In [13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
train = pd.read_csv("train.csv")
```

In [3]:
```
#Load the two datasets as pandas DataFrames
#train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [22]:
```
train.shape
```

Out[22]:
```
(891, 12)
```

In [23]:
```
train.dtypes
```

Out[23]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [24]:
```
pd.isnull(train).sum()
```

Out[24]:
```
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
```

In [14]:
```
numeric_cols = ['Age', 'Survived']
train[numeric_cols].describe()
```

Out[14]:

	Age	Survived
count	714.000000	891.000000
mean	29.699118	0.383838
std	14.526497	0.486592
min	0.420000	0.000000
25%	20.125000	0.000000
50%	28.000000	0.000000
75%	38.000000	1.000000
max	80.000000	1.000000

In [30]:
```
train[numeric_cols].min()
```

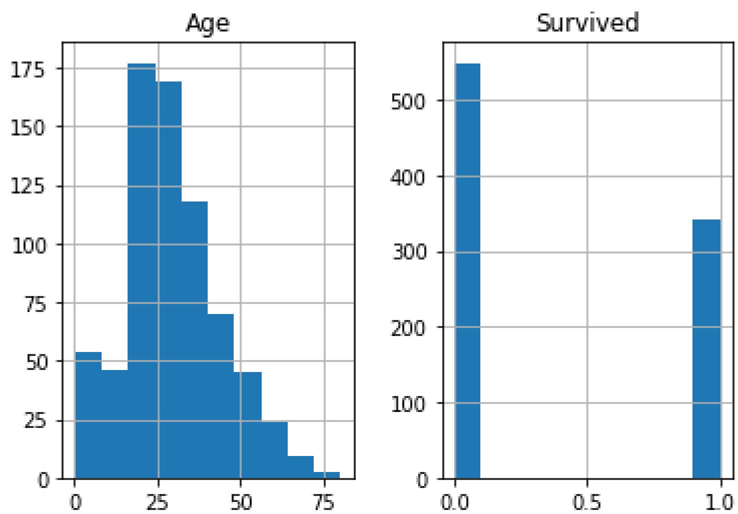
Out[30]:
```
Age         0.42
Survived    0.00
dtype: float64
```

In [31]:
```
train[numeric_cols].mean()
```

Out[31]:
```
Age         29.699118
Survived     0.383838
dtype: float64
```

In [15]:
```
# plot the distribution as histograms
train[numeric_cols].hist()
```

Out[15]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000020D8EB1FE88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000020D8EB6E7C8>]],
      dtype=object)
```



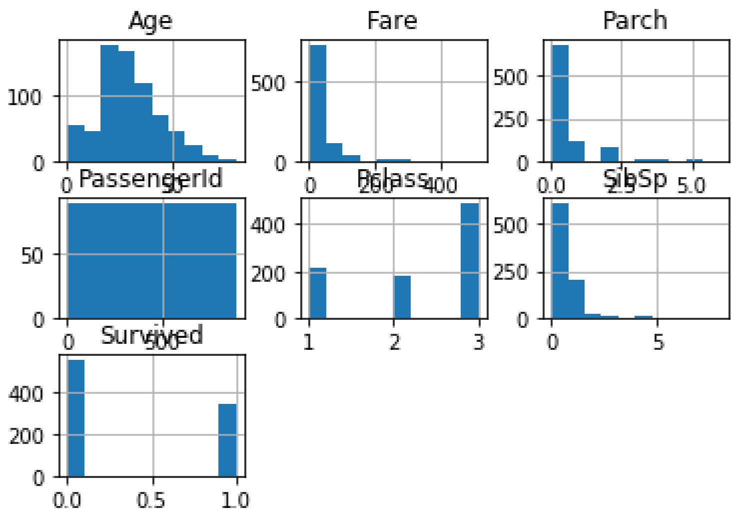
In [3]:
```
train.describe()
```

Out[3]:

	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 [6]:
```
train.hist()
```

Out[6]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002909BFB9608>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002909C27B8C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002909C2B5748>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002909C2EE888>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002909C325988>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002909C35CA48>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002909C394B88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002909C3CDC88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002909C3D9848>]],
      dtype=object)
```



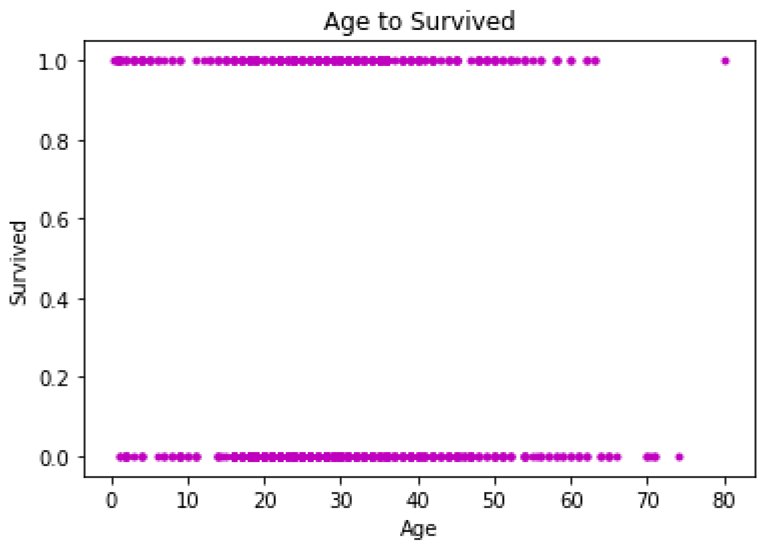
In [16]:
```
#Use correlation coefficients for Age
np.corrcoef(train['Age'], train['Survived'])[0, 1]
```

