```python
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Loading the datasets
train_data = pd.read_csv('train (2).csv')
test_data = pd.read_csv('test (2).csv')

# Inspecting the datasets
print("Train Data Head:")
train_data.head()
```

```
Train Data Head:
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                 Name     Sex   Age
SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                              Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                            Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
print("\nTest Data Head:")
test_data.head()
```

```
Test Data Head:

   PassengerId  Pclass                                          Name
Sex  \
0          892       3                              Kelly, Mr. James
male
1          893       3              Wilkes, Mrs. James (Ellen Needs)
female
2          894       2                     Myles, Mr. Thomas Francis
male
3          895       3                              Wirz, Mr. Albert
male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

```
print("\nTrain Data Info:")
print(train_data.info())

print("\nTest Data Info:")
print(test_data.info())
```

```
Train 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
None

Test Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

print("\nTrain Data Description:")
print(train_data.describe())

print("\nTest Data Description:")
print(test_data.describe())


Train Data Description:
       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

Test Data Description:
       PassengerId      Pclass         Age       SibSp       Parch
Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000
417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344
35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429
55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000
0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000
7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000
14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000
31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000
512.329200
```

```python
# Checking for missing values
print("\nMissing values in train data:")
print(train_data.isnull().sum())

print("\nMissing values in test data:")
print(test_data.isnull().sum())
```

```
Missing values in train data:
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

Missing values in test data:
PassengerId      0
Pclass           0
Name             0
```

```
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
# Handling missing values (if any)
# Separate numeric and non-numeric columns
numeric_cols = train_data.select_dtypes(include=[np.number]).columns
non_numeric_cols =
train_data.select_dtypes(exclude=[np.number]).columns

# Fill missing values for numeric columns with mean
train_data[numeric_cols] =
train_data[numeric_cols].fillna(train_data[numeric_cols].mean())
#test_data[numeric_cols] =
test_data[numeric_cols].fillna(test_data[numeric_cols].mean())

# Fill missing 'Age' values with the mean of the column
test_data['Age'].fillna(test_data['Age'].mean(), inplace=True)
test_data['Fare'].fillna(test_data['Fare'].mean(), inplace=True)
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\2103912666.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  test_data['Age'].fillna(test_data['Age'].mean(), inplace=True)
C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\2103912666.py:3:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.
```

```
  test_data['Fare'].fillna(test_data['Fare'].mean(), inplace=True)

# Fill missing values for non-numeric columns with mode
for col in non_numeric_cols:
    train_data[col] = train_data[col].fillna(train_data[col].mode()
[0])
    test_data[col] = test_data[col].fillna(test_data[col].mode()[0])

# Re-checking for missing values after handling
print("\nMissing values in train data after handling:")
train_data.isnull().sum()
```

```
Missing values in train data after handling:

PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

```
print("\nMissing values in test data after handling:")
test_data.isnull().sum()
```

```
Missing values in test data after handling:

PassengerId    0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

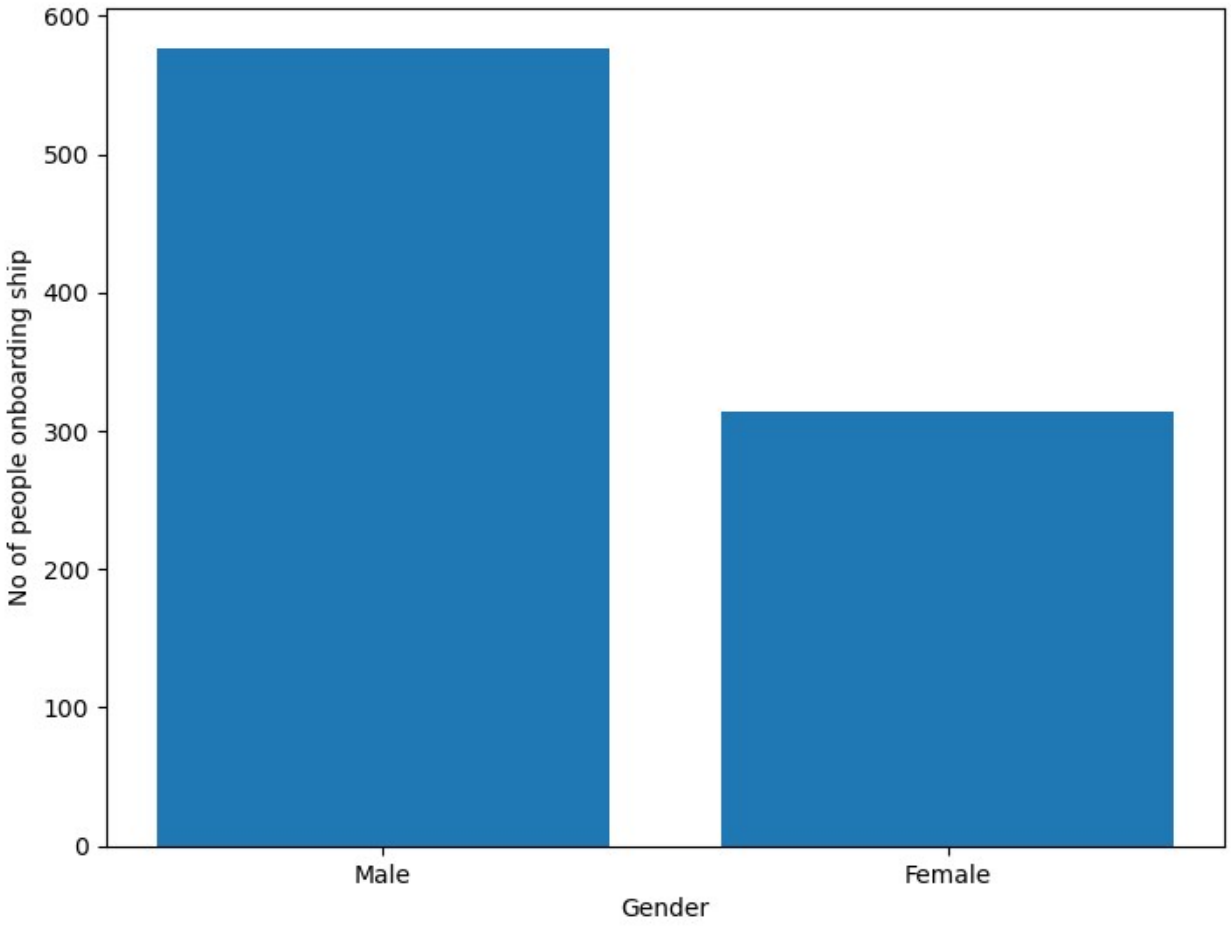
```
male_ind = len(train_data[train_data['Sex'] == 'male'])
print("No of Males in Titanic:",male_ind)
```

```
