In [170]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
```

Exploratory Data Exploration

In [171]:

```
## Importing train dataset
df_train = pd.read_csv("C:/Users/computer world/OneDrive/Desktop/Titanic-Dataset.csv")
```

In [172]:

```
df_train.head()
```

Out[172]:

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

In [173]:
```
## Let's have a look at bottom five rows
df_train.tail()
```

Out[173]:

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

In [174]:
```
## Checking for  the number of rows and columns  in the dataset
print(f"Number of rows :{df_train.shape[0]} \nNumber of columns:{df_train.shape[1]}")
```

```
Number of rows :891
Number of columns:12
```

In [175]:
```
df_train = df_train.drop(["Name", "Ticket", "Cabin"], axis=1)
```

In [176]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 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   Sex          891 non-null    object 
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64  
 6   Parch        891 non-null    int64  
 7   Fare         891 non-null    float64
 8   Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(2)
memory usage: 62.8+ KB
```

In [177]:

```
df_train.dtypes
```

Out[177]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Sex             object
Age            float64
SibSp            int64
Parch            int64
Fare           float64
Embarked        object
dtype: object
```

In [178]:

```
df_train.isna().sum()
```

Out[178]:

```
PassengerId      0
Survived         0
Pclass           0
Sex              0
Age            177
SibSp            0
Parch            0
Fare             0
Embarked         2
dtype: int64
```

In [179]:

```
df_train["Age"]= df_train["Age"].fillna(df_train["Age"].mean())
df_train["Age"].isna().sum()
```

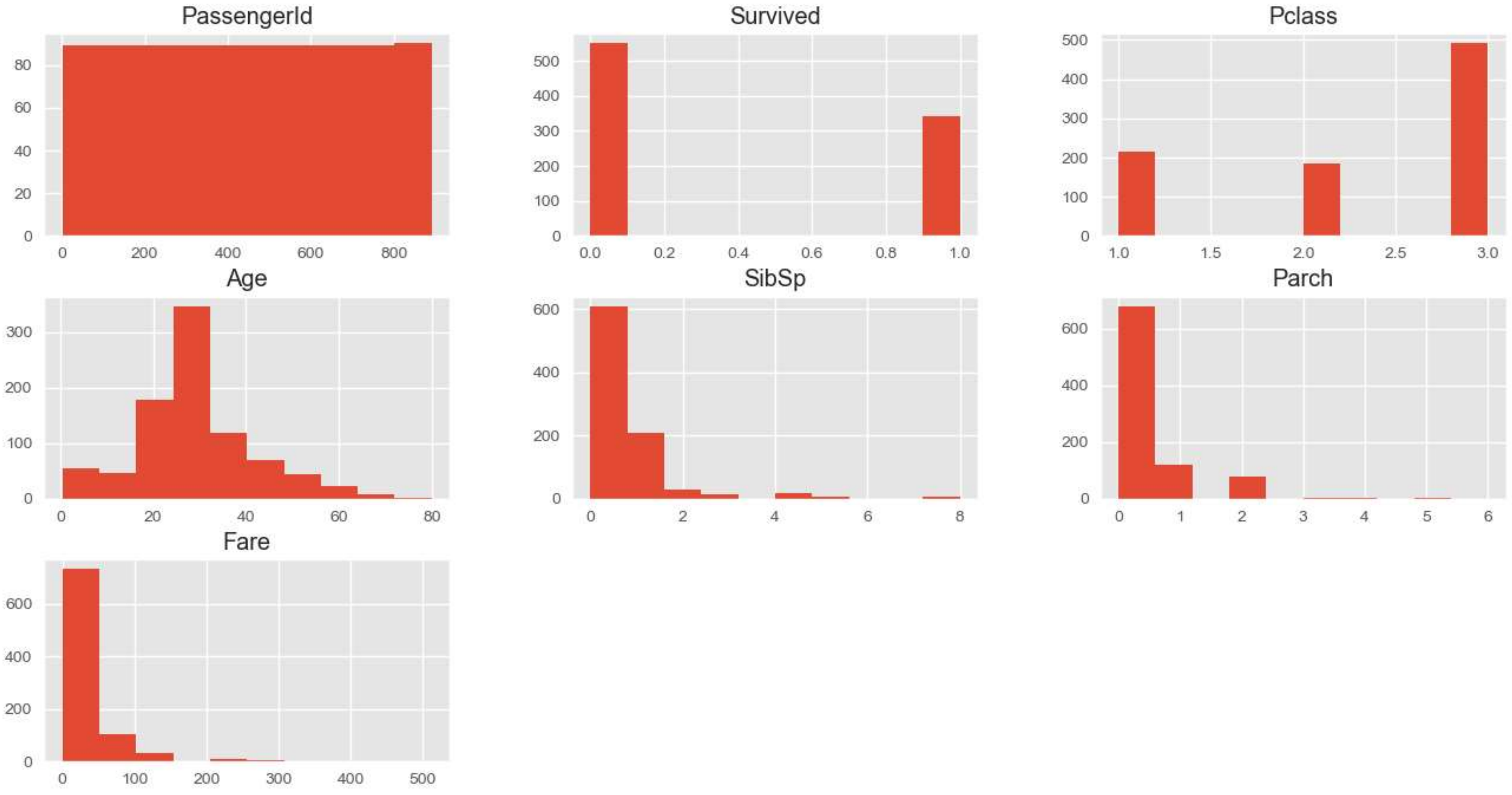
Out[179]:

```
0
```

In [180]:
```
df_train.isna().sum()
```

Out[180]:
```
PassengerId    0
Survived       0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       2
dtype: int64
```

In [181]:
```
df_train.hist(figsize=(16,8));
```

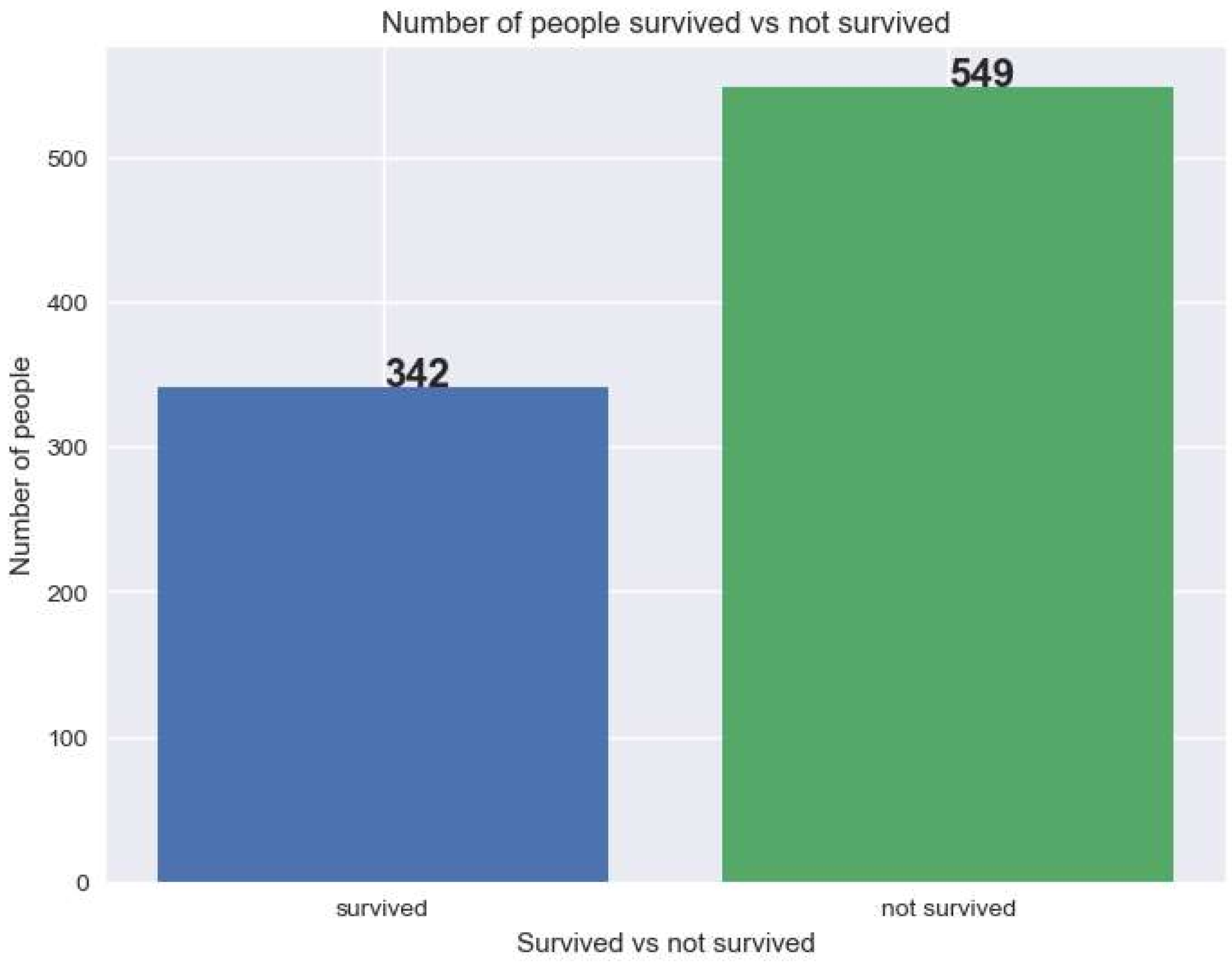


Let compare the number of people survived vs not survived

In [182]:

```
survived = df_train[df_train.Survived==1].count()[0]
not_survived = df_train[df_train.Survived==0].count()[0]
text = ["survived","not survived"]
label = [survived,not_survived]
plt.style.use('seaborn')
plt.figure(figsize=(8,6),dpi=100)
for bar in range(0,2):
    plt.bar(text[bar],label[bar])
    plt.text(text[bar],label[bar],str(label[bar]),fontsize=16, fontweight='bold')
plt.title("Number of people survived vs not survived")
plt.xlabel("Survived vs not survived")
plt.ylabel("Number of people")
plt.show()
```

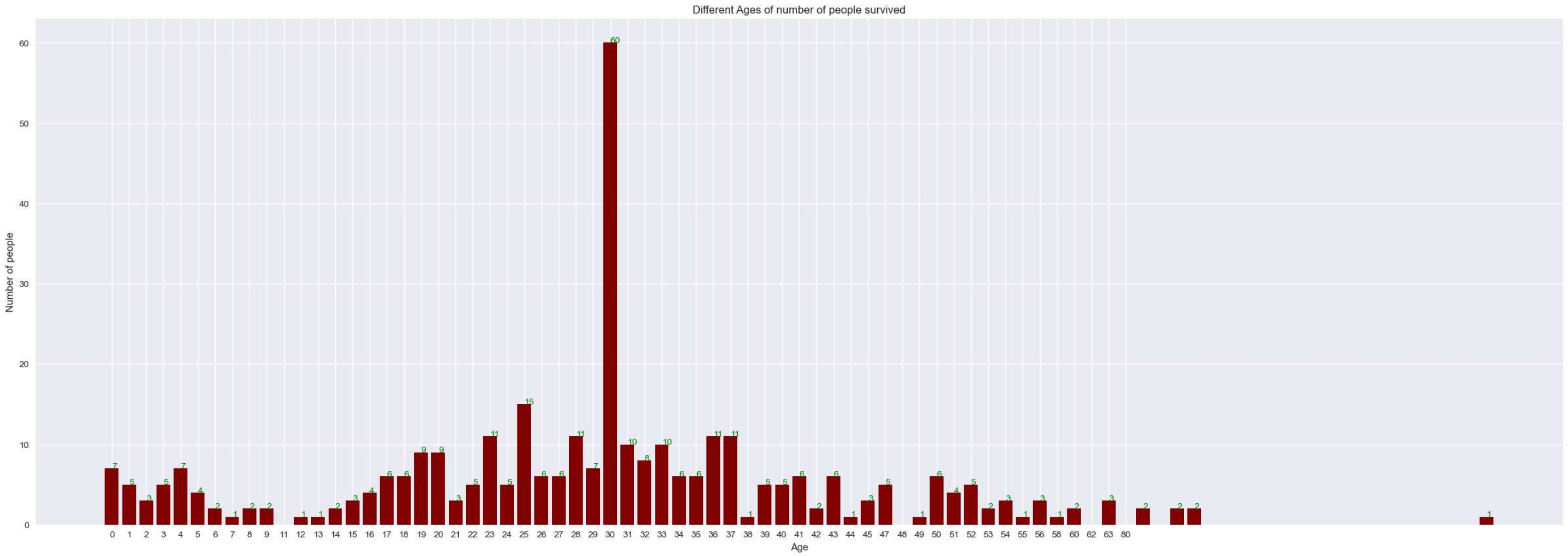
```
C:\Users\computer world\AppData\Local\Temp\ipykernel_2328\1395879324.py:5: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles sh
ipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use
the seaborn API instead.
  plt.style.use('seaborn')
```



Let’s find out the survival rate based on age

In [183]:

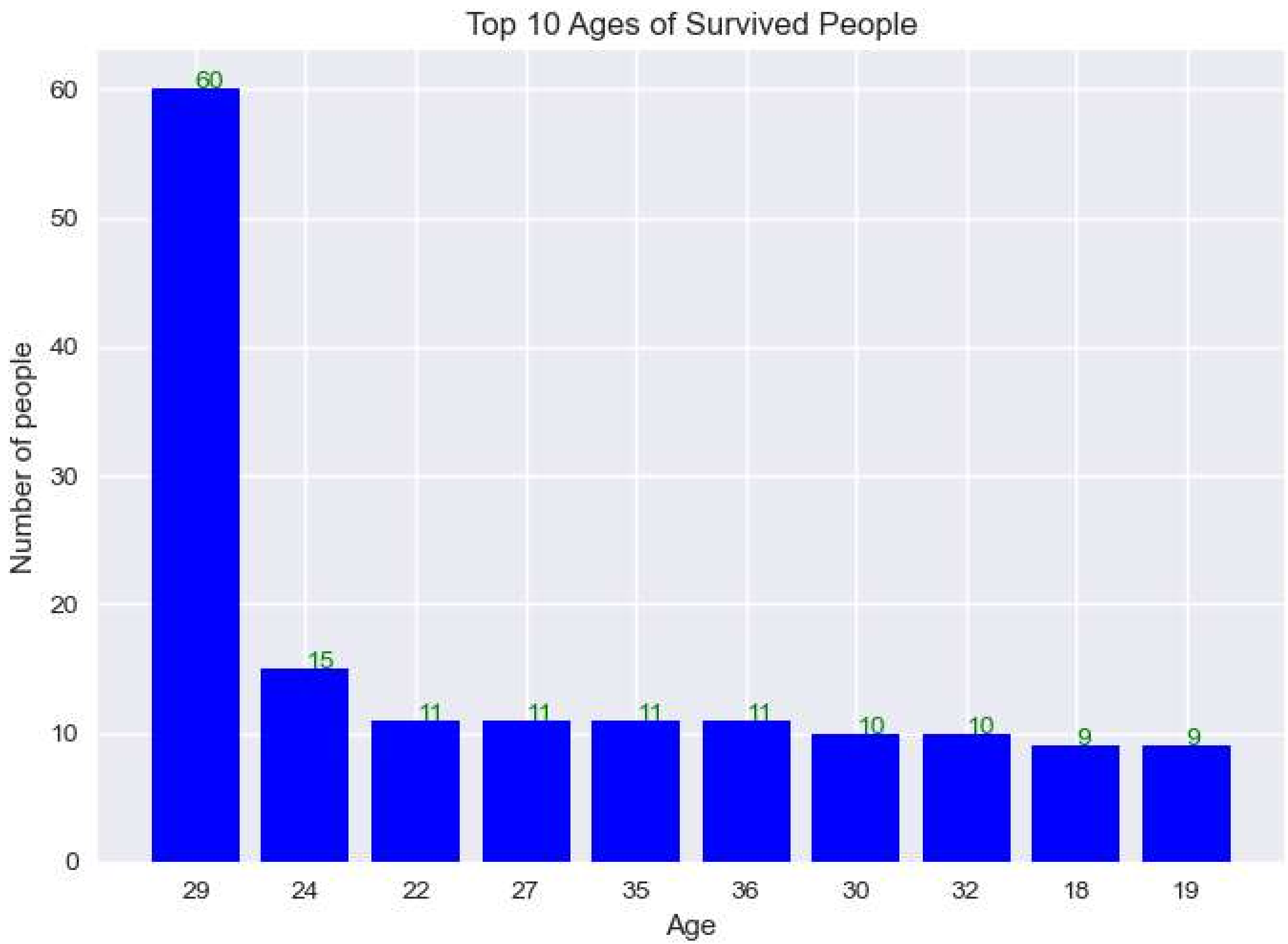
```python
df_train.Age = df_train.Age.astype(int)
ages = df_train[df_train.Survived==1]["Age"].sort_values()
dc =  {}
for age in ages:
    if age not in dc.keys():
        dc[age] = 1
    else:
        dc[age] +=1
plt.figure(figsize=(30,10))
