Description: This program predicts if a passenger will survive on the titanic

Titanic Survival Prediction Using Machine Learning¶

In [1]:
```
#import Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

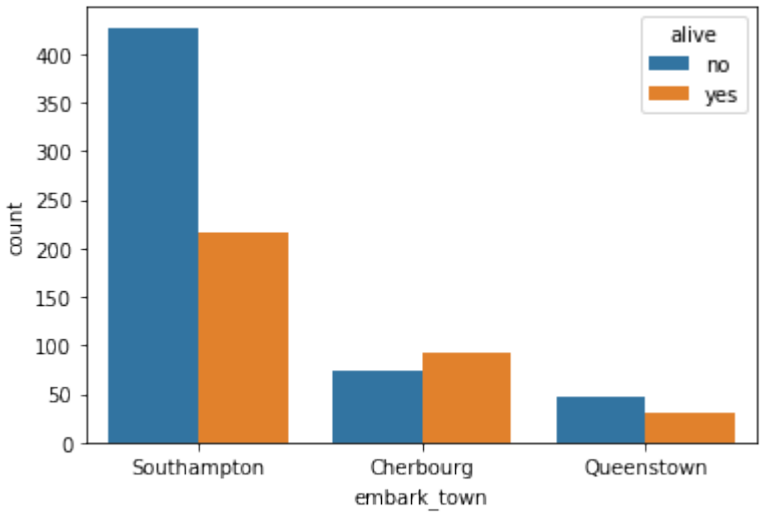
In [4]:
```
#Load the data
titanic = sns.load_dataset('titanic')
titanic.head(10)
```

Out[4]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
5	0	3	male	NaN	0	0	8.4583	Q	Third	man	True
6	0	1	male	54.0	0	0	51.8625	S	First	man	True
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False
8	1	3	female	27.0	0	2	11.1333	S	Third	woman	False
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False

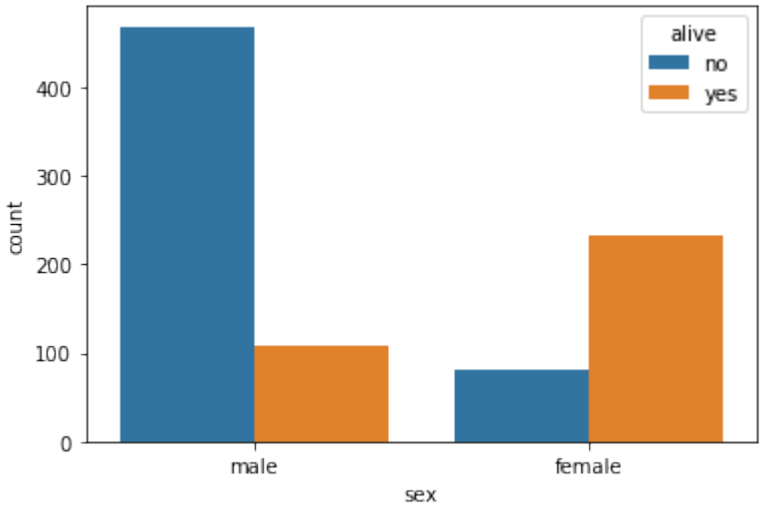
In [11]:
```
sns.countplot(x="embark_town", hue="alive" , data=titanic)
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19244205898>