No of Males in Titanic: 577
```

```
female_ind = len(train_data[train_data['Sex'] == 'female'])
print("No of Females in Titanic:",female_ind)
```

```
No of Females in Titanic: 314
```

```
#Plotting
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
gender = ['Male','Female']
index = [577,314]
ax.bar(gender,index)
plt.xlabel("Gender")
plt.ylabel("No of people onboarding ship")
plt.show()
```

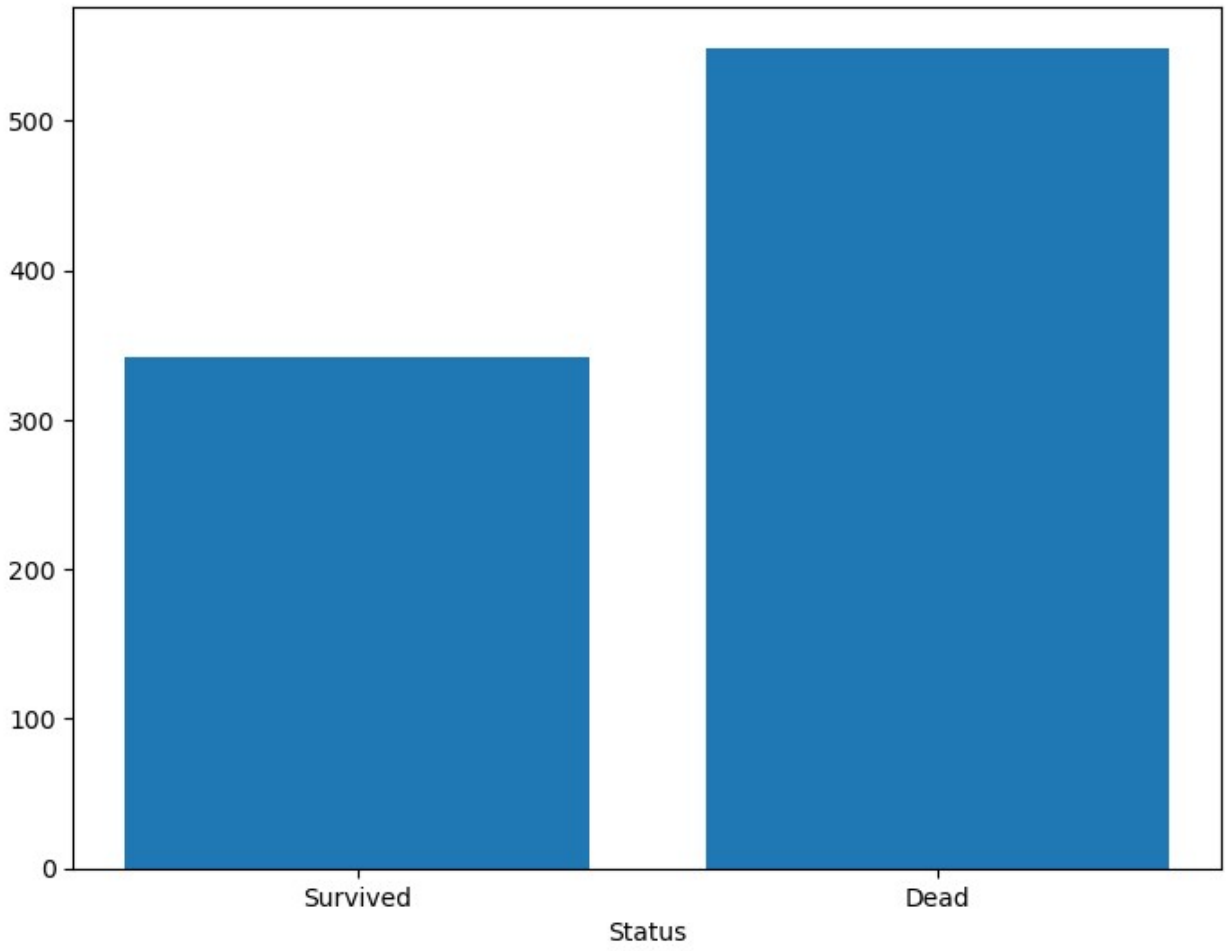