Out[16]:
```
nan
```

In [17]:
```
#Try scattorplot for Age
train['Age'].describe()
```

Out[17]:
```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

In [27]:
```
#plot out Age vs Survived
plt.title("Age to Survived")
plt.plot(train['Age'], train['Survived'], 'm.')
plt.xlabel("Age")
plt.ylabel("Survived")
plt.show()
```



In [18]:
```
train['Embarked'].head()
```

Out[18]:
```
0    S
1    C
2    S
3    S
4    S
Name: Embarked, dtype: object
```

In [19]:
```
embarked = ['Embarked', 'Survived']
train[embarked].describe()
```

Out[19]:

	Survived
count	891.000000
mean	0.383838
std	0.486592
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

In [20]:
```
s_embarked = train[(train['Embarked'] == "S")]
s_embarked['Embarked']
```

Out[20]:
```
0      S
2      S
3      S
4      S
6      S
      ..
883    S
884    S
886    S
887    S
888    S
Name: Embarked, Length: 644, dtype: object
```

In [21]:
```
q_embarked = train[(train['Embarked'] == "Q")]
q_embarked['Embarked']
```

Out[21]:
```
5      Q
16     Q
22     Q
28     Q
32     Q
      ..
790    Q
825    Q
828    Q
885    Q
890    Q
Name: Embarked, Length: 77, dtype: object
```

In [22]:
```
c_embarked = train[(train['Embarked'] == "C")]
c_embarked['Embarked']
```

Out[22]:
```
1      C
9      C
19     C
26     C
30     C
      ..
866    C
874    C
875    C
879    C
889    C
Name: Embarked, Length: 168, dtype: object
```

In [23]:
```
c_embarked.describe()
```

Out[23]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	168.000000	168.000000	168.000000	130.000000	168.000000	168.000000	168.000000
mean	445.357143	0.553571	1.886905	30.814769	0.386905	0.363095	59.954144
std	259.454201	0.498608	0.944100	15.434860	0.557213	0.660481	83.912994
min	2.000000	0.000000	1.000000	0.420000	0.000000	0.000000	4.012500
25%	235.500000	0.000000	1.000000	21.250000	0.000000	0.000000	13.697950
50%	455.000000	1.000000	1.000000	29.000000	0.000000	0.000000	29.700000
75%	651.000000	1.000000	3.000000	40.000000	1.000000	1.000000	78.500025
max	890.000000	1.000000	3.000000	71.000000	2.000000	3.000000	512.329200

In [24]:
```
train['Embarked'] = pd.to_numeric(train['Embarked'])
#Use correlation coefficients for Embarked
np.corrcoef(s_embarked['Embarked'], train['Survived'])[0, 1]
```

```
During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-24-e559b898ae8b> in <module>
----> 1 train['Embarked'] = pd.to_numeric(train['Embarked'])
      2 #Use correlation coefficients for Embarked
      3 np.corrcoef(s_embarked['Embarked'], train['Survived'])[0, 1]

~\Anaconda3\lib\site-packages\pandas\core\tools\numeric.py in to_numeric(arg,
errors, downcast)
    149             coerce_numeric = errors not in ("ignore", "raise")
    150             values = lib.maybe_convert_numeric(
--> 151                 values, set(), coerce_numeric=coerce_numeric
    152             )
    153