key = list(dc.keys())
value = list(dc.values())
for index in range(len(key)):
    plt.bar(key[index],value[index],color ='maroon')
    plt.text(key[index],value[index],str(value[index]),color="green")
plt.xticks(np.arange(len(key)),key)
plt.title("Different Ages of number of people survived")
plt.xlabel("Age")
plt.ylabel("Number of people")
plt.show()
```



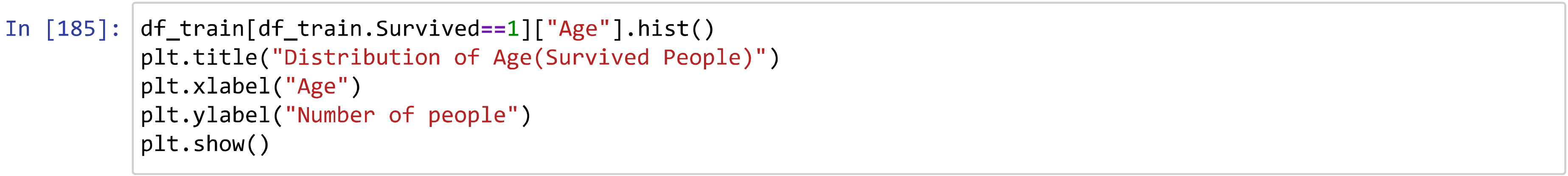
Let’s find top 10 ages of survived people

In [184]:

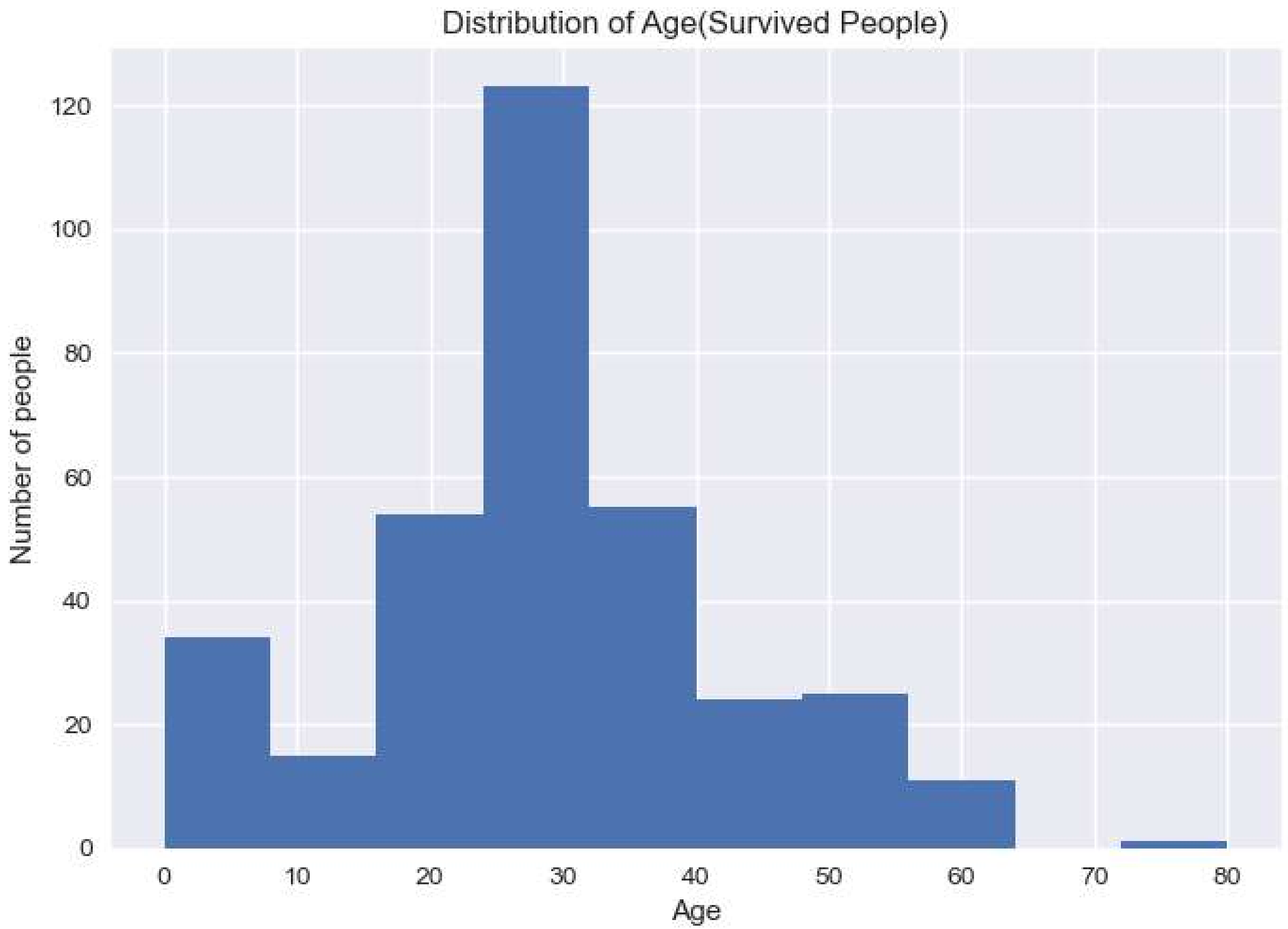
```
dc = {0: 7, 1: 5, 2: 3, 3: 5, 4: 7, 5: 4, 6: 2, 7: 1, 8: 2, 9: 2, 11: 1, 12: 1, 13: 2, 14: 3, 15: 4, 16: 6, 1
dc_sorted = sorted(dc.items(), key=lambda x: x[1], reverse=True)
key_10 = [dc_sorted[i][0] for i in range(len(dc_sorted))][:10]
value_10= [dc_sorted[i][1] for i in range(len(dc_sorted))][:10]
plt.bar(np.arange(len(key_10)),value_10, color ='blue')
for index in range(len(key_10)):
    plt.text(index,value_10[index],str(value_10[index]),color="green")
plt.xticks(np.arange(len(key_10)),key_10)
plt.title("Top 10 Ages of Survived People")
plt.xlabel("Age")
plt.ylabel("Number of people")
plt.show()
```



Distribution of Age