```



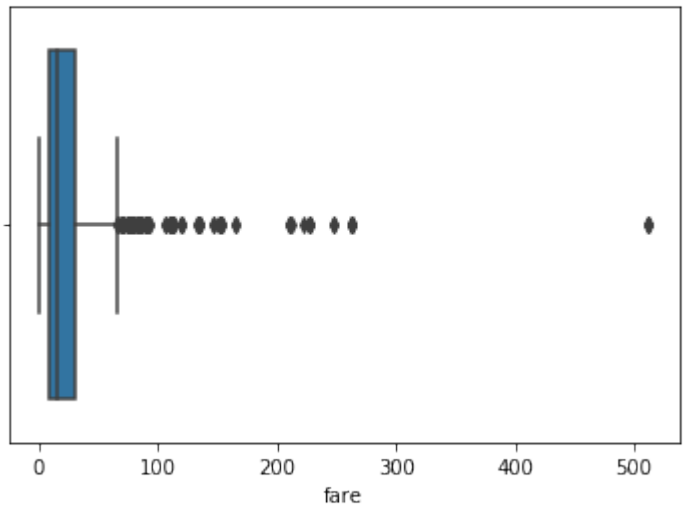
In [7]:
```
sns.countplot(x="sex", hue="alive" , data=titanic)
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x192439531d0>
```



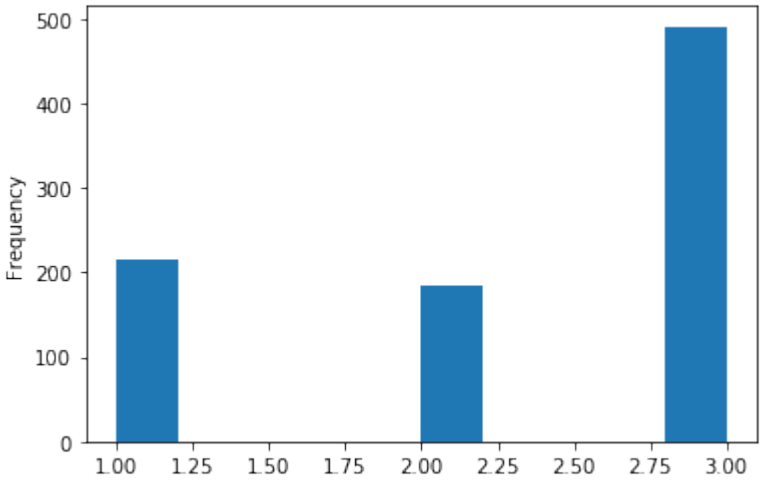
In [8]:
```
sns.boxplot(x = titanic["fare"])
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19243bb4c88>
```



In [9]:
```
titanic["pclass"].plot.hist()
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x19243e3def0>
```



In [37]:
```
titanic.count()
```

Out[37]:
```
survived       891
pclass         891
sex            891
age            714
sibsp          891
parch          891
fare           891
embarked       889
class          891
who            891
adult_male     891
deck           203
embark_town    889
alive          891
alone          891
dtype: int64