pandas\_libs\lib.pyx in pandas._libs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "S" at position 0
```

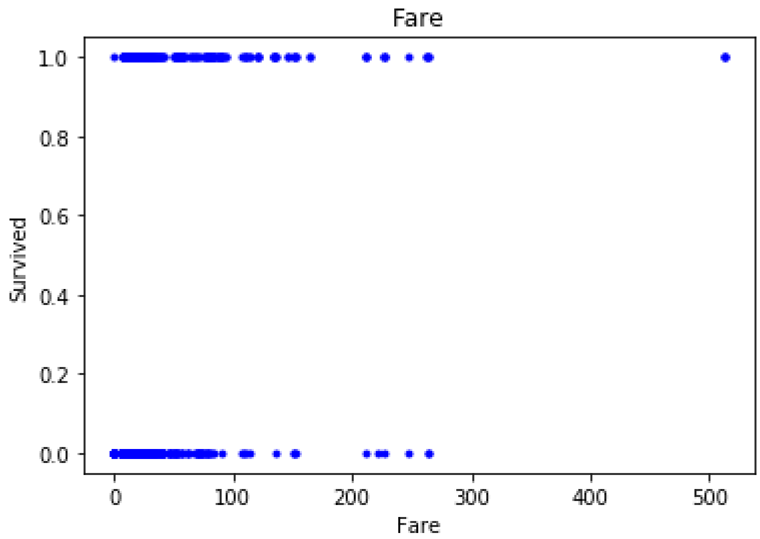
In [39]:
```
#Try scatterplot for Embark
plt.title("Embarked")
plt.plot(train['Embarked'], train['Survived'], 'm.')
plt.xlabel("Embarked")
plt.ylabel("Survived")
plt.show()
```

...

In [41]:
```
np.corrcoef(train['Fare'], train['Survived'])[0, 1]
```

Out[41]:
```
0.2573065223849624
```

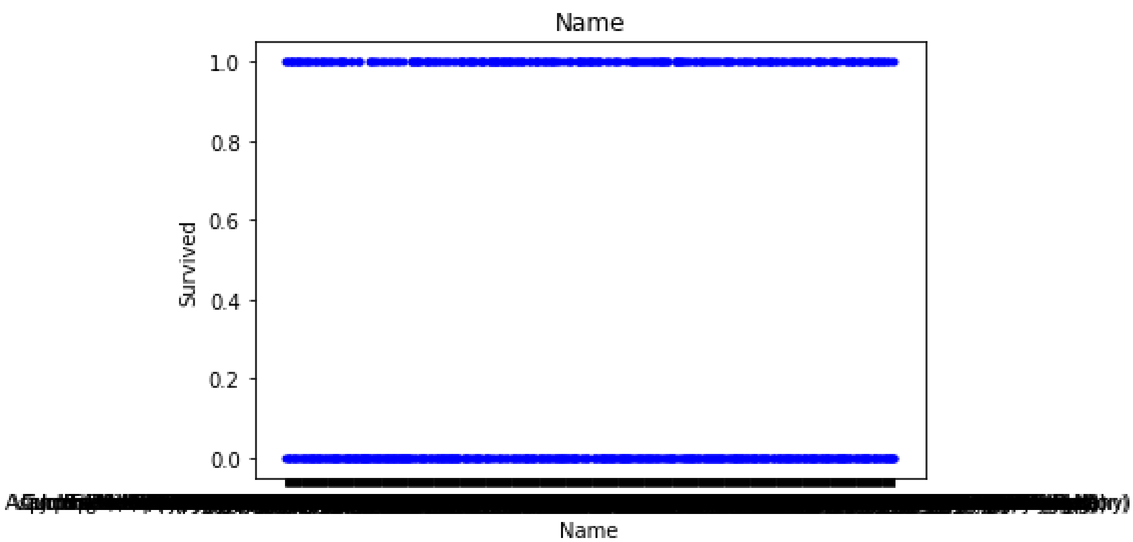
In [42]:
```
plt.title("Fare")
plt.plot(train['Fare'], train['Survived'], 'b.')
plt.xlabel("Fare")
plt.ylabel("Survived")
plt.show()
```



In [11]:
```
np.corrcoef(train['Name'], train['Survived'])[0, 1]
```

...

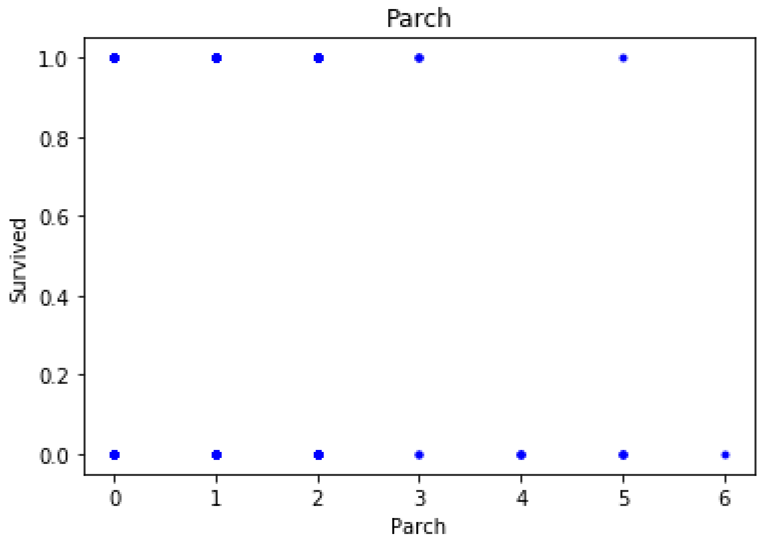
In [43]:
```
plt.title("Name")
plt.plot(train['Name'], train['Survived'], 'b.')
plt.xlabel("Name")
plt.ylabel("Survived")
plt.show()
```



In [12]:
```
np.corrcoef(train['Parch'], train['Survived'])[0, 1]
```

Out[12]: 0.08162940708348373

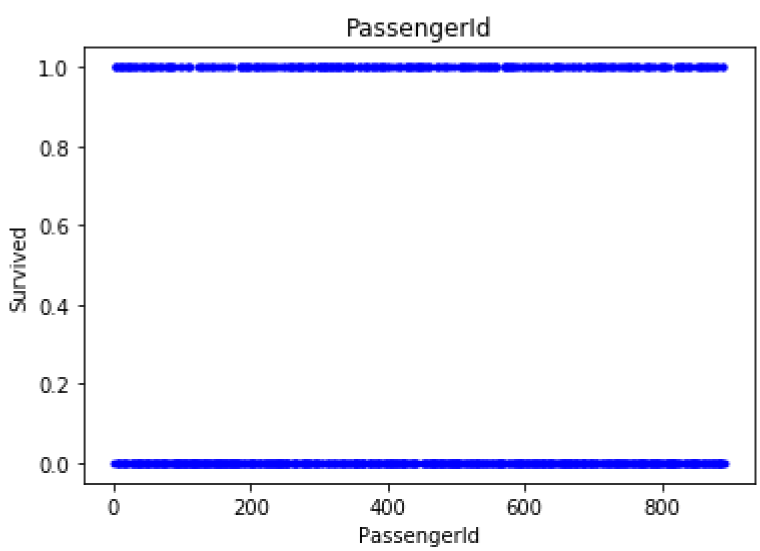
In [44]:
```
plt.title("Parch")
plt.plot(train['Parch'], train['Survived'], 'b.')
plt.xlabel("Parch")
plt.ylabel("Survived")
plt.show()
```



In [13]:
```
np.corrcoef(train['PassengerId'], train['Survived'])[0, 1]
```

Out[13]:
```
-0.005006660767066484 6
```

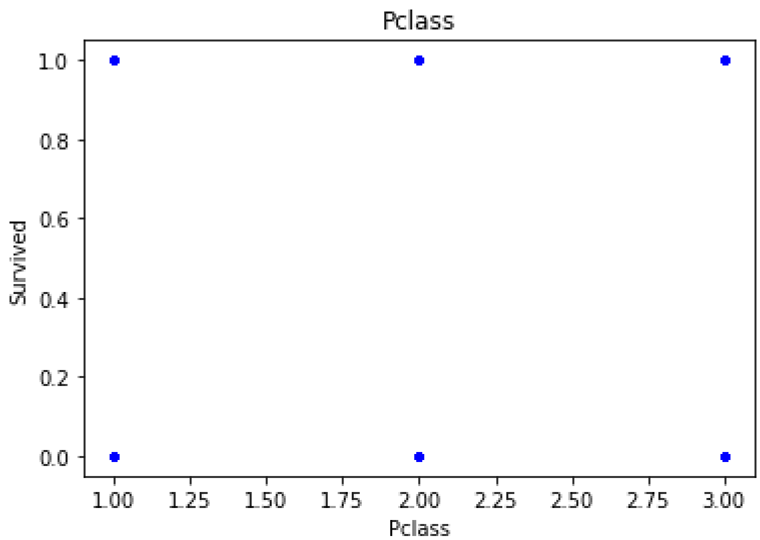
In [45]:
```
plt.title("PassengerId")
plt.plot(train['PassengerId'], train['Survived'], 'b.')
plt.xlabel("PassengerId")
plt.ylabel("Survived")
plt.show()
```



In [76]:
```
#Lack of relevance
train.drop(['PassengerId'], axis=1, inplace=True)
```

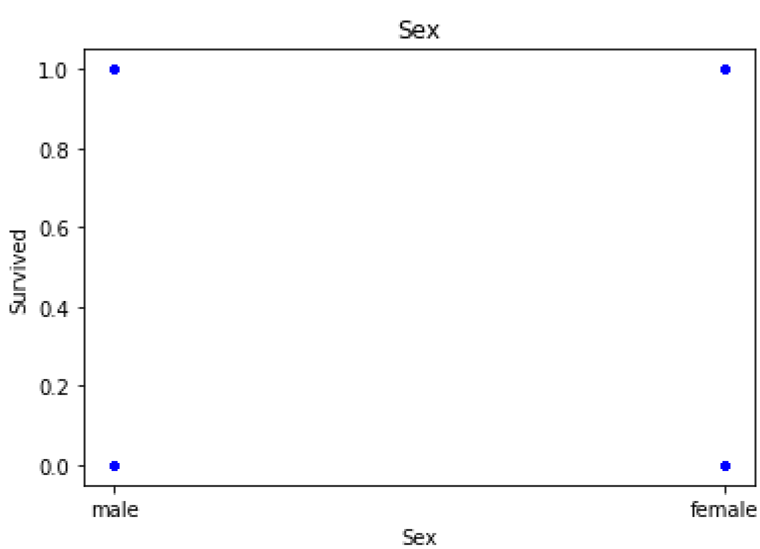
In [14]:
```
np.corrcoef(train['Pclass'], train['Survived'])[0, 1]
```
Out[14]:
```
-0.33848103596101575
```

In [46]:
```
plt.title("Pclass")
plt.plot(train['Pclass'], train['Survived'], 'b.')
plt.xlabel("Pclass")
plt.ylabel("Survived")
plt.show()
```



In [11]:
```
np.corrcoef(train['Sex'], train['Survived'])[0, 1]
```
Out[11]:
```
-0.5433513806577552
```

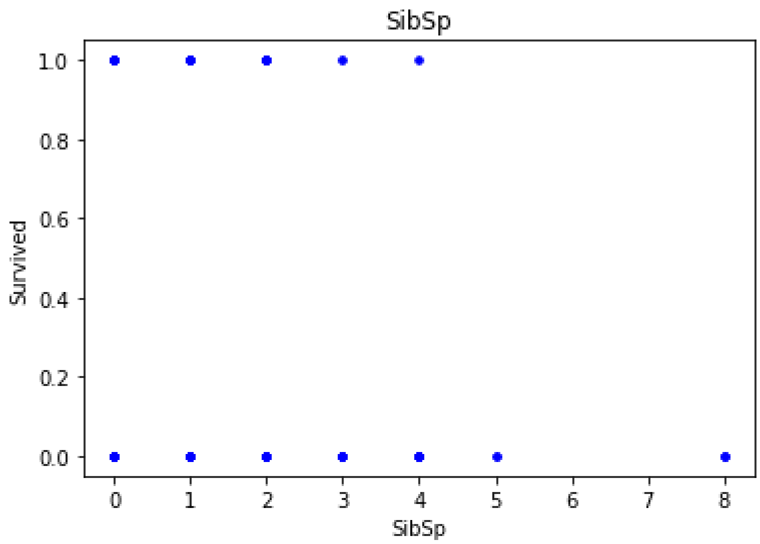
In [47]:
```
plt.title("Sex")
plt.plot(train['Sex'], train['Survived'], 'b.')
plt.xlabel("Sex")
plt.ylabel("Survived")
plt.show()
```



In [16]:
```
np.corrcoef(train['SibSp'], train['Survived'])[0, 1]
```

Out[16]:
```
-0.03532249888573569
```

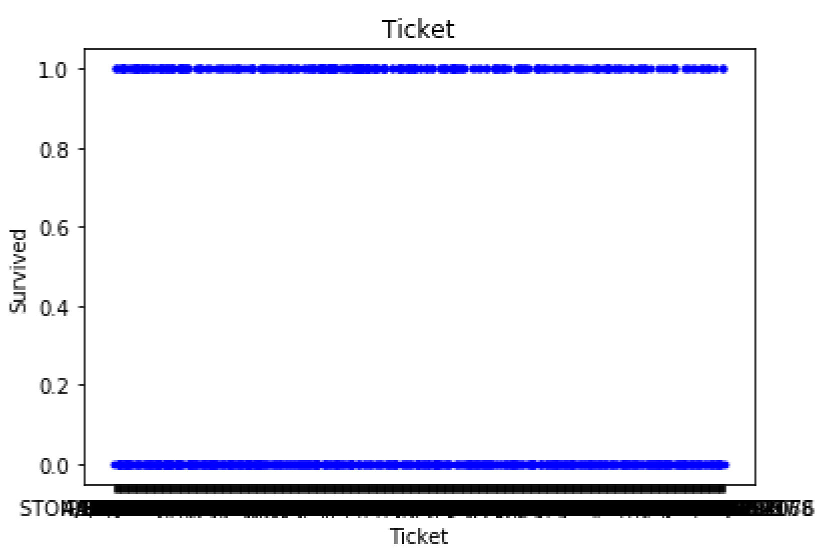
In [48]:
```
plt.title("SibSp")
plt.plot(train['SibSp'], train['Survived'], 'b.')
plt.xlabel("SibSp")
plt.ylabel("Survived")
plt.show()
```



In [23]:
```
np.corrcoef(train['Ticket'], train['Survived'])[0, 1]
```

...

In [49]:
```
plt.title("Ticket")
plt.plot(train['Ticket'], train['Survived'], 'b.')
plt.xlabel("Ticket")
plt.ylabel("Survived")
plt.show()
```



In [25]:
```
#Are there missing values in each column?
print(train.isnull().sum())
print(train.describe())
```