```
alive = len(train_data[train_data['Survived'] == 1])
dead = len(train_data[train_data['Survived'] == 0])

train_data.groupby('Sex')[['Survived']].mean()
```

```
        Survived
Sex
female  0.742038
male    0.188908
```

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
status = ['Survived','Dead']
ind = [alive,dead]
ax.bar(status,ind)
plt.xlabel("Status")
plt.show()
```

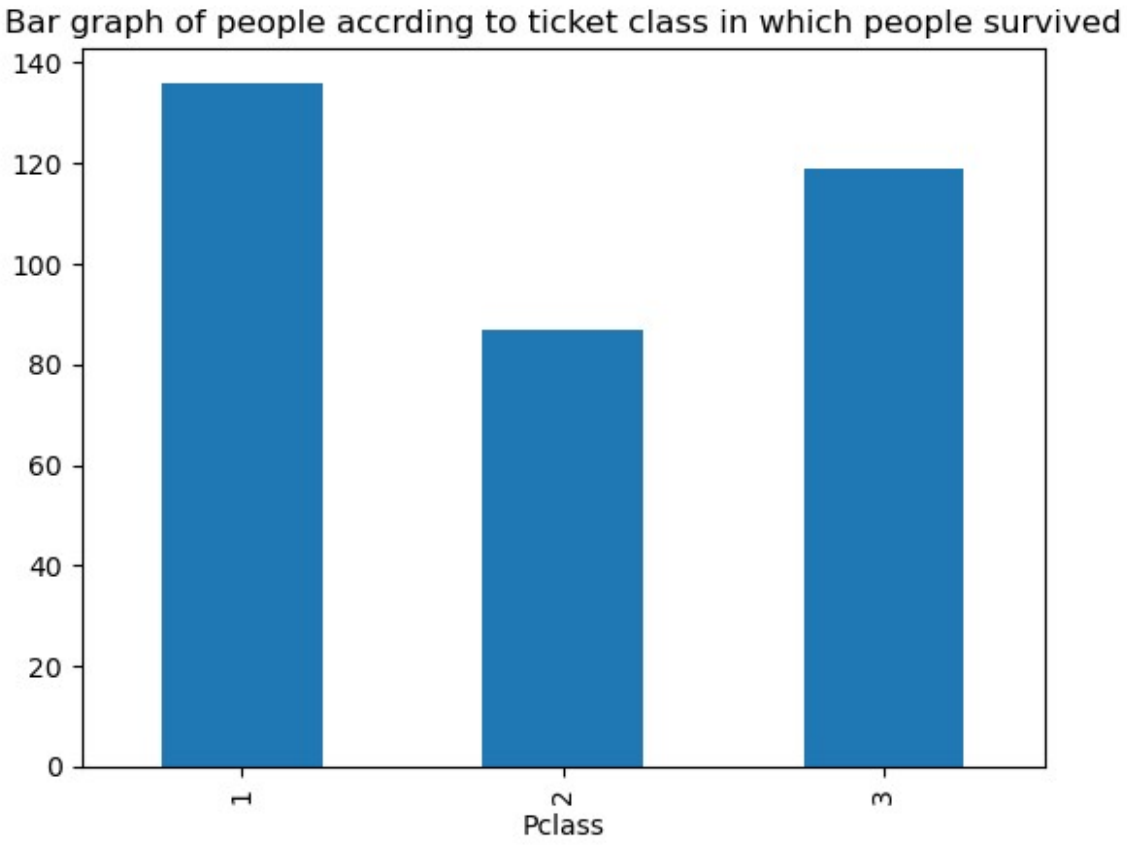


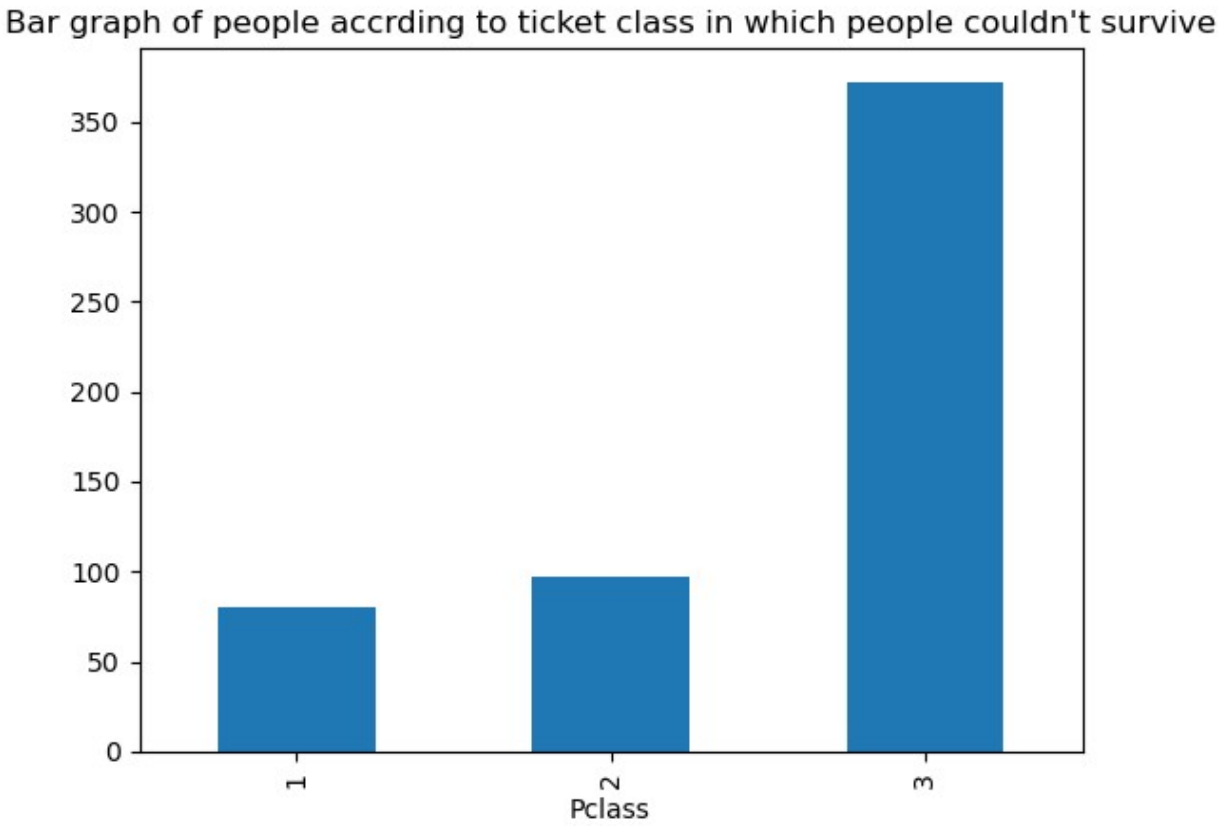

```
plt.figure(1)
train_data.loc[train_data['Survived'] == 1,
```

```
'Pclass'].value_counts().sort_index().plot.bar()
plt.title('Bar graph of people accrding to ticket class in which
people survived')


plt.figure(2)
train_data.loc[train_data['Survived'] == 0,
'Pclass'].value_counts().sort_index().plot.bar()
plt.title('Bar graph of people accrding to ticket class in which
people couldn\'t survive')
```

```
Text(0.5, 1.0, "Bar graph of people accrding to ticket class in which
people couldn't survive")
```



