Titanic_Project

Problem Statement

The Titanic Problem is based on the sinking of the 'Unsinkable' ship Titanic in early 1912. It gives you information about multiple people like their ages, sexes, sibling counts, embarkment points, and whether or not they survived the disaster. Based on these features, you have to predict if an arbitrary passenger on Titanic would survive the sinking or not.

In [28]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [29]:
```
titanic_data=pd.read_csv('Titanic Project.csv')
titanic_data.head()
```

Out[29]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	I
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [30]:
```
#show the last five rows
titanic_data.tail()
```

Out[30]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	

In [3]:
```
# number of rows and Columns
titanic_data.shape
```

Out[3]:
```
(891, 12)
```

In [4]:
```
# getting some informations about the data
titanic_data.info()
```

```
<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
```

In [5]:
```
# check the number of missing values in each column
titanic_data.isnull().sum()
```

Out[5]:
```
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
```

Handling the Missing values

In [6]:
```
# drop the "Cabin" column from the dataframe
titanic_data = titanic_data.drop(columns='Cabin', axis=1)
```

In [7]:
```
# replacing the missing values in "Age" column with mean value
titanic_data['Age'].fillna(titanic_data['Age'].mean(), inplace=True)
```

In [8]:
```
# finding the mode value of "Embarked" column
print(titanic_data['Embarked'].mode())
```

```
0    S
Name: Embarked, dtype: object
```

In [9]:
```
print(titanic_data['Embarked'].mode()[0])
```

```
S
```

In [10]:
```
# replacing the missing values in "Embarked" column with mode value
titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0], inplace=True)
```

In [11]:
```
# check the number of missing values in each column
titanic_data.isnull().sum()
```

Out[11]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

found no missing values

Data Analysis

In [12]:
```
# getting some statistical measures about the data
titanic_data.describe()
```

Out[12]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	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	13.002015	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	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

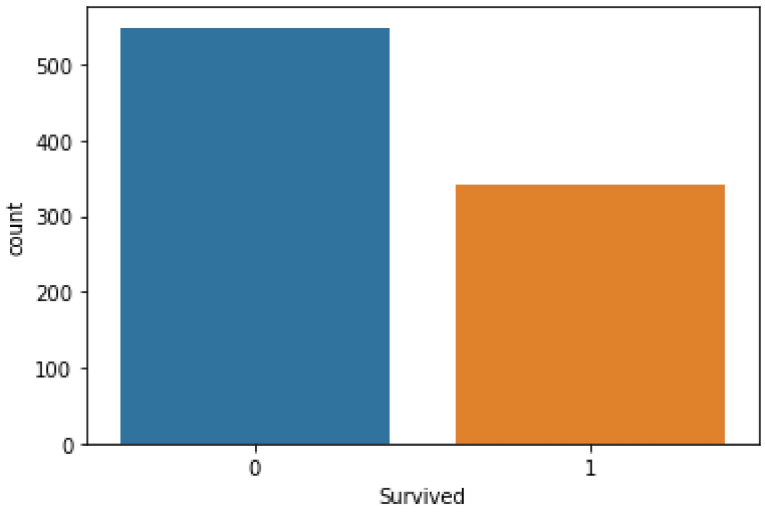
In [13]:
```
# finding the number of people survived and not survived
titanic_data['Survived'].value_counts()
```

Out[13]:
```
0    549
1    342
Name: Survived, dtype: int64
```

Data Visualization

In [14]:
```
# making a count plot for "Survived" column
sns.countplot('Survived', data=titanic_data)
```