```
dtype: int64
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

In [26]:
```
#There are 891 rows, yet Cabin has omitted 687.
#Such a large omnission is reason enough to discard this feature
train.drop(['Cabin'], axis=1, inplace=True)
train.head()
```

				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

In [27]:
```
#I will input the mean value for 'Age' in all NaN rows

train['Age'] = train['Age'].replace(to_replace = np.nan, value = 29.6)
print(train['Age'])
train['Age'].isnull().sum()
```

```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       ...
886    27.0
887    19.0
888    29.6
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

Out[27]: 0

In [75]:
```
#I will input the mean value for 'Embarked' in all NaN rows
train['Embarked'].fillna("S", inplace = True)
print(train['Embarked'])
train['Embarked'].isnull().sum()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, ke
y, method, tolerance)
   2896             try:
-> 2897                 return self._engine.get_loc(key)
   2898             except KeyError:

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

KeyError: 'Embarked'
```

In [87]:
```
#Let's check for rare titles

train['Title'] = train.object.contains(pat = "Dr.")
train['Title']
```

...

In [29]:
```
#Data Prep

#Let's convert Sex into 0's and 1's
sex_binary = train['Sex'].map({'male': 1, 'female': 0})
train['Sex'] = sex_binary
train['Sex']
```

...

In [30]:
```
#Divide Embarked into 3 dummy columns
embarked_dummy = pd.get_dummies(train['Embarked'])
embarked_dummy
```

Out[30]:

	C	Q	S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
886	0	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	0	1	0

891 rows × 3 columns

In [31]:

```
#Make a dummy Frame with appropriate Column names
embarked_dummy = embarked_dummy.rename(columns={'S': "Embark_S"})
embarked_dummy = embarked_dummy.rename(columns={'Q': "Embark_Q"})
embarked_dummy = embarked_dummy.rename(columns={'C': "Embark_C"})
embarked_dummy
```

Out[31]:

	Embark_C	Embark_Q	Embark_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
886	0	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	0	1	0

891 rows × 3 columns

In [32]:

```
# Replace Embarked with Embark_S, Embark_C, and Embark_Q dummy variable
train['Embark_C'] =  embarked_dummy['Embark_C']
train['Embark_Q'] = embarked_dummy['Embark_Q']
train['Embark_S'] = embarked_dummy['Embark_S']
train.drop(['Embarked'], axis=1, inplace=True)
# Lack of relavance, so dropped
train.drop(['Ticket'], axis=1, inplace=True)
train.drop(['Name'], axis=1, inplace=True)
train.head()
```

Out[32]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embark_C	Embark_Q	Emba
0	1	0	3	1	22.0	1	0	7.2500	0	0	
1	2	1	1	0	38.0	1	0	71.2833	1	0	
2	3	1	3	0	26.0	0	0	7.9250	0	0	
3	4	1	1	0	35.0	1	0	53.1000	0	0	
4	5	0	3	1	35.0	0	0	8.0500	0	0	

In [33]:
```
train.dtypes
```

Out[33]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Sex              int64
Age            float64
SibSp            int64
Parch            int64
Fare           float64
Embark_C         uint8
Embark_Q         uint8
Embark_S         uint8
dtype: object
```

In [35]:
```
np.corrcoef(train['Embark_C'], train['Survived'])[0, 1]
```

Out[35]:
```
0.1682404312182333
```

In [36]:
```
np.corrcoef(train['Embark_Q'], train['Survived'])[0, 1]
```

Out[36]:
```
0.0036503826839721777
```

In [37]:
```
np.corrcoef(train['Embark_S'], train['Survived'])[0, 1]
```

Out[37]:
```
-0.14968272327068632
```

In [104]:
```
#Let's standardize the Data
#Providing scale to the model
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
print(scaler.fit(train))
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [79]:
```
#Create training set and validation set
#Allowing a verification of efficacy
from sklearn.model_selection import train_test_split
train_train, train_test = train_test_split(train, test_size = 0.2)
print(train_train.shape, train_test.shape)
```

```
(712, 10) (179, 10)
```

See which model fits best!

In [39]:
```
#linear regression

# Construct matrix X using np.hstack(), np.ones()
m, n = train_train.shape
X = np.hstack([np.ones([m, 1]), train_train[['Fare',
                                              'Parch',
                                              'Pclass',
                                              'Sex',
                                              'SibSp',
                                              'Embark_C',
                                              'Embark_Q',
                                              'Embark_S']].values])
print(X)
```

```
[[ 1.       7.925   0.      ... 0.      0.      1.     ]
 [ 1.      52.      0.      ... 0.      0.      1.     ]
 [ 1.      12.475   1.      ... 0.      0.      1.     ]
 ...
 [ 1.       6.4958  0.      ... 0.      0.      1.     ]
 [ 1.      15.2458  1.      ... 1.      0.      0.     ]
 [ 1.      38.5     0.      ... 0.      0.      1.     ]]
```

In [40]:
```
# Construct vector y
y = train_train[['Survived']].values
print(y)
```

```
[[0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [0]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]
```

```
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
```

```
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
```

```
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
```

```
 [1]
 [0]
 [0]
 [1]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [1]
 [0]
 [1]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [0]
```

```
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[1]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
```

```
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[1]
[1]
[1]
[0]
```

```
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
```

```
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
```

```
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
[0]
[1]
[0]
[1]
[1]
[1]
[0]
[1]
[0]
```

```
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
```

```
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
```

```
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]]
```

In [41]:

```
# Apply the normal equation to find theta
num = X.T.dot(X)
theta = np.linalg.inv(num).dot(X.T.dot(y))
print(theta)
```

```
[[ 4.        ]
 [ 0.00872086]
 [ 0.06271524]
 [ 0.23522227]
 [-0.80183856]
 [-0.05467896]
 [-4.        ]
 [-6.        ]
 [-4.        ]]
```

In [42]:
```
# 2. define a function that returns the error of a given instance.
def get_squared_error(train_train, name, theta):
    # Extract x and y from data
    x = train_train.loc[name, ['Fare',
                               'Parch',
                               'Pclass',
                               'Sex',
                               'SibSp',
                               'Embark_C',
                               'Embark_Q',
                               'Embark_S']].values
    y = train_train.loc[name, ['Survived']].values
    # calculate prediction
    prediction = theta[0] + theta[1]*x[0] + theta[2]*x[1]
    # calculate the squared error
    squared_error = (prediction - y)**2
    return squared_error
```

In [43]:
```
# 3. calculate all errors
all_errors = [get_squared_error(train_train,name,theta) for name in train_train.i

# 4. calculate the average.
mse = np.mean(all_errors)
print("MSE:", mse)
print("Root mean squared error (RMSE):", np.sqrt(mse))
```

```
MSE: 15.74129679330938
Root mean squared error (RMSE): 3.9675303141008738
```

In [44]:
```
train_train
```

Out[44]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embark_C	Embark_Q	Em
816	817	0	3	0	23.0	0	0	7.9250	0	0	
383	384	1	1	0	35.0	1	0	52.0000	0	0	
751	752	1	3	1	6.0	0	1	12.4750	0	0	
727	728	1	3	0	29.6	0	0	7.7375	0	1	
606	607	0	3	1	30.0	0	0	7.8958	0	0	
...	...	...	...	...	...	...	...	...	...	...	
638	639	0	3	0	41.0	0	5	39.6875	0	0	
355	356	0	3	1	28.0	0	0	9.5000	0	0	
371	372	0	3	1	18.0	1	0	6.4958	0	0	
709	710	1	3	1	29.6	1	1	15.2458	1	0	
462	463	0	1	1	47.0	0	0	38.5000	0	0	

712 rows × 11 columns

In [46]:
```
# Build a linear regression model using LinearRegression from sklearn.linear_mode
from sklearn.linear_model import LinearRegression
titanic_lr = LinearRegression()
titanic_lr.fit(train_train[['Fare',
                              'Parch',
                              'Pclass',
                              'Sex',
                              'SibSp',
                              'Embark_C',
                              'Embark_Q',
                              'Embark_S'
                          ]],
               train_train['Survived'])
```

Out[46]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [109]:
```
# Show the parameter values
print(titanic_lr.intercept_)
print(titanic_lr.coef_)
```

```
1.0542066060100563
[ 4.79911097e-04 -4.72048814e-04 -1.42876701e-01 -5.04718397e-01
 -3.78204496e-02  3.79008722e-02 -6.84655801e-03 -3.10543142e-02]
```

In [113]:
```
train.head()
```

Out[113]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embark_C	Embark_Q	Embark_S
0	0	3	1	22.0	1	0	7.2500	0	0	1
1	1	1	0	38.0	1	0	71.2833	1	0	0
2	1	3	0	26.0	0	0	7.9250	0	0	1
3	1	1	0	35.0	1	0	53.1000	0	0	1
4	0	3	1	35.0	0	0	8.0500	0	0	1

In [115]:
```
#cross validation
from sklearn.model_selection import cross_val_score
input_cols = train.columns[1:9]
print(cross_val_score(titanic_lr, train_test[input_cols], train_test['Survived'],
                      cv=3))
```