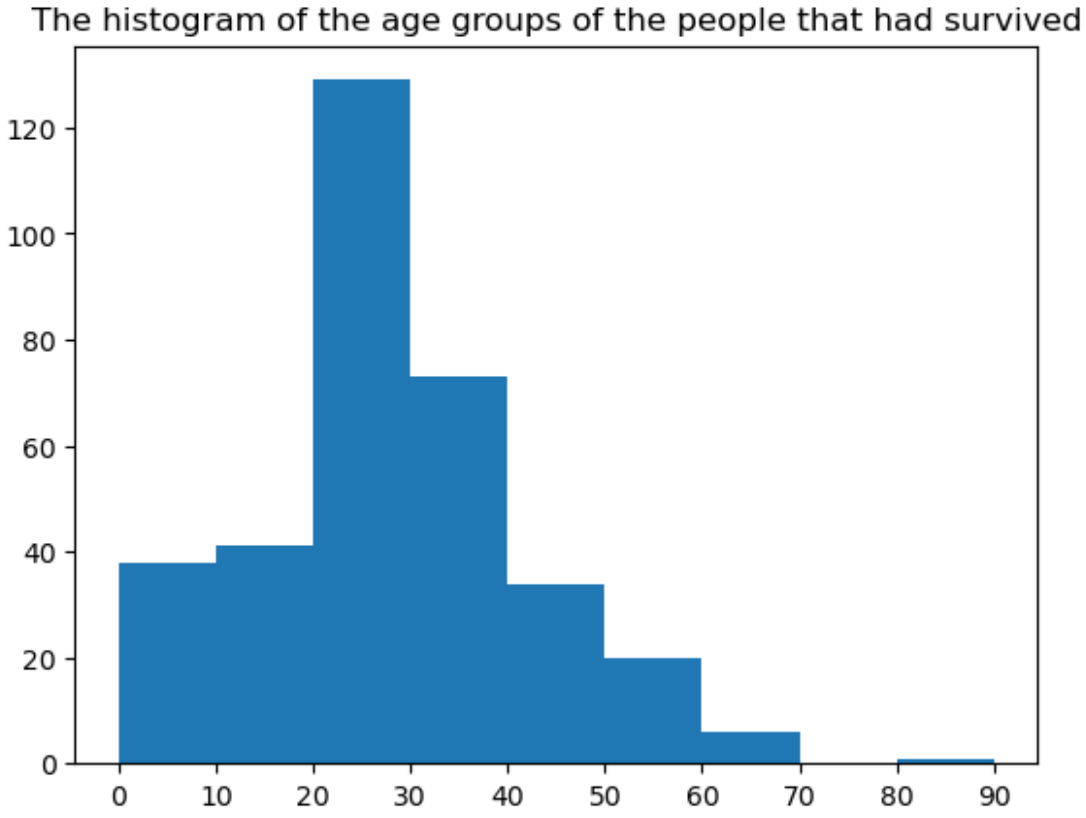


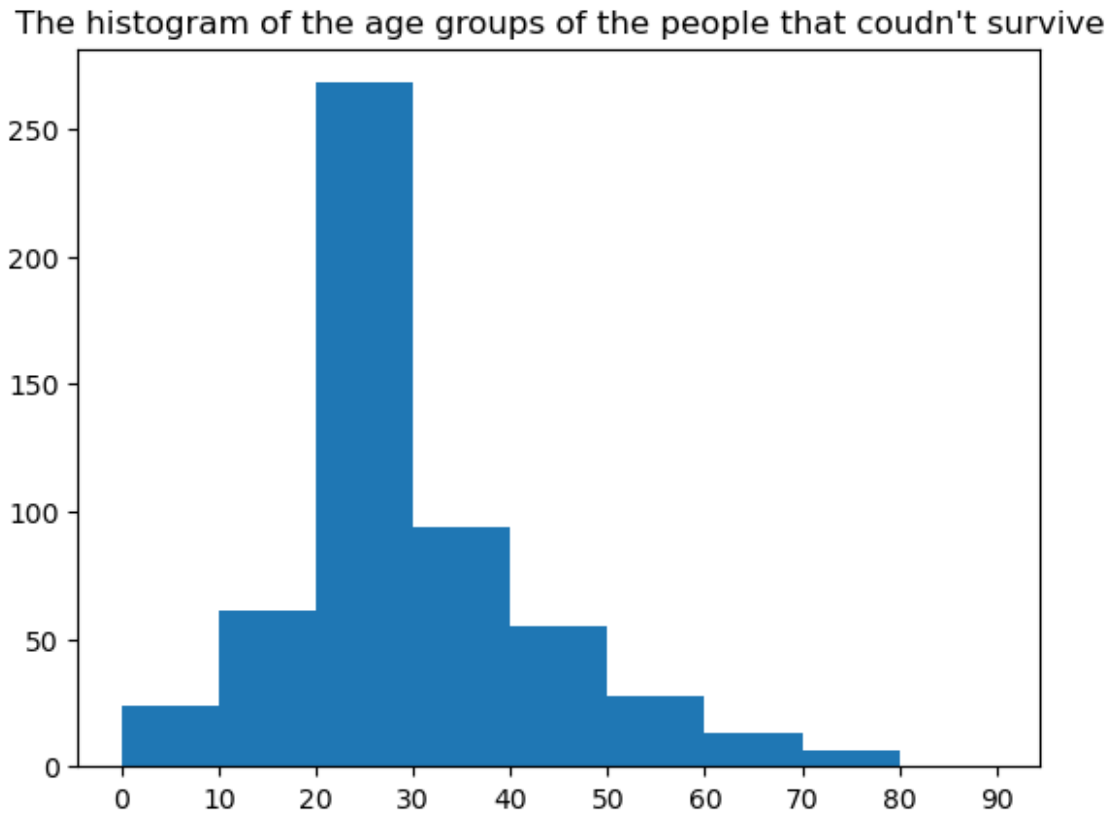

```
plt.figure(1)
age  = train_data.loc[train_data.Survived == 1, 'Age']
plt.title('The histogram of the age groups of the people that had
survived')
plt.hist(age, np.arange(0,100,10))
plt.xticks(np.arange(0,100,10))


plt.figure(2)
age  = train_data.loc[train_data.Survived == 0, 'Age']
plt.title('The histogram of the age groups of the people that coudn\'t
survive')
plt.hist(age, np.arange(0,100,10))
plt.xticks(np.arange(0,100,10))
```

```
([<matplotlib.axis.XTick at 0x1f75cfbae70>,
  <matplotlib.axis.XTick at 0x1f75d512810>,
  <matplotlib.axis.XTick at 0x1f75cf8a930>,
  <matplotlib.axis.XTick at 0x1f75d54cb90>,
  <matplotlib.axis.XTick at 0x1f75d54d460>,
  <matplotlib.axis.XTick at 0x1f75d54dd60>,
  <matplotlib.axis.XTick at 0x1f75d54e6c0>,
  <matplotlib.axis.XTick at 0x1f75d54cf20>,
  <matplotlib.axis.XTick at 0x1f75d54f020>,
```

```
 <matplotlib.axis.XTick at 0x1f75d54f980>],
[Text(0, 0, '0'),
 Text(10, 0, '10'),
 Text(20, 0, '20'),
 Text(30, 0, '30'),
 Text(40, 0, '40'),
 Text(50, 0, '50'),
 Text(60, 0, '60'),
 Text(70, 0, '70'),
 Text(80, 0, '80'),
 Text(90, 0, '90')])
```






```
train_data[["SibSp", "Survived"]].groupby(['SibSp'],
as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

```
train_data[["Pclass", "Survived"]].groupby(['Pclass'],
as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

```
train_data[["Age", "Survived"]].groupby(['Age'],
as_index=False).mean().sort_values(by='Age', ascending=True)
```