Out[14]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```

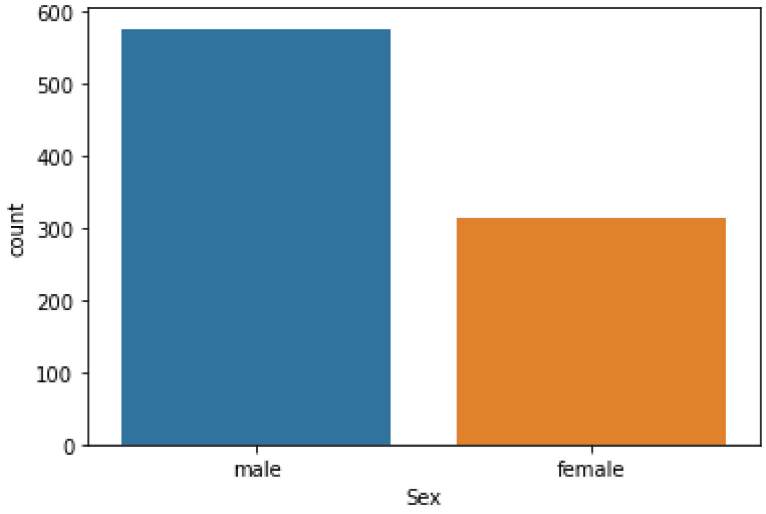


In [15]:
```
titanic_data['Sex'].value_counts()
```

Out[15]:
```
male      577
female    314
Name: Sex, dtype: int64
```

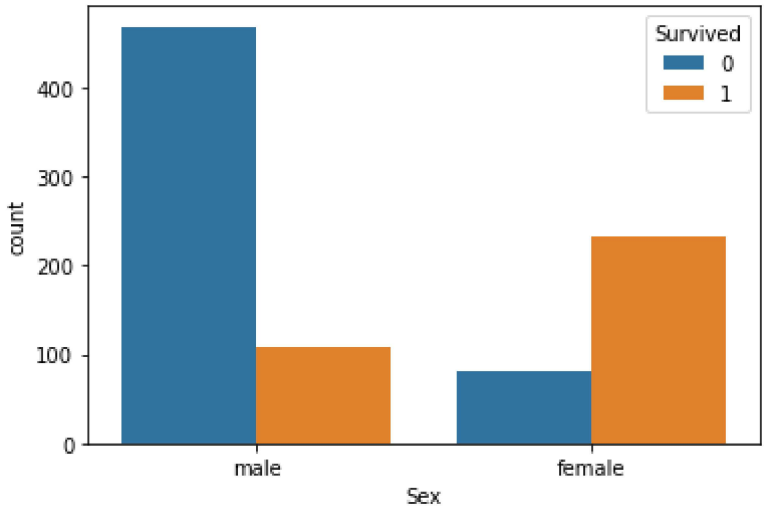
In [16]:
```
# making a count plot for "Sex" column
sns.countplot('Sex', data=titanic_data)
```

Out[16]:
```
<AxesSubplot:xlabel='Sex', ylabel='count'>
```



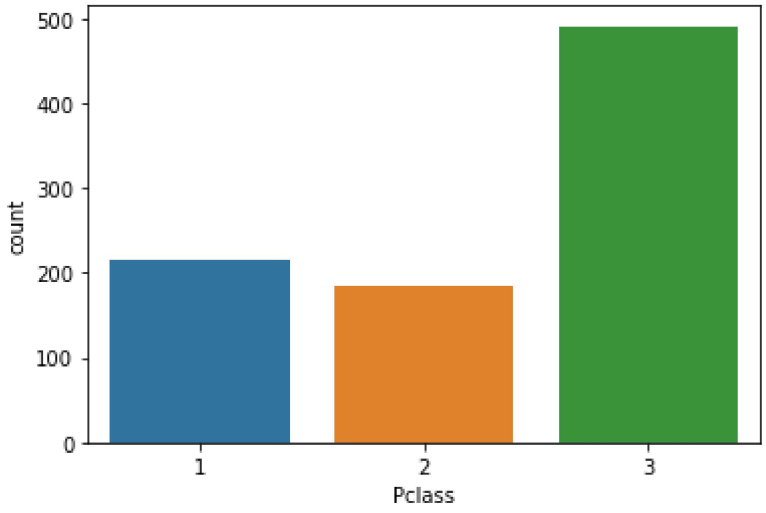
In [17]:
```
# number of survivors Gender wise
sns.countplot('Sex', hue='Survived', data=titanic_data)
```