```
In [185]: df_train[df_train.Survived==1]["Age"].hist()
          plt.title("Distribution of Age(Survived People)")
          plt.xlabel("Age")
          plt.ylabel("Number of people")
          plt.show()
```



Replacing Sex [“male”:0 and “female”:1]

In [186]:

```
print(df_train.Sex[:5])
df_train["Sex"]= df_train["Sex"].replace({"female":0, "male":1})
df_train.Sex.head()
```

```
0      male
1    female
2    female
3    female
4      male
Name: Sex, dtype: object
```

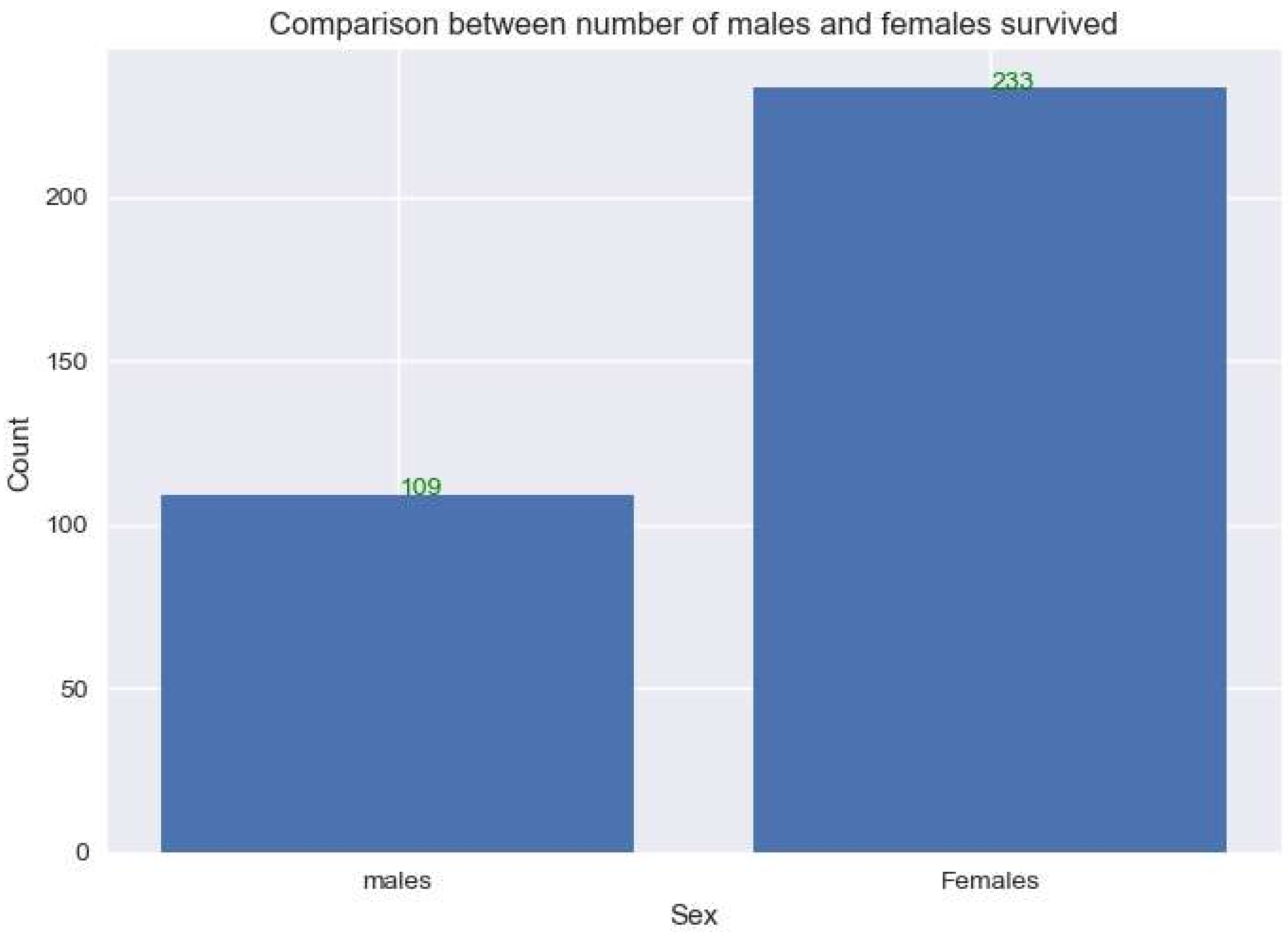
Out[186]:

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

Comparision between number of males and females survivied

In [187]:

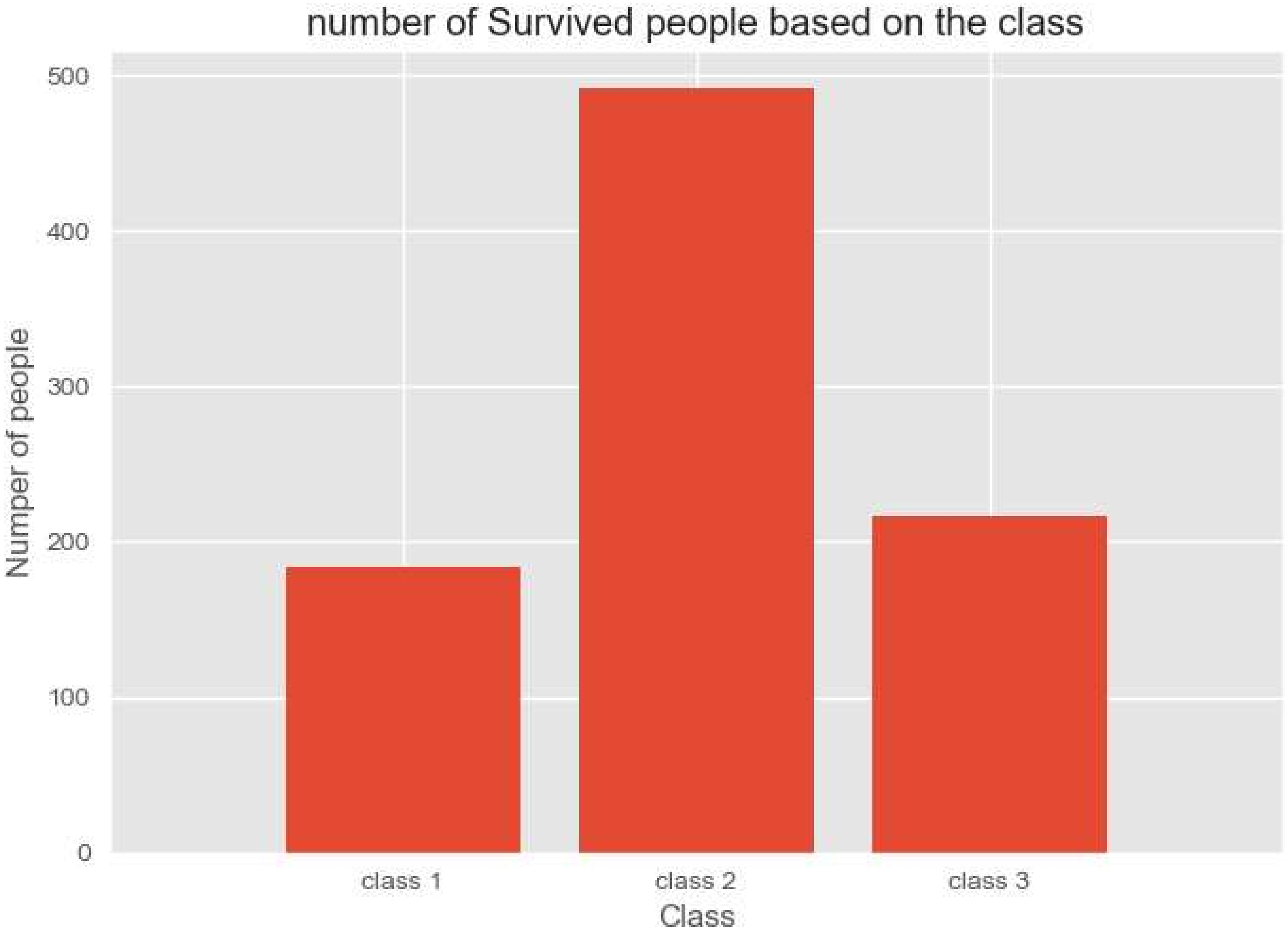
```
males = df_train[(df_train["Survived"]==1) & (df_train.Sex==1)]["Sex"].count()
female = df_train[(df_train["Survived"]==1) & (df_train.Sex==0)]["Sex"].count()
value = [males,female]
labels = ["males","Females"]
plt.bar(np.arange(len(value)),value);
for index in range(len(value)):
    plt.text(index,value[index],str(value[index]),color="green")
plt.xticks(np.arange(len(labels)),labels)
plt.title("Comparison between number of males and females survived")
plt.xlabel("Sex")
plt.ylabel("Count")
plt.style.use("ggplot")
plt.show()
```



Number of Survived people based on the class

In [188]:

```
class_1 = df_train[df_train.Pclass==1].count()[0]
class_2 = df_train[df_train.Pclass==2].count()[0]
class_3 = df_train[df_train.Pclass==3].count()[0]
classs = [class_1,class_2,class_3]
plt.bar(df_train.Pclass.unique(),classs)
plt.xticks(np.arange(5),["","class 1","class 2","class 3",""])
plt.title("number of Survived people based on the class")
plt.xlabel("Class")
plt.ylabel("Numper of people")
plt.show()
```



In [189]:

```
df_train.dropna(inplace=True)
print(df_train.Embarked.unique())
df_train.Embarked = df_train.Embarked.replace({"S":0, "C":1,"Q":2})
df_train.Embarked.isnull().sum()
print(df_train.shape)
```

```
['S' 'C' 'Q']
(889, 9)
```

In [190]:

```
# Everything except target variable
print(df_train.iloc[:,2:-1].head())
X = df_train.iloc[:,2:-1].values

# Target variable
y = df_train.Survived.values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
X_train[:5]
X_test[:5]
```

```
   Pclass  Sex  Age  SibSp  Parch     Fare
0       3    1   22      1      0   7.2500
1       1    0   38      1      0  71.2833
2       3    0   26      0      0   7.9250
3       1    0   35      1      0  53.1000
4       3    1   35      0      0   8.0500
```

Out[190]:

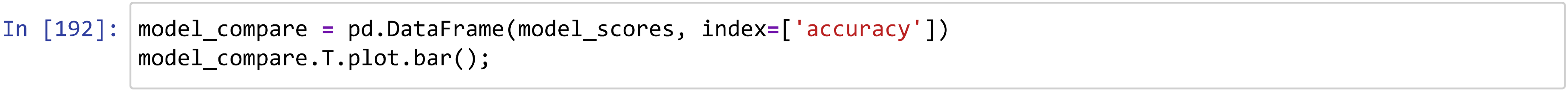
```
array([[  3.    ,   0.    ,  14.    ,   0.    ,   0.    ,   7.8542],
       [  3.    ,   1.    ,  29.    ,   8.    ,   2.    ,  69.55  ],
       [  1.    ,   0.    ,  36.    ,   1.    ,   2.    , 120.    ],
       [  1.    ,   1.    ,  36.    ,   1.    ,   0.    ,  78.85  ],
       [  3.    ,   0.    ,  63.    ,   0.    ,   0.    ,   9.5875]])
```

Training

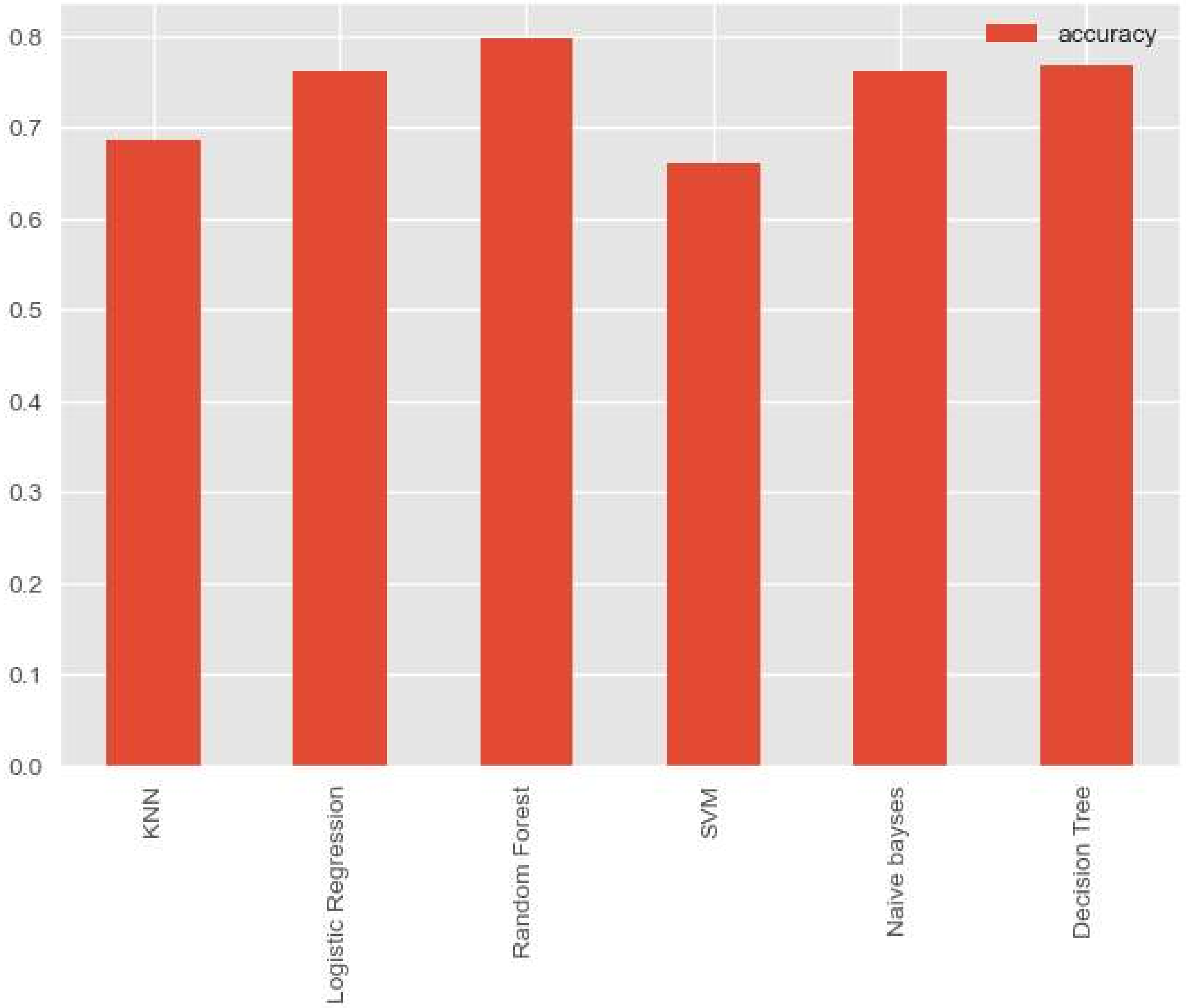
```
In [191]: # Put models in a dictionary
          models = {"KNN": KNeighborsClassifier(),
                     "Logistic Regression": LogisticRegression(),
                     "Random Forest": RandomForestClassifier(),
                    "SVM": SVC(),
                    "Naive bayses":GaussianNB(),
                    "Decision Tree":DecisionTreeClassifier()}

          # Create function to fit and score models
          def fit_and_score(models, X_train,y_train,X_test,y_test):
              # Random seed for reproducible results
              np.random.seed(42)
              # Make a list to keep model scores
              model_scores = {}
              # Loop through models
              for name, model in models.items():
                  # Fit the model to the data
                  model.fit(X_train, y_train)
                  # Evaluate the model and append its score to model_scores
                  model_scores[name] = model.score(X_test, y_test)
              return model_scores
          model_scores = fit_and_score(models=models,
                                       X_train=X_train,
                                       y_train=y_train,
                                       X_test=X_test,
                                       y_test=y_test)
          model_scores
```

```
Out[191]: {'KNN': 0.6868686868686869,
           'Logistic Regression': 0.7609427609427609,
           'Random Forest': 0.797979797979798,
           'SVM': 0.6599326599326599,
           'Naive bayses': 0.7609427609427609,
           'Decision Tree': 0.7676767676767676}
```

Random Forest with an accuracy of 79 is highest.