```

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

Out[38]:
```
(891, 15)
```

In [39]:
```
titanic.describe()
```

Out[39]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [54]:
```
titanic['embarked'].value_counts()
```

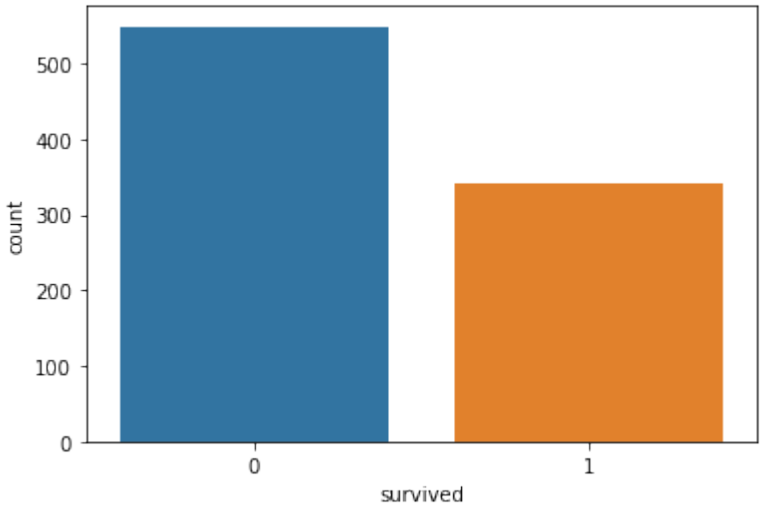
Out[54]:
```
S    644
C    168
Q     77
Name: embarked, dtype: int64
```

In [41]:
```
titanic['survived'].value_counts()
```

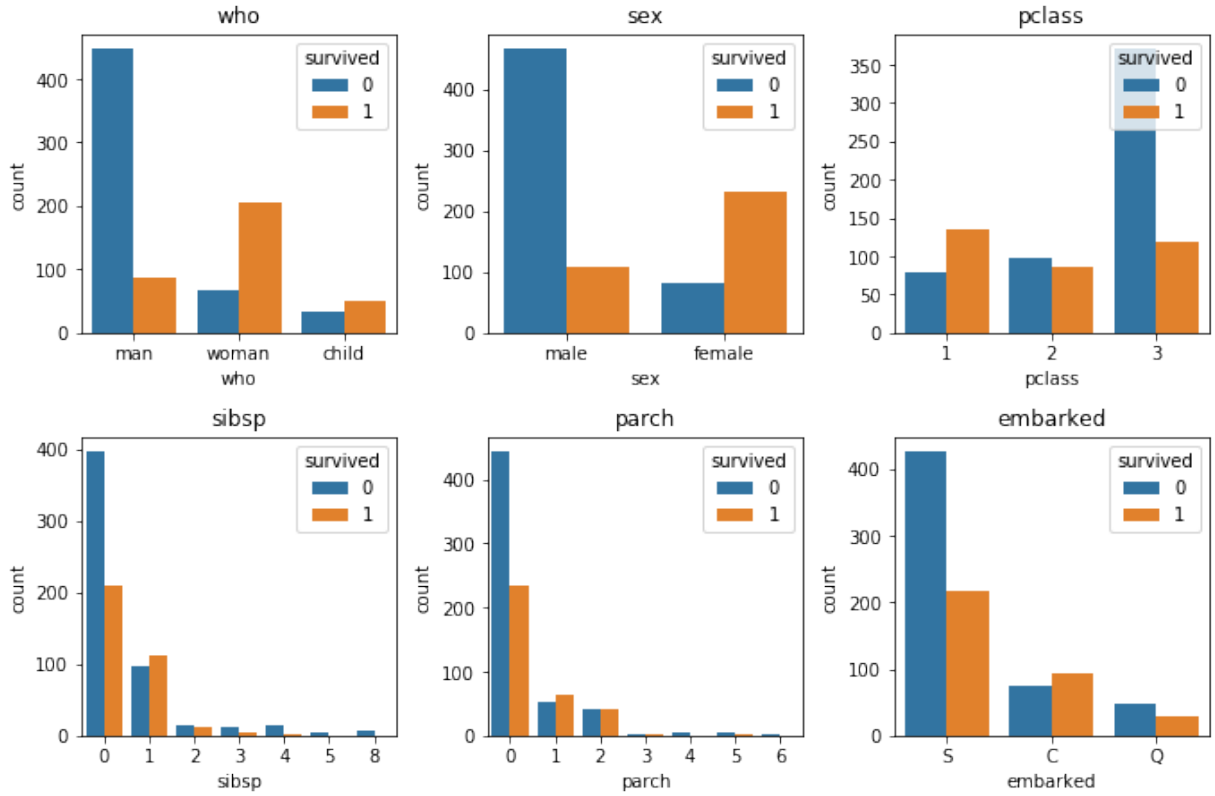
Out[41]:
```
0    549
1    342
Name: survived, dtype: int64
```

In [42]:
```
sns.countplot( titanic['survived'])
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27569d98f98>
```



In [43]:
```
# Visualize the count of survivors for columns 'who', 'sex', 'pclass', '
sibsp', 'parch', and 'embarked'
cols = ['who', 'sex', 'pclass', 'sibsp', 'parch', 'embarked']

