimators = [('knn',best_knn),('rf',best_rf),('svc'

print('voting_clf_hard :',cross_val_score(voting_clf_hard,X_train,y_train,cv=5))
print('voting_clf_hard mean :',cross_val_score(voting_clf_hard,X_train,y_train,cv=5).m

print('voting_clf_soft :',cross_val_score(voting_clf_soft,X_train,y_train,cv=5))
print('voting_clf_soft mean :',cross_val_score(voting_clf_soft,X_train,y_train,cv=5).m

print('voting_clf_all :',cross_val_score(voting_clf_all,X_train,y_train,cv=5))
print('voting_clf_all mean :',cross_val_score(voting_clf_all,X_train,y_train,cv=5).mea

print('voting_clf_xgb :',cross_val_score(voting_clf_xgb,X_train,y_train,cv=5))
print('voting_clf_xgb mean :',cross_val_score(voting_clf_xgb,X_train,y_train,cv=5).mea
```

In [ ]:
```
# In a soft voting classifier you can weight some models more than others. Use a grid
# No new results here

params = {'weights' :[[1,1,1],[1,2,1],[1,1,2],[2,1,1],[2,2,1],[1,2,2],[2,1,2]]}

vote_weight = GridSearchCV(voting_clf_soft, param_grid = params, cv = 5, verbose = Tru
best_clf_weight = vote_weight.fit(X_train_scaled,y_train)
clf_performance(best_clf_weight,'VC Weights')
voting_clf_sub = best_clf_weight.best_estimator_.predict(X_test_scaled)
```

In [ ]:
```
# Make predictions

voting_clf_hard.fit(X_train_scaled, y_train)
voting_clf_soft.fit(X_train_scaled, y_train)
voting_clf_all.fit(X_train_scaled, y_train)
voting_clf_xgb.fit(X_train_scaled, y_train)

best_rf.fit(X_train_scaled, y_train)
y_hat_vc_hard = voting_clf_hard.predict(X_test_scaled)
y_hat_rf = best_rf.predict(X_test_scaled)
y_hat_vc_soft = voting_clf_soft.predict(X_test_scaled)
y_hat_vc_all = voting_clf_all.predict(X_test_scaled)
y_hat_vc_xgb = voting_clf_xgb.predict(X_test_scaled)
```

In [ ]:
```
# Convert output to dataframe for exports

final_data = {'PassengerId': test.PassengerId, 'Survived': y_hat_rf}
submission = pd.DataFrame(data=final_data)

final_data_2 = {'PassengerId': test.PassengerId, 'Survived': y_hat_vc_hard}
submission_2 = pd.DataFrame(data=final_data_2)

final_data_3 = {'PassengerId': test.PassengerId, 'Survived': y_hat_vc_soft}
submission_3 = pd.DataFrame(data=final_data_3)

final_data_4 = {'PassengerId': test.PassengerId, 'Survived': y_hat_vc_all}
submission_4 = pd.DataFrame(data=final_data_4)

final_data_5 = {'PassengerId': test.PassengerId, 'Survived': y_hat_vc_xgb}
submission_5 = pd.DataFrame(data=final_data_5)

final_data_comp = {'PassengerId': test.PassengerId, 'Survived_vc_hard': y_hat_vc_hard,
comparison = pd.DataFrame(data=final_data_comp)
```

In [ ]:
```
# track differences between outputs

comparison['difference_rf_vc_hard'] = comparison.apply(lambda x: 1 if x.Survived_vc_ha
comparison['difference_soft_hard'] = comparison.apply(lambda x: 1 if x.Survived_vc_har
comparison['difference_hard_all'] = comparison.apply(lambda x: 1 if x.Survived_vc_all
```

In [ ]:
```
comparison.difference_hard_all.value_counts()
```

In [ ]:
```
#excel exports

submission.to_csv('submission_rf.csv', index=False)
submission_2.to_csv('submission_vc_hard.csv', index=False)
submission_3.to_csv('submission_vc_soft.csv', index=False)
submission_4.to_csv('submission_vc_all.csv', index=False)
submission_5.to_csv('submission_vc_xgb2.csv', index=False)
```

In [ ]:
```

```