	Age	Survived
0	0.42	1.0

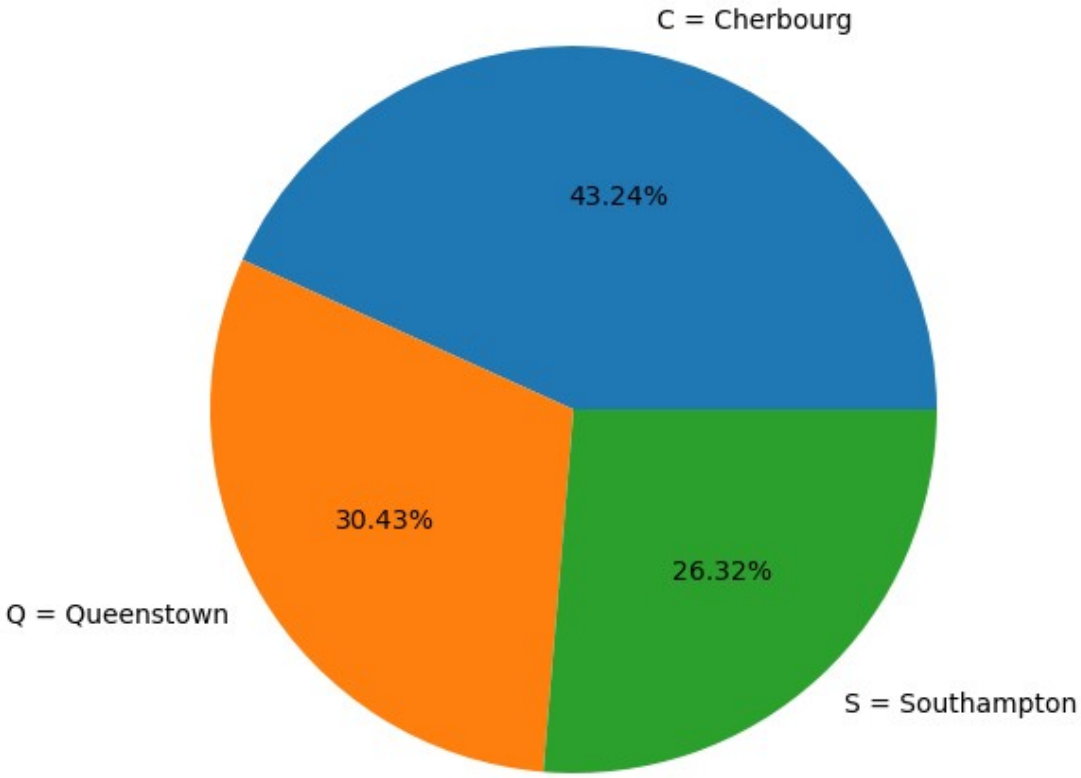
```
1    0.67       1.0
2    0.75       1.0
3    0.83       1.0
4    0.92       1.0
..    ...       ...
84  70.00       0.0
85  70.50       0.0
86  71.00       0.0
87  74.00       0.0
88  80.00       1.0

[89 rows x 2 columns]
```

```
train_data[["Embarked", "Survived"]].groupby(['Embarked'],
as_index=False).mean().sort_values(by='Survived', ascending=False)
```

```
  Embarked  Survived
0        C  0.553571
1        Q  0.389610
2        S  0.339009
```

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
l = ['C = Cherbourg', 'Q = Queenstown', 'S = Southampton']
s = [0.553571,0.389610,0.336957]
ax.pie(s, labels = l,autopct='%1.2f%%')
plt.show()
```




```
test_data.describe(include="all")
```

	PassengerId	Pclass	Name	Sex	Age	\
count	418.000000	418.000000	418	418	418.000000	
unique	NaN	NaN	418	2	NaN	
top	NaN	NaN	Kelly, Mr. James	male	NaN	
freq	NaN	NaN	1	266	NaN	
mean	1100.500000	2.265550	NaN	NaN	30.272590	
std	120.810458	0.841838	NaN	NaN	12.634534	
min	892.000000	1.000000	NaN	NaN	0.170000	
25%	996.250000	1.000000	NaN	NaN	23.000000	
50%	1100.500000	3.000000	NaN	NaN	30.272590	
75%	1204.750000	3.000000	NaN	NaN	35.750000	
max	1309.000000	3.000000	NaN	NaN	76.000000	

	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	418.000000	418.000000	418	418.000000	418	418
unique	NaN	NaN	363	NaN	76	3

```
top            NaN         NaN  PC 17608         NaN  B57 B59 B63 B66
S
freq           NaN         NaN         5         NaN              330
270
mean      0.447368    0.392344       NaN   35.627188              NaN
NaN
std       0.896760    0.981429       NaN   55.840500              NaN
NaN
min       0.000000    0.000000       NaN    0.000000              NaN
NaN
25%       0.000000    0.000000       NaN    7.895800              NaN
NaN
50%       0.000000    0.000000       NaN   14.454200              NaN
NaN
75%       1.000000    0.000000       NaN   31.500000              NaN
NaN
max       8.000000    9.000000       NaN  512.329200              NaN
NaN
```

```
#Droping Useless Columns
train_data = train_data.drop(['Ticket'], axis = 1)
test_data = test_data.drop(['Ticket'], axis = 1)

train_data = train_data.drop(['Cabin'], axis = 1)
test_data = test_data.drop(['Cabin'], axis = 1)
train_data = train_data.drop(['Name'], axis = 1)
test_data = test_data.drop(['Name'], axis = 1)

#Feature Selection
column_train=['Age','Pclass','SibSp','Parch','Fare','Sex','Embarked']
#training values
X=train_data[column_train]
#target value
Y=train_data['Survived']