```
In [192]: model_compare = pd.DataFrame(model_scores, index=['accuracy'])
          model_compare.T.plot.bar();
```

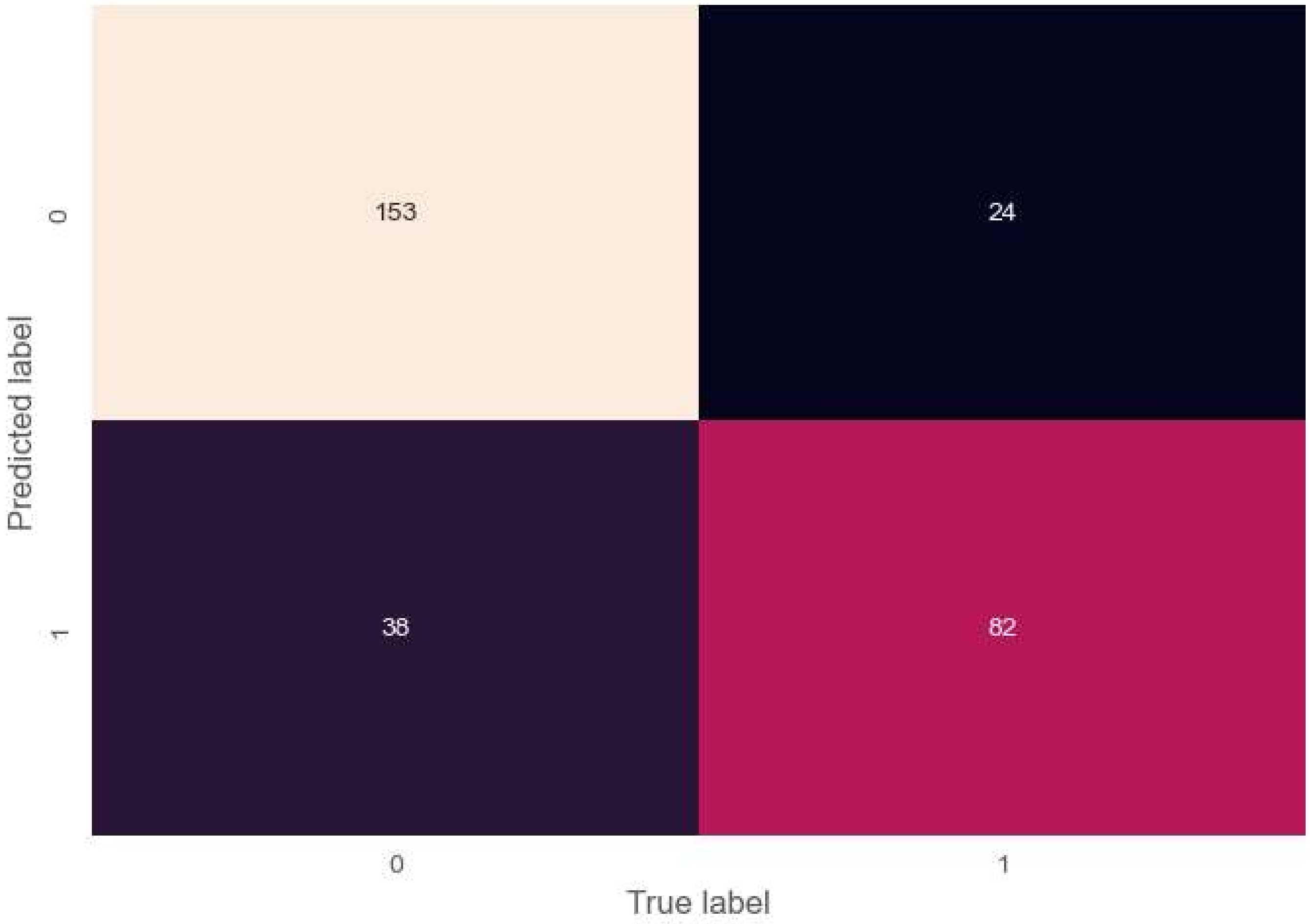


Using Gradient Boost Classifier for getting performance

In [193]:

```
from sklearn.ensemble import GradientBoostingClassifier
gradboost= GradientBoostingClassifier(n_estimators=300, random_state=0).fit(X_train, y_train)
preds= gradboost.predict(X_test)
sns.heatmap(confusion_matrix(y_test,preds), annot=True,cbar=False, fmt='g')
plt.xlabel("True label")
plt.ylabel("Predicted label");
print(gradboost.score(X_test,y_test))
```

```
0.7912457912457912
```



Classification Report

```
In [194]: print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       177
           1       0.77      0.68      0.73       120

    accuracy                           0.79       297
   macro avg       0.79      0.77      0.78       297
weighted avg       0.79      0.79      0.79       297
```

Test Data

In [195]:

```
df_test = pd.read_csv("C:/Users/computer world/OneDrive/Desktop/Titanic-Dataset.csv")
df_test = df_test.drop(["Name", "Ticket", "Cabin"], axis=1)
df_test["Sex"]= df_test["Sex"].replace({"female":0, "male":1})
df_test["Age"]= df_test["Age"].fillna(df_test["Age"].mean())
df_test.Age = df_test.Age.astype(int)
df_test.Embarked = df_test.Embarked.replace({"S":0, "C":1,"Q":2,"nan":3})
df_test["Fare"]= df_test["Fare"].fillna(df_test["Fare"].median())
test_x = df_test.iloc[:,1:-1].values
print(test_x)
data = pd.read_csv("C:/Users/computer world/OneDrive/Desktop/Titanic-Dataset.csv")
test_y = data["Survived"].values
test_y
```

```
[[ 0.       3.       1.      ...  1.       0.       7.25  ]
 [ 1.       1.       0.      ...  1.       0.      71.2833]
 [ 1.       3.       0.      ...  0.       0.       7.925 ]
 ...
 [ 0.       3.       0.      ...  1.       2.      23.45  ]
 [ 1.       1.       1.      ...  0.       0.      30.     ]
 [ 0.       3.       1.      ...  0.       0.       7.75  ]]
```

```
Out[195]: array([0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
                 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
                 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
                 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
                 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
                 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
                 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
                 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
                 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
                 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
                 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
                 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
                 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
                 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
                 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
                 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
                 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
                 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
                 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
                 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
                 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
                 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
                 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
                 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
                 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
                 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
                 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
                 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
                 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1,
                 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
                 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
                 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
                 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
                 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

```
In [ ]:
```