Out[17]:
```
<AxesSubplot:xlabel='Sex', ylabel='count'>
```



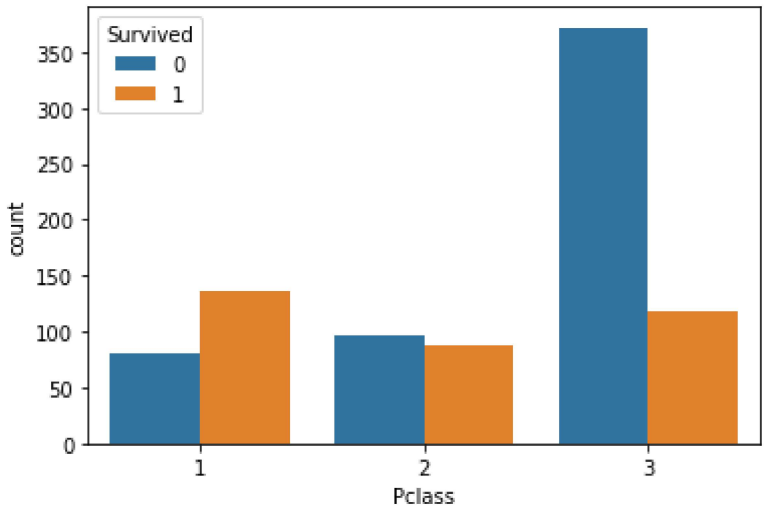
In [18]:
```
# making a count plot for "Pclass" column
sns.countplot('Pclass', data=titanic_data)
```

Out[18]:
```
<AxesSubplot:xlabel='Pclass', ylabel='count'>
```



In [19]:
```
sns.countplot('Pclass', hue='Survived', data=titanic_data)
```

Out[19]:
```
<AxesSubplot:xlabel='Pclass', ylabel='count'>
```



Correlation

In [55]:
```
plt.figure(figsize=(15,12))
sns.heatmap(titanic_data.corr(),cbar=True,square=True,fmt='.3f',annot=True,annot_kws={
```

Out[55]:
```
<AxesSubplot:>
```

Encoding the Categorical Columns

In [20]:
```
titanic_data['Sex'].value_counts()
```

Out[20]:
```
male      577
female    314
Name: Sex, dtype: int64
```

In [21]:
```
titanic_data['Embarked'].value_counts()
```

Out[21]:
```
S    646
C    168
Q     77
Name: Embarked, dtype: int64
```

In [22]:
```
# converting categorical Columns

titanic_data.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'S':0,'C':1,'Q':2}}, ir
```

In [23]:
```
titanic_data.head()
```

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	1
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	0
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	0

Separating features & Target

In [24]:
```
X = titanic_data.drop(columns = ['PassengerId','Name','Ticket','Survived'],axis=1)
Y = titanic_data['Survived']
```

In [25]:
```
print(X.shape)
print(Y.shape)
```

```
(891, 7)
(891,)
```

Splitting the data into training data & Test data

In [26]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state=2
```

In [27]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(891, 7) (712, 7) (179, 7)
```

Model Training

Logistic Regression

In [45]:
```
lr = LogisticRegression()

# training the Logistic Regression model with training data
lr.fit(X_train, Y_train)
```

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

Model Evaluation

Accuracy Score

In [46]:
```
# accuracy on training data
X_train_prediction = lr.predict(X_train)

print(X_train_prediction)
```

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

In [33]:
```
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8075842696629213
```

In [47]:
```
# accuracy on test data
X_test_prediction = lr.predict(X_test)

print(X_test_prediction)
```

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

In [36]:
```
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.7821229050279329
```