```
[0.37949375 0.39507143 0.28610441]
```

In [116]:
```
#confusion matrix
from sklearn.metrics import confusion_matrix
test_predictions = titanic_lr.predict(train_test[input_cols])
print(test_predictions)
matrix = confusion_matrix(train_test['Survived'], test_predictions)
print(matrix)
```

```
[-2.02451182  5.4416135  -3.5320743  -3.76026952 -2.89613999 -7.78345348
 -2.78613119 -1.71479126 -2.95634543 -1.79296181 -0.63269251 -2.6016729
 -2.18951321 -4.94033604  0.81284084 -3.92009071  0.12471816 -5.93780572
 -2.01895929 -1.34255836 -2.90592978 -2.86650524 -6.69084639 -2.91082657
  0.81407621 -6.51211945 -3.36859883 -2.36146686 -1.79264552 -3.61747696
 -4.10693921 -1.82096671 -1.13940868  2.74539687  1.42606947 -3.03665578
 -1.76146415 -2.96874181  2.24162263 -3.51191355 -2.34682611 -2.87471836
 -4.06296289 -2.6907708   0.33128784  0.83140981  1.15598107 -2.62441342
 -3.02190037  2.64628687 -3.06849042  0.94524695 -4.57271117 -2.91129862
 -3.18837481 -3.79939984 -2.46799727 -1.81333182 -1.79390933 -0.13927177
 -2.47418034 -4.11186793 -6.79787286 -4.5322112   0.40716512  1.42690634
 -1.7667815  -1.92325515 -2.9501509   0.09780058  0.79219257  1.22203478
 -1.61692013 -2.9765242  -3.61705764 -3.32592648  0.22413321 -2.43818011
 -1.6857426  -5.0919975   0.94797716 -4.92192608 -4.79759904 -1.42919533
 -2.74998294  1.95959111 -4.63664473  5.13009055 -1.37564006 -3.11132601
 -5.08007346 -1.53153455 -2.86887405  0.18874665 -2.861773    1.86723732
 -2.13113995 -2.52321127 -3.08118195 -2.88735244 -3.17493589 -2.24954209
 -2.59243634 -0.49746579 -2.51130459 -1.16120209 -6.79787286 -1.33503239
 -8.75495571 -3.40468226 -3.88143688 -2.79753004 -1.4976027  -5.88218275
```

In [112]:
```
# precision - recall
from sklearn.metrics import precision_score, recall_score
precision = precision_score(train_test['Survived'], test_predictions) #correctly
recall = recall_score(train_test['Survived'], test_predictions) #predictions are
print(precision, recall)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-112-096b93c4b5c7> in <module>
      1 # precision - recall
      2 from sklearn.metrics import precision_score, recall_score
----> 3 precision = precision_score(train_test['Survived'], test_predictions)
#correctly identified?
      4 recall = recall_score(train_test['Survived'], test_predictions) #pred
ictions are correct?
      5 print(precision, recall)

~\Anaconda3\lib\site-packages\sklearn\metrics\classification.py in precision_
score(y_true, y_pred, labels, pos_label, average, sample_weight)
   1567                                                  average=average,
   1568                                                  warn_for=('precisio
n',),
-> 1569                                                  sample_weight=sample
_weight)
   1570     return p
   1571

~\Anaconda3\lib\site-packages\sklearn\metrics\classification.py in precision_
recall_fscore_support(y_true, y_pred, beta, labels, pos_label, average, warn_
for, sample_weight)
   1413         raise ValueError("beta should be >0 in the F-beta score")
   1414     labels = _check_set_wise_labels(y_true, y_pred, average, labels,
-> 1415                                     pos_label)
   1416
   1417     # Calculate tp_sum, pred_sum, true_sum ###

~\Anaconda3\lib\site-packages\sklearn\metrics\classification.py in _check_set
_wise_labels(y_true, y_pred, average, labels, pos_label)
   1237                              str(average_options))
   1238
-> 1239     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
   1240     present_labels = unique_labels(y_true, y_pred)
   1241     if average == 'binary':

~\Anaconda3\lib\site-packages\sklearn\metrics\classification.py in _check_tar
gets(y_true, y_pred)
     79     if len(y_type) > 1:
     80         raise ValueError("Classification metrics can't handle a mix o
f {0} "
---> 81                          "and {1} targets".format(type_true, type_pre
d))
     82
     83     # We can't have more than one value on y_type => The set is no mo
re needed

ValueError: Classification metrics can't handle a mix of binary and continuou
```

```
s targets
```

In [108]:

```
#polynomial regression

# Apply the normal equation to find the degree 2 polynomial fit of X and y
ones = np.ones([m, 1]) # The first column of X
X10 = X ** 10 # The eleventh column of X
X_matrix = np.hstack([np.ones([m, 1]), train_train[['Fare',
                                                     'Parch',
                                                     'Pclass',
                                                     'Sex',
                                                     'SibSp',
                                                     'Embark_C',
                                                     'Embark_Q',
                                                     'Embark_S']].values])
# concatenate the columns horizontally

theta = np.linalg.inv(X_matrix.T.dot(X_matrix)).dot(X_matrix.T).dot(y)
print(theta)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-108-8a52c0b29cea> in <module>
     14 # concatenate the columns horizontally
     15 
---> 16 theta = np.linalg.inv(X_matrix.T.dot(X_matrix)).dot(X_matrix.T).dot(y)
     17 print(theta)

ValueError: shapes (9,712) and (891,) not aligned: 712 (dim 1) != 891 (dim 0)
```

In [52]:

```
# Use sklearn
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
poly_features.fit(X)
X_poly = poly_features.transform(X)
model_titanic = LinearRegression()
model_titanic.fit(X_poly, y)
print(model_titanic.coef_, model_titanic.intercept_)
```

```
[[ 4.07018166e+09 -1.24539511e-02 -2.49472572e-02 -4.01437249e-02
  -1.20363042e+00  9.82847007e-02 -2.58041955e-02  1.54040024e-01
  -1.28235829e-01  3.48740481e-12 -1.24539510e-02 -2.49472572e-02
  -4.01437248e-02 -1.20363042e+00  9.82847007e-02 -2.58041955e-02
   1.54040024e-01 -1.28235829e-01  9.99643949e-06  9.41468127e-04
  -2.60439345e-03  1.16514015e-02  1.25450298e-03  1.60398879e-02
  -4.52629266e-02  1.67690876e-02  9.19911644e-02 -4.53100503e-02
  -3.33085984e-01 -5.65801459e-01 -1.97441701e-01 -2.91937684e-01
   4.64432128e-01 -7.70750456e-02  8.44498608e-01 -4.26394202e-02
   1.08315011e-01  4.03249995e-01 -5.51708730e-01 -1.20363042e+00
   4.91380159e-01 -7.54644708e-01 -1.48175663e+00  1.03277092e+00
   1.12715079e-01  6.21433817e-02  7.15467241e-02 -3.54054051e-02
  -2.58041955e-02  0.00000000e+00  0.00000000e+00  1.54040024e-01
   0.00000000e+00 -1.28235829e-01]] [-4.07018165e+09]
```

In [53]:
```
# Calculate the MSE
from sklearn.metrics import mean_squared_error
predictions = model_titanic.predict(X_poly)
mse = mean_squared_error(y, predictions)
print("MSE:", mse)
print("Root mean squared error (RMSE):", np.sqrt(mse))
```

```
MSE: 0.7415904797787476
Root mean squared error (RMSE): 0.8611564781029912
```

In [63]:
```
# 2. cross validation
from sklearn.model_selection import cross_val_score
input_cols = train.columns[1:]
print(cross_val_score(model_titanic, train_test[input_cols], train_train['Surviv
                      cv=3))
```

...

In [65]:
```
train_test[input_cols].describe
#train_test.head()
```

Out[65]:
```
<bound method NDFrame.describe of      Survived  Pclass  Sex   Age  SibSp  Parc
h     Fare  Embark_C  Embark_Q  \
148         0       2    1  36.5      0      2   26.0000        0         0
448         1       3    0   5.0      2      1   19.2583        1         0
443         1       2    0  28.0      0      0   13.0000        0         0
95          0       3    1  29.6      0      0    8.0500        0         0
681         1       1    1  27.0      0      0   76.7292        1         0
..        ...     ...  ...   ...    ...    ...       ...      ...       ...
586         0       2    1  47.0      0      0   15.0000        0         0
360         0       3    1  40.0      1      4   27.9000        0         0
618         1       2    0   4.0      2      1   39.0000        0         0
319         1       1    0  40.0      1      1  134.5000        1         0
99          0       2    1  34.0      1      0   26.0000        0         0

     Embark_S
148         1
448         0
443         1
95          1
681         0
..        ...
586         1
360         1
618         1
319         0
99          1

[179 rows x 10 columns]>
```

In [153]:

```
#confusion matrix
from sklearn.metrics import confusion_matrix
test_predictions = titanic_lr.predict(train_test[input_cols])
print(test_predictions)
matrix = confusion_matrix(train_test['Survived'], test_predictions)
print(matrix)
```

```
[-1.11026118 -0.23289124 -2.07439876 -3.00230868 -2.83148225 -4.8817978
 -7.21811593 -0.87652872 -2.83745776 -5.90721262 -1.99538976 -2.51231546
 -1.17767407 -4.14898492 -2.87429727 -2.43366461 -2.80886396 -2.05122685
  0.9870776   0.1637875  -3.78291843 -1.36901109 -0.95915247  0.03298658
 -8.14065079 -5.1640593  -3.0276898  -1.5347021   0.33108966 -1.90347403
  0.44397571  0.50600417 -3.23762723 -3.62060386 -6.34907714 -2.65522851
 -4.17884806 -0.44363479 -1.79713555 -4.36939809 -3.79853289 -4.64359829
 -1.17767407 -4.08264924 -1.89925433 -4.36560127 -2.91590858 -2.78034865
 -4.0082467  -5.21733327 -4.24895492 -1.04816902 -4.81282762 -5.94051886
 -5.21851007 -3.16892849 -2.07116344 -2.32799347 -1.71958514 -3.45683725
 -5.14372211 -2.34500061 -2.99516199 -1.75434359 -2.60987622 -3.16846712
 -3.31896881 -1.08521713  1.16531818 -3.3782791  -3.48106635 -2.03888378
 -1.94066328 -3.06646415 -3.50593393 -3.0276898  -3.00273175 -1.50107693
 -2.98998774 -1.8747219  -1.07088351  0.25017962 -3.16846712 -3.00785537
 -2.85803769 -7.51598185 -3.01527692 -0.65569422 -1.87574571 -4.59267331
 -4.0914558  -1.49757035 -1.76547058 -3.73483349 -3.43472559 -2.91504156
 -2.59977303 -4.02150319 -3.5550204  -3.35232923 -1.62410738 -3.0276898
 -7.92584779  0.09828293 -1.58651496 -3.0276898  -4.32608115 -1.32388444
 -2.51133226 -4.16357263 -1.76547058 -4.27363024 -2.37047148 -2.98037274
```

In [12]:

```
# precision - recall
from sklearn.metrics import precision_score, recall_score
precision = precision_score(train_test['Survived'], test_predictions) #correctly
recall = recall_score(train_test['Survived'], test_predictions) #predictions are
print(precision, recall)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-096b93c4b5c7> in <module>
      1 # precision - recall
      2 from sklearn.metrics import precision_score, recall_score
----> 3 precision = precision_score(train_test['Survived'], test_predictions) #
correctly identified?
      4 recall = recall_score(train_test['Survived'], test_predictions) #predic
tions are correct?
      5 print(precision, recall)

NameError: name 'train_test' is not defined
```

In [78]:
```
train
```

Out[78]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embark_C	Embark_Q	Embark_S
0	0	3	1	22.0	1	0	7.2500	0	0	1
1	1	1	0	38.0	1	0	71.2833	1	0	0
2	1	3	0	26.0	0	0	7.9250	0	0	1
3	1	1	0	35.0	1	0	53.1000	0	0	1
4	0	3	1	35.0	0	0	8.0500	0	0	1
...	...	...	...	...	...	...	...	...	...	...
886	0	2	1	27.0	0	0	13.0000	0	0	1
887	1	1	0	19.0	0	0	30.0000	0	0	1
888	0	3	0	29.6	1	2	23.4500	0	0	1
889	1	1	1	26.0	0	0	30.0000	1	0	0
890	0	3	1	32.0	0	0	7.7500	0	1	0

891 rows × 10 columns

In [95]:
```
#logistic regression