n_rows = 2
n_cols = 3

#Number of rows/columns of the subplot grid and the figure size of each
graph
#NOTE: This returns a Figure (fig) and an Axes Object (axs)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(n_cols*3.2,n_rows*3.2))

for r in range(0,n_rows):
    for c in range(0,n_cols):

        i = r*n_cols+ c #index to go through the number of columns

        ax = axs[r][c] #Show where to position each subplot
        sns.countplot(titanic[cols[i]], hue=titanic["survived"], ax=ax)
        ax.set_title(cols[i])
        ax.legend(title="survived", loc='upper right')

plt.tight_layout()   #tight_layout automatically adjusts subplot params
so that the subplot(s) fits in to the figure area
```



In [55]:
```
titanic.groupby('sex')[['survived']].mean()
```

Out[55]:

sex	survived
female	0.742038
male	0.188908

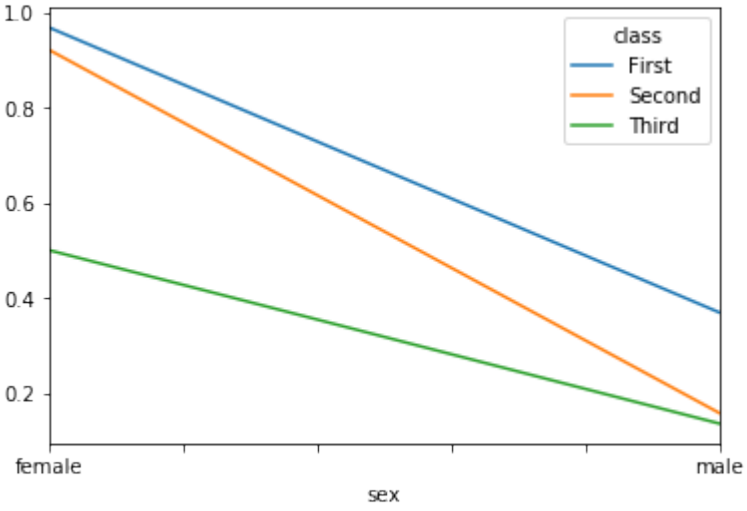
In [56]:
```
titanic.pivot_table('survived', index='sex', columns='class')
```

Out[56]:

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

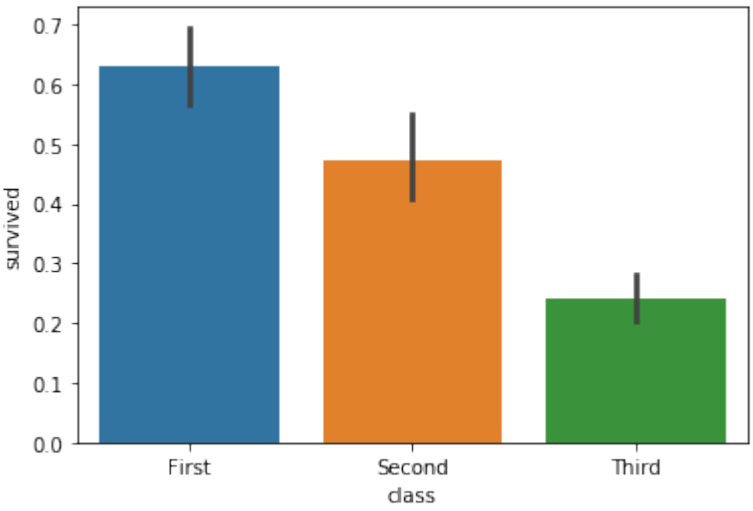
In [12]:
```
titanic.pivot_table('survived', index='sex', columns='class').plot()
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bdfed257b8>
```



In [13]:
```
sns.barplot(x='class', y='survived', data= titanic)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1bdfed25a58>