Other Machine Learning Models

In [40]:
```
#Create a fuction within many machine learning Models
def models(x_train,y_train):

    #using Logistic Regression Algorithm to the Training set
    from sklearn.linear_model import LogisticRegression
    lr=LogisticRegression()
    lr.fit(x_train,y_train)

    #using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor algo
    from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
    knn.fit(x_train,y_train)

    #using SVC method of svm class use Support Vector Machine Algorithm
    from sklearn.svm import SVC
    svc=SVC(kernel='linear',random_state=0)
    svc.fit(x_train,y_train)

    #using GaussinNB method of navie_bayes class to use Naive Bayes Algorithm
    from sklearn.naive_bayes import GaussianNB
    gnb=GaussianNB()
    gnb.fit(x_train,y_train)

    #using DecisionTreeClassifier of tree class to use Decision Tree Classifier algori
    from sklearn.tree import DecisionTreeClassifier
    dtc=DecisionTreeClassifier()
    dtc.fit(x_train,y_train)

    #using RandomForestClassifier method of ensemble class to use Random Forest Classi
    from sklearn.ensemble import RandomForestClassifier
    rfc=RandomForestClassifier(n_estimators=11,criterion='entropy',random_state=0)
    rfc.fit(x_train,y_train)

    #print model accuracy on the training data.
    print('[0] Logistic Regression Training Accuracy : ',lr.score(X_train,Y_train))
    print('[1] K Nearest Neighbor Training Accuracy : ',knn.score(X_train,Y_train))
    print('[2] Support Vector Machine(Linear Classifier) Training Accuracy : ',svc.sc
    print('[3] Gussian Naive Bayes Training Accuracy : ',gnb.score(X_train,Y_train))
    print('[4] Decision Tree Classifier Training Accuracy : ',dtc.score(X_train,Y_trai
    print('[5] Random Forest Classifier Training Accuracy : ',rfc.score(X_train,Y_trai

    return lr,knn,svc,gnb,dtc,rfc
```

Evaluating Performance on Training Sets

In [41]:
```
#Get and train all of the models
model=models(X_train,Y_train)
```

```
[0] Logistic Regression Training Accuracy :  0.8075842696629213
[1] K Nearest Neighbor Training Accuracy :  0.8075842696629213
[2] Support Vector Machine(Linear Classifier) Training Accuracy :  0.800561797752809
[3] Gussian Naive Bayes Training Accuracy :  0.7935393258426966
[4] Decision Tree Classifier Training Accuracy :  0.9859550561797753
[5] Random Forest Classifier Training Accuracy :  0.9803370786516854
```

Evaluating Performance on Testing Sets

In [42]:
```
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
    cm=confusion_matrix(Y_test, model[i].predict(X_test))
    #extracting TN,FP,FN,TP
    TN,FP,FN,TP = confusion_matrix(Y_test, model[i].predict(X_test)).ravel()
    print(cm)
    print('model[{}] Testing Accuracy="{} !"'.format(i,(TP + TN) / (TP + TN + FN + FP)
    print() #Print a new line
```

```
[[91  9]
 [30 49]]
model[0] Testing Accuracy="0.7821229050279329 !"

[[83 17]
 [40 39]]
model[1] Testing Accuracy="0.6815642458100558 !"

[[88 12]
 [31 48]]
model[2] Testing Accuracy="0.7597765363128491 !"

[[86 14]
 [29 50]]
model[3] Testing Accuracy="0.7597765363128491 !"

[[84 16]
 [21 58]]
model[4] Testing Accuracy="0.7932960893854749 !"

[[88 12]
 [25 54]]
model[5] Testing Accuracy="0.7932960893854749 !"
```

conclusion

In [49]:
```
a=np.array(Y_test)
predicted=np.array(lr.predict(X_test))
titanic_data.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a))
titanic_data.com
```

Out[49]:

	original	predicted
0	1	0
1	0	0
2	1	1
3	0	0
4	1	0
...	...	...
174	0	0
175	0	0
176	0	0
177	0	0
178	0	0

179 rows × 2 columns

In [ ]: