In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

In [2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
tested_output = pd.read_csv('gender_submission.csv')
```

Data description:

In [3]:
```
train.columns
```

Out[3]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [4]:
```
test.columns
```

Out[4]:
```
Index(['PassengerId', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [5]:
```
tested_output.columns
```

Out[5]:
```
Index(['PassengerId', 'Survived'], dtype='object')
```

In [6]:
```
train.head()
```

Out[6]:

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

Filling Null Values:

In [7]:
```
train.isnull().sum()
```

Out[7]:
```
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 [8]:
```
test.isnull().sum()
```

Out[8]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

As we can see, train dataset has null values in Age, Cabin and Embarked, whereas test has the same in Age, Fare and Cabin.

So, to fill the null values of the dataset, for age and fare, we will average the values, for cabin and embarked we will use the mode for getting the average.

In [9]:
```
train['Age'].fillna(train['Age'].mean(), inplace=True)
test['Age'].fillna(test['Age'].mean(), inplace=True)
```

In [10]:
```
test['Fare'].fillna(test['Fare'].mean(), inplace=True)
```

In [11]:
```
print('Train cabin values and counts:')
print(train['Cabin'].value_counts())
print()
print('Test cabin values and counts:')
test['Cabin'].value_counts()
```

```
Train cabin values and counts:
B96 B98        4
C23 C25 C27    4
G6             4
D              3
F2             3
              ..
A23            1
B79            1
D15            1
D11            1
C99            1
Name: Cabin, Length: 147, dtype: int64

Test cabin values and counts:
```

Out[11]:
```
B57 B59 B63 B66    3
C101               2
C89                2
C6                 2
C78                2
                  ..
E46                1
G6                 1
B69                1
E52                1
D38                1
Name: Cabin, Length: 76, dtype: int64
```

In [12]:
```
def fillNullValInCabin(df, value1, value2):
    toFill = np.array([])
    value_counts = df['Cabin'].value_counts()

    for value, count in value_counts.items():
        if count == value1 or count == value2:
            toFill = np.append(toFill, value)

    random_index = np.random.randint(0, len(toFill))
    df['Cabin'].fillna(toFill[random_index], inplace=True)
```

In [13]:
```
fillNullValInCabin(train, 4, 3)
```

In [14]:
```
fillNullValInCabin(test, 2, 3)
```

In [15]:
```
train['Embarked'].value_counts() # a clear winner, so gonna fill s(72% ir
```

Out[15]:
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [16]:
```
train['Embarked'].fillna('S', inplace=True)
```

Encoding data:

In [17]:
```
train.head()
```

Out[17]:

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

In [18]:
```
test.head()
```

Out[18]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	C C C
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	C C C
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	C C C
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	C C C
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	C C C

In [19]:
```
# since name isn't a factor which can help in learning of the model, as r
# and which wouldn't, hence name is removed from the dataset
```

In [20]:
```
train = train.drop('Name', axis =1)
test = test.drop('Name', axis =1)
```

OneHotEncoding columns Sex and Embarked

train['Embarked'].unique()

In [21]:
```
test['Embarked'].unique()
```

Out[21]:
```
array(['Q', 'S', 'C'], dtype=object)
```

In [22]:
```
train_encoded = pd.concat([train, pd.get_dummies(train['Embarked'], prefi
train_encoded.drop(['Embarked','Sex'], axis =1, inplace =True)
```

In [23]:
```
test_encoded = pd.concat([test, pd.get_dummies(test['Embarked'], prefix=
test_encoded.drop(['Embarked','Sex'], axis =1, inplace =True)
```

In [24]:
```
train_encoded.head(2)
```

Out[24]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	Fare	Cabin	Embar
0	1	0	3	22.0	1	0	A/5 21171	7.2500	B96 B98	
1	2	1	1	38.0	1	0	PC 17599	71.2833	C85	

In [25]:
```
test_encoded.head(2)
```

Out[25]:

	PassengerId	Pclass	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked_C	En
0	892	3	34.5	0	0	330911	7.8292	C23 C25 C27	0	
1	893	3	47.0	1	0	363272	7.0000	C23 C25 C27	0	

In [26]:
```
encoder = LabelEncoder()
train_encoded['cabin_encoded'] =encoder.fit_transform(train_encoded['Cabi
```

In [27]:
```
train_encoded.drop('Cabin', axis=1, inplace=True)
```

In [28]:
```
test_encoded['cabin_encoded'] =encoder.fit_transform(test_encoded['Cabin'
test_encoded.drop('Cabin', axis=1, inplace=True)
```

In [29]:
```
# since passenger id is unique, we don't use it to learn in a model becau
```

In [30]:
```
train_encoded.drop('PassengerId',axis=1,inplace = True)
```

In [31]:
```
test_passengerID = test_encoded['PassengerId']
```

In [32]:
```
test_encoded.drop('PassengerId',axis=1,inplace = True)
```

In [33]:
```
len(train_encoded['Ticket'].unique())/train_encoded.shape[0]
```

Out[33]: 0.7643097643097643

In [34]:
```
len(test_encoded['Ticket'].unique())/test_encoded.shape[0]
```

Out[34]: 0.868421052631579

In [35]:
```
#since most of the tickets are unique, we drop them too, following same r
```

In [36]:
```
train_encoded.drop('Ticket',axis=1,inplace = True)
test_encoded.drop('Ticket',axis=1,inplace = True)
```

In [37]:
```
train_encoded.shape
```

Out[37]: (891, 12)

In [38]:
```
test_encoded.shape
```

Out[38]: (418, 11)

In [39]:
```
X = train_encoded.drop('Survived', axis=1)
y = train_encoded['Survived']
```

In [40]:
```
y.sum()/y.shape # 38 : 62 -> survive: not survived ratio
```

Out[40]: array([0.38383838])

Scaling:

In [41]:
```
X.head()
```

Out[41]:

	Pclass	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S	Sex
0	3	22.0	1	0	7.2500	0	0	1	
1	1	38.0	1	0	71.2833	1	0	0	
2	3	26.0	0	0	7.9250	0	0	1	
3	1	35.0	1	0	53.1000	0	0	1	
4	3	35.0	0	0	8.0500	0	0	1	

In [42]:
```
y.head()
```

Out[42]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [43]:
```
scaler = StandardScaler()
```

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
```

In [45]:
```
scaler.fit(X_train)
```

Out[45]:
```
StandardScaler()
```

In [46]:
```
X_train_scaled = scaler.transform(X_train)
```

In [47]:
```
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression Model:

In [48]:
```
logreg = LogisticRegression()
```

In [49]:
```
logreg.fit(X_train_scaled,y_train)
```

Out[49]:
```
LogisticRegression()
```

In [50]:
```
y_pred = logreg.predict(X_test_scaled)
```

In [51]:
```
accuracy_score(y_pred, y_test)
```

Out[51]:
```
0.8444444444444444
```

Testing the real test file given

In [52]:
```
test_encoded_scaled = scaler.transform(test_encoded)
```

In [53]:
```
y_logistic_predicted = logreg.predict(test_encoded_scaled)
```

In [54]:
```
tested_output_value = np.array(tested_output['Survived'])
```

In [55]:
```
y_logistic_predicted.shape == tested_output_value.shape
```

Out[55]:
```
True
```

In [56]:
```
accuracy_score(y_logistic_predicted, tested_output_value)
```

Out[56]:
```
0.937799043062201
```

Getting a 94% accurate prediction with the logistic regression model.

SVM Model:

In [57]:
```
x_train, x_test, yo_train, yo_test = train_test_split(X, y, test_size=0.1
```

In [58]:
```
scaler.fit(x_train)
```

Out[58]:
```
StandardScaler()
```

In [59]:
```
x_train_scaled = scaler.transform(x_train)
```

In [60]:
```
x_test_scaled = scaler.transform(x_test)
```

In [61]:
```
model2 = SVC()
```

In [62]:
```
model2.fit(x_train_scaled, yo_train)
```

Out[62]:
```
SVC()
```

In [63]:
```
yo_pred = model2.predict(x_test_scaled)
```

In [64]:
```
accuracy_score(yo_pred,yo_test)
```

Out[64]:
```
0.8111111111111111
```

Testing the real test file given

In [65]:
```
Test_encoded_scaled = scaler.transform(test_encoded)
```

In [66]:
```
y_SVM_predicted = model2.predict(Test_encoded_scaled)
```

In [67]:
```
accuracy_score(y_SVM_predicted, tested_output_value)
```

Out[67]:
```
0.930622009569378
```

Getting a 93% accurate prediction using SVM model.