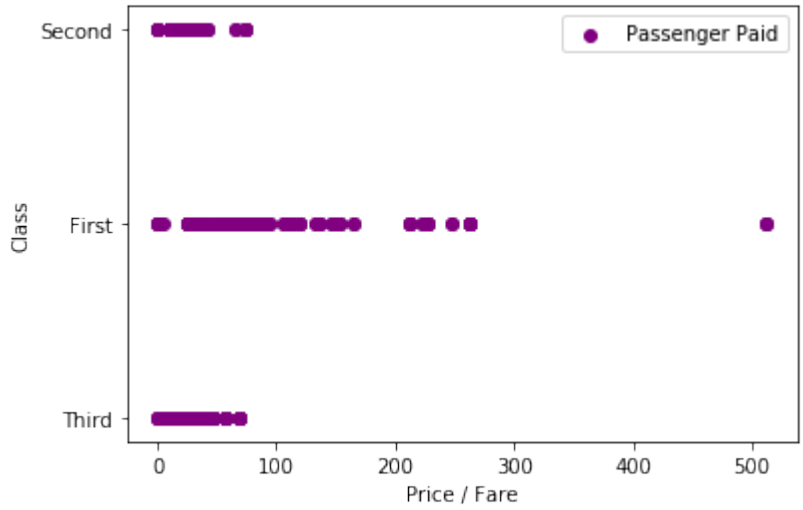


In [45]:
```
age = pd.cut(titanic['age'], [0,18,80])
titanic.pivot_table('survived', ['sex', age], 'class')
```

Out[45]:

	class	First	Second	Third
sex	age			
female	(0, 18]	0.909091	1.000000	0.511628
	(18, 80]	0.972973	0.900000	0.423729
male	(0, 18]	0.800000	0.600000	0.215686
	(18, 80]	0.375000	0.071429	0.133663

In [58]:
```
plt.scatter(titanic['fare'], titanic['class'], color='purple', label='Passenger Paid')
plt.ylabel('Class')
plt.xlabel('Price / Fare')
plt.legend()
plt.show()
```



In [59]:
```
#Count the empty values in each colums
titanic.isna().sum()
```

Out[59]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [60]:
```
titanic.dtypes
```

Out[60]:
```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck           category
embark_town      object
alive            object
alone              bool
dtype: object
```

In [61]:
```
print(titanic['sex'].unique())
print(titanic['embarked'].unique())
```

```
['male' 'female']
['S' 'C' 'Q' nan]
```

In [63]:
```
#Changement de text vers numerique
from sklearn.preprocessing import LabelEncoder
labelencoder_titanic=LabelEncoder()
#titanic['deck']=labelencoder_titanic.fit_transform(titanic['deck'])
titanic['adult_male']=labelencoder_titanic.fit_transform(titanic['adult_
male'])
titanic['alive']=labelencoder_titanic.fit_transform(titanic['alive'])
titanic['alone']=labelencoder_titanic.fit_transform(titanic['alone'])
titanic['sex']=labelencoder_titanic.fit_transform(titanic['sex'])
titanic.head()
```

Out[63]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	1	22.0	1	0	7.2500	S	Third	man	1	NaN
1	1	1	0	38.0	1	0	71.2833	C	First	woman	0	C
2	1	3	0	26.0	0	0	7.9250	S	Third	woman	0	NaN
3	1	1	0	35.0	1	0	53.1000	S	First	woman	0	C
4	0	3	1	35.0	0	0	8.0500	S	Third	man	1	NaN

In [ ]:
```
#Encoding categorical data values (Transforming object data types to int
egers)
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()

#Encode sex column
titanic.iloc[:,2]= labelencoder.fit_transform(titanic.iloc[:,2].values)
print(labelencoder.fit_transform(titanic.iloc[:,2].values))

#Encode embarked
titanic.iloc[:,7]= labelencoder.fit_transform(titanic.iloc[:,7].values)
print(labelencoder.fit_transform(titanic.iloc[:,7].values))

#Print the NEW unique values in the columns
print(titanic['sex'].unique())
print(titanic['embarked'].unique())
```

In [72]:
```
#Print the unique values in the colums
print(titanic['sex'].unique())
```

```
[1 0]
```

In [22]:
```
#Look at the NEW data types
titanic.dtypes
```

Out[22]:
```
survived          int64
pclass            int64
sex               int32
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
```

```
adult_male        int64
deck           category
embark_town      object
alive             int32
alone             int64
dtype: object
```