X['Age'].isnull().sum()
X['Pclass'].isnull().sum()
X['SibSp'].isnull().sum()
X['Parch'].isnull().sum()
X['Fare'].isnull().sum()
X['Sex'].isnull().sum()
X['Embarked'].isnull().sum()
```

```
0
```

```
X['Age']=X['Age'].fillna(X['Age'].median())
X['Age'].isnull().sum()
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\2050903711.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X['Age']=X['Age'].fillna(X['Age'].median())

0

X['Embarked'] = train_data['Embarked'].fillna(method ='pad')
X['Embarked'].isnull().sum()

C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\586271917.py:1:
FutureWarning: Series.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  X['Embarked'] = train_data['Embarked'].fillna(method ='pad')
C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\586271917.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X['Embarked'] = train_data['Embarked'].fillna(method ='pad')

0

d={'male':0, 'female':1}
X['Sex']=X['Sex'].apply(lambda x:d[x])
X['Sex'].head()

C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\1173502709.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X['Sex']=X['Sex'].apply(lambda x:d[x])

0    0
1    1
2    1
3    1
4    0
Name: Sex, dtype: int64
```

```
e={'C':0, 'Q':1 ,'S':2}
X['Embarked']=X['Embarked'].apply(lambda x:e[x])
X['Embarked'].head()
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_11628\4035217270.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X['Embarked']=X['Embarked'].apply(lambda x:e[x])
```

```
0    2
1    0
2    2
3    2
4    2
Name: Embarked, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test =
train_test_split(X,Y,test_size=0.3,random_state=7)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,Y_pred))
```

```
Accuracy Score: 0.7574626865671642
```

```
from sklearn.metrics import accuracy_score,confusion_matrix
confusion_mat = confusion_matrix(Y_test,Y_pred)
print(confusion_mat)
```

```
[[130  26]
 [ 39  73]]
```

```
from sklearn.svm import SVC
model1 = SVC()
model1.fit(X_train,Y_train)

pred_y = model1.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Acc=",accuracy_score(Y_test,pred_y))
```

```
Acc= 0.6604477611940298
```

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,pred_y)
print(confusion_mat)
print(classification_report(Y_test,pred_y))
```

```
[[149   7]
 [ 84  28]]
              precision    recall  f1-score   support

           0       0.64      0.96      0.77       156
           1       0.80      0.25      0.38       112

    accuracy                           0.66       268
   macro avg       0.72      0.60      0.57       268
weighted avg       0.71      0.66      0.61       268
```

```
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=5)
model2.fit(X_train,Y_train)
y_pred2 = model2.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred2))
```

```
Accuracy Score: 0.6492537313432836
```

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,y_pred2)
print(confusion_mat)
print(classification_report(Y_test,y_pred2))
```

```
[[126  30]
 [ 64  48]]
              precision    recall  f1-score   support

           0       0.66      0.81      0.73       156
           1       0.62      0.43      0.51       112

    accuracy                           0.65       268
   macro avg       0.64      0.62      0.62       268
weighted avg       0.64      0.65      0.64       268
```

```
from sklearn.naive_bayes import GaussianNB
model3 = GaussianNB()
model3.fit(X_train,Y_train)
y_pred3 = model3.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred3))
```

```
Accuracy Score: 0.7686567164179104
```

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,y_pred3)
print(confusion_mat)
print(classification_report(Y_test,y_pred3))
```

```
[[129  27]
 [ 35  77]]
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       156
           1       0.74      0.69      0.71       112

    accuracy                           0.77       268
   macro avg       0.76      0.76      0.76       268
weighted avg       0.77      0.77      0.77       268
```

```
from sklearn.tree import DecisionTreeClassifier
model4 = DecisionTreeClassifier(criterion='entropy',random_state=7)
model4.fit(X_train,Y_train)
y_pred4 = model4.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred4))
```

```
Accuracy Score: 0.7201492537313433
```

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
confusion_mat = confusion_matrix(Y_test,y_pred4)
print(confusion_mat)
print(classification_report(Y_test,y_pred4))
```

```
[[124  32]
 [ 43  69]]
              precision    recall  f1-score   support

           0       0.74      0.79      0.77       156
           1       0.68      0.62      0.65       112

    accuracy                           0.72       268
   macro avg       0.71      0.71      0.71       268
weighted avg       0.72      0.72      0.72       268
```

```
results = pd.DataFrame({
    'Model': ['Logistic Regression','Support Vector Machines', 'Naive
Bayes','KNN' ,'Decision Tree'],
    'Score': [0.75,0.66,0.76,0.66,0.74]})

result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

Score	Model
0.76	Naive Bayes
0.75	Logistic Regression
0.74	Decision Tree
0.66	Support Vector Machines
0.66	KNN