# Build the logistic regression model
from sklearn.linear_model import LogisticRegression
Titanic_LogR = LogisticRegression(solver='liblinear')
Titanic_LogR.fit(train_train.iloc[:, 1:9], train_train['Survived'])
```

Out[95]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [96]:
```
# 1. Find the prediction accuracy on test set
from sklearn.metrics import accuracy_score
prediction = Titanic_LogR.predict(train_test.iloc[:, 1:9])
accuracy = accuracy_score(train_test['Survived'], prediction)
print(accuracy)
```

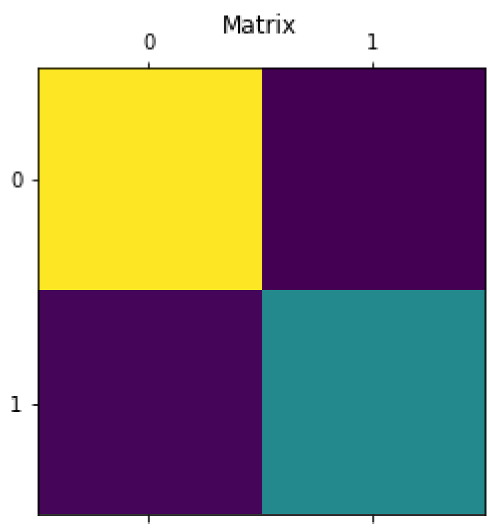
```
0.7932960893854749
```

In [97]:
```
# 2. cross validation
from sklearn.model_selection import cross_val_score
input_cols = train.columns[1:9]
print(cross_val_score(Titanic_LogR, train_test[input_cols], train_test['Survived
                      cv=3))
```

```
[0.76666667 0.78333333 0.76271186]
```

In [100]:
```
# 3. confusion matrix
from sklearn.metrics import confusion_matrix
test_predictions = Titanic_LogR.predict(train_test[input_cols])
print(test_predictions)
matrix = confusion_matrix(train_test['Survived'], test_predictions)
plt.matshow(matrix)
plt.title("Matrix")
print(matrix)
```

```
[0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0
 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0 0 0
 0 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 1
 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 0]
[[90 18]
 [19 52]]
```



In [101]:
```
# precision - recall
from sklearn.metrics import precision_score, recall_score
precision = precision_score(train_test['Survived'], test_predictions) #correctly
recall = recall_score(train_test['Survived'], test_predictions) #predictions are
print(precision, recall)
```

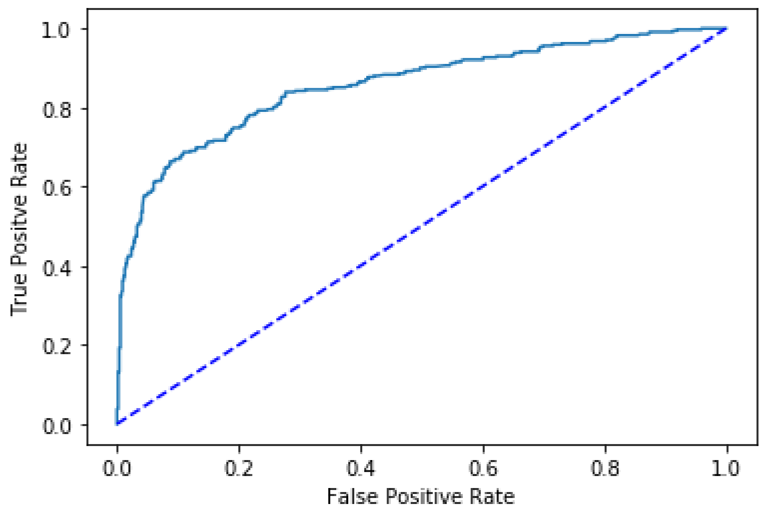
```
0.7428571428571429 0.7323943661971831
```

In [102]:
```
from sklearn.metrics import roc_curve
probs = model_train.predict_proba(train.iloc[:, 1:9])
fpr, tpr, thresholds = roc_curve(train['Survived'], probs[:, 1])

plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positve Rate')

# worse case: tpr increases along with fpr
plt.plot([0, 1], [0, 1], 'b--')
```

Out[102]:
```
[<matplotlib.lines.Line2D at 0x20d8f1c2c48>]
```



In [103]:
```
from sklearn.metrics import roc_auc_score

roc_auc_score(train['Survived'], probs[:, 1])
```

Out[103]:
```
0.8558543444220752
```

The Logistic model is quite good!

In [ ]:
```
```

In [105]:
```
#k-nearest neighbors method
X = train[['Fare',
                'Parch',
                'Pclass',
                'Sex',
                'SibSp',
                'Embark_C',
                'Embark_Q',
                'Embark_S']].values
y = train['Survived'].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20)

X_train = scaler.transform(train_train)
X_test = scaler.transform(train_test)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

#Create 1 nearest neighbor model
classifier_1=KNeighborsClassifier(n_neighbors=1)
classifier_1.fit(X_train, y_train)
y_pred=classifier_1.predict(X_test)
accuracy=accuracy_score(y_test, y_pred)

print("1-nearest neighbor accuracy")
print(accuracy)

#Create 15 nearest neighbor model
classifier_15=KNeighborsClassifier(n_neighbors=15)
classifier_15.fit(X_train, y_train)
y_pred=classifier_15.predict(X_test)
accuracy=accuracy_score(y_test, y_pred)

print("15-nearest neighbor accuracy")
print(accuracy)

#Create 50 nearest neighbor model
classifier_50=KNeighborsClassifier(n_neighbors=50)
classifier_50.fit(X_train, y_train)
y_pred=classifier_50.predict(X_test)
accuracy=accuracy_score(y_test, y_pred)

print("50-nearest neighbor accuracy")
print(accuracy)
```

```
1-nearest neighbor accuracy
0.45251396648044695
15-nearest neighbor accuracy
0.6033519553072626
50-nearest neighbor accuracy
0.6312849162011173
```

In [106]:
```
#confusion matrix
from sklearn.metrics import confusion_matrix

matrix = confusion_matrix(y_test, y_pred)
print(matrix)
```

```
[[111   0]
 [ 66   2]]
```

In [107]:
```
#precision and recall
from sklearn.metrics import precision_score, recall_score
precision = precision_score(y_test, y_pred) #correctly identified?
recall = recall_score(y_test, y_pred) #predictions are correct?
print(precision, recall)
```

```
1.0 0.029411764705882353
```

I believe the Logistic model holds promise on the training data provided and if I had to I would send kaggle Titanic_LogR!

In regard to which variables played a part in better survival, being female, child and embark Q all seem to be factors that increased survivability!

So, Sex, Parch and Embark_Q were significant variables!