In [23]:
```
#Split the data into independent 'X' and dependent 'Y' variables
X = titanic.iloc[:, 1:8].values #Notice I started from index  1 to 7, es
sentially removing the first column
Y = titanic.iloc[:, 0].values #Get the target variable
```

In [24]:
```
# Split the dataset into 80% Training set and 20% Testing set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.
2, random_state = 0)
```

In [ ]:
```
# Scale the data to bring all features to the same level of magnitude
# This means the data will be within a specific range for example 0 -100
 or 0 - 1

#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
#Create a function 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
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)

  #Using KNeighborsClassifier Method of neighbors class to use Nearest N
eighbor algorithm
  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 to use Support Vector Machine Algorithm
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
  svc_lin.fit(X_train, Y_train)

  #Using SVC method of svm class to use Kernel SVM Algorithm
  from sklearn.svm import SVC
  svc_rbf = SVC(kernel = 'rbf', random_state = 0)
  svc_rbf.fit(X_train, Y_train)

  #Using GaussianNB method of naïve_bayes class to use Naïve Bayes Algor
ithm
  from sklearn.naive_bayes import GaussianNB
  gauss = GaussianNB()
  gauss.fit(X_train, Y_train)

  #Using DecisionTreeClassifier of tree class to use Decision Tree Algor
```

```
ithm
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
  tree.fit(X_train, Y_train)

  #Using RandomForestClassifier method of ensemble class to use Random F
orest Classification algorithm
  from sklearn.ensemble import RandomForestClassifier
  forest = RandomForestClassifier(n_estimators = 10, criterion = 'entrop
y', random_state = 0)
  forest.fit(X_train, Y_train)

  #print model accuracy on the training data.
  print('[0]Logistic Regression Training Accuracy:', log.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_lin.score(X_train, Y_train))
  print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:',
 svc_rbf.score(X_train, Y_train))
  print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_trai
n, Y_train))
  print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X_t
rain, Y_train))
  print('[6]Random Forest Classifier Training Accuracy:', forest.score(X
_train, Y_train))

  return log, knn, svc_lin, svc_rbf, gauss, tree, forest
```

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

In [ ]:
```
#Show the confusion matrix and accuracy for all of the models on the tes
t data
#Classification accuracy is the ratio of correct predictions to total pr
edictions made.
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
  cm = confusion_matrix(Y_test, model[i].predict(X_test))

  #extracting true_positives, false_positives, true_negatives, false_neg
atives
  TN, FP, FN, TP = confusion_matrix(Y_test, model[i].predict(X_test)).ra
vel()

  print(cm)
  print('Model[{}] Testing Accuracy = "{} !"'.format(i,  (TP + TN) / (TP
 + TN + FN + FP)))
  print()# Print a new line
```

In [ ]:
```
#Get the importance of the features
forest = model[6]
importances = pd.DataFrame({'feature':titanic.iloc[:, 1:8].columns,'impo
rtance':np.round(forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).set_
index('feature')
importances
```

In [ ]:
```
#Visualize the importance
importances.plot.bar()
```

In [ ]:
```
#Print Prediction of Random Forest Classifier model
pred = model[6].predict(X_test)
print(pred)

#Print a space
print()

#Print the actual values
print(Y_test)
```

In [ ]:
```
# Given the data points would I have survived ?
# Most likely I would've been in 3rd class (pclass = 3), Im a male (sex
= 1), age is older than 18 (age = 21), no siblings onboard (sibsp = 0),
#no parents or children (parch =0), fare the minimum price (fare = 0), e
mbarked queens town = (embarked =1)
my_survival = [[3,1,21,0, 0, 0, 1]]

#uncomment to see all of the models predictions
#for i in range(len(model)):
#  pred = model[i].predict(my_survival)
#  print(pred)


#Print Prediction of Random Forest Classifier model
pred = model[6].predict(my_survival)
print(pred)

if pred == 0:
  print('Oh no! You didn’t make it')
else:
  print('Nice! You survived')
```