This project focuses on building various machine learning models for Heart Disease prediction

In [1]:
```
## Importing the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import janitor
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
## Load the dataset
df = pd.read_csv("C:/Users/ADMIN/Desktop/Data Science/Datasets/Datasets/heart.csv")
```

In [3]:
```
## View the first few observations of the dataset
df.head(5)
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	

In [4]:
```
## Assess the structure of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [5]:
```
## Checking for missing values
df.isna().sum()
```

Out[5]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [6]:
```
## Check for duplicates
df.duplicated().sum()
```

Out[6]:
```
723
```

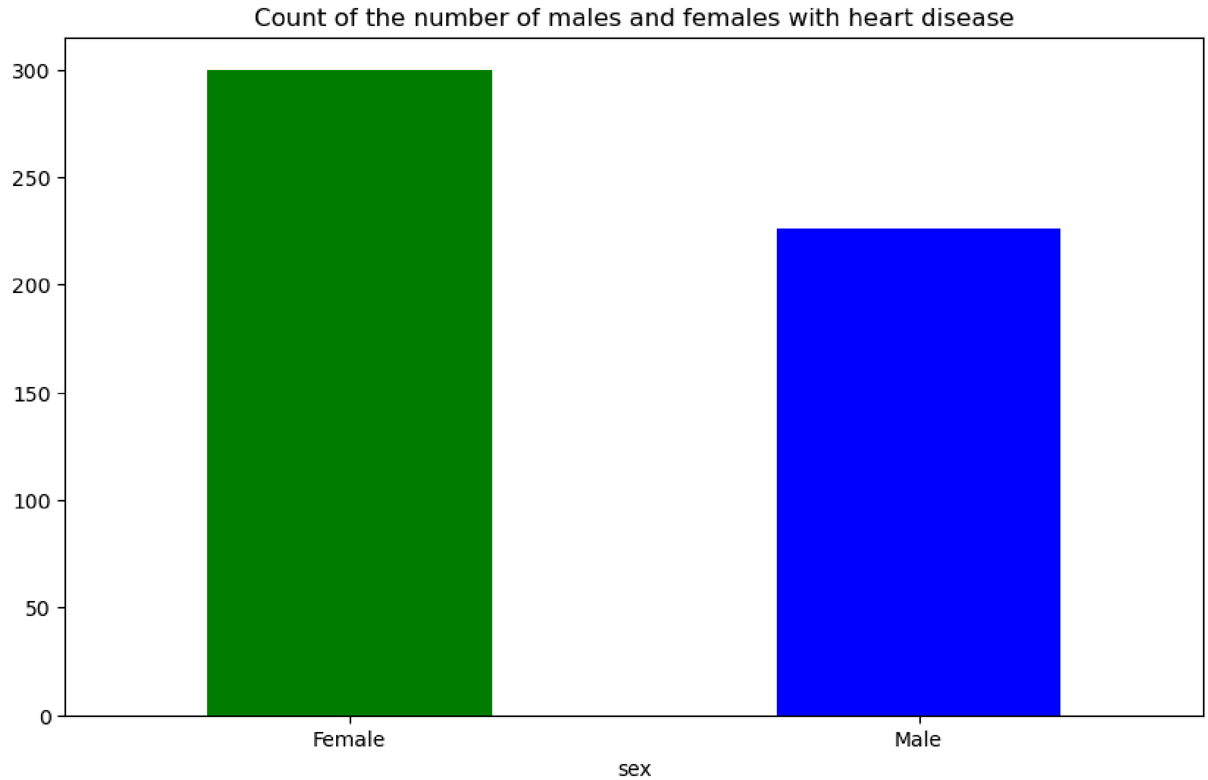
In [7]:
```
## Number of males and females whose heart data is stored in the dataset
df.sex.value_counts()
```

Out[7]:
```
sex
1    713
0    312
Name: count, dtype: int64
```

In [8]:
```
## Count of the number of males and females who have heart disease
df.sex[df.target==1].value_counts()
```

Out[8]:
```
sex
1    300
0    226
Name: count, dtype: int64
```

In [9]:
```
## Visualize your results
df.sex[df.target==1].value_counts().plot(kind='bar',figsize=(10,6),color=['green','
plt.title("Count of the number of males and females with heart disease")
plt.xticks([0,1], labels = ['Female', 'Male'], rotation =0)
plt.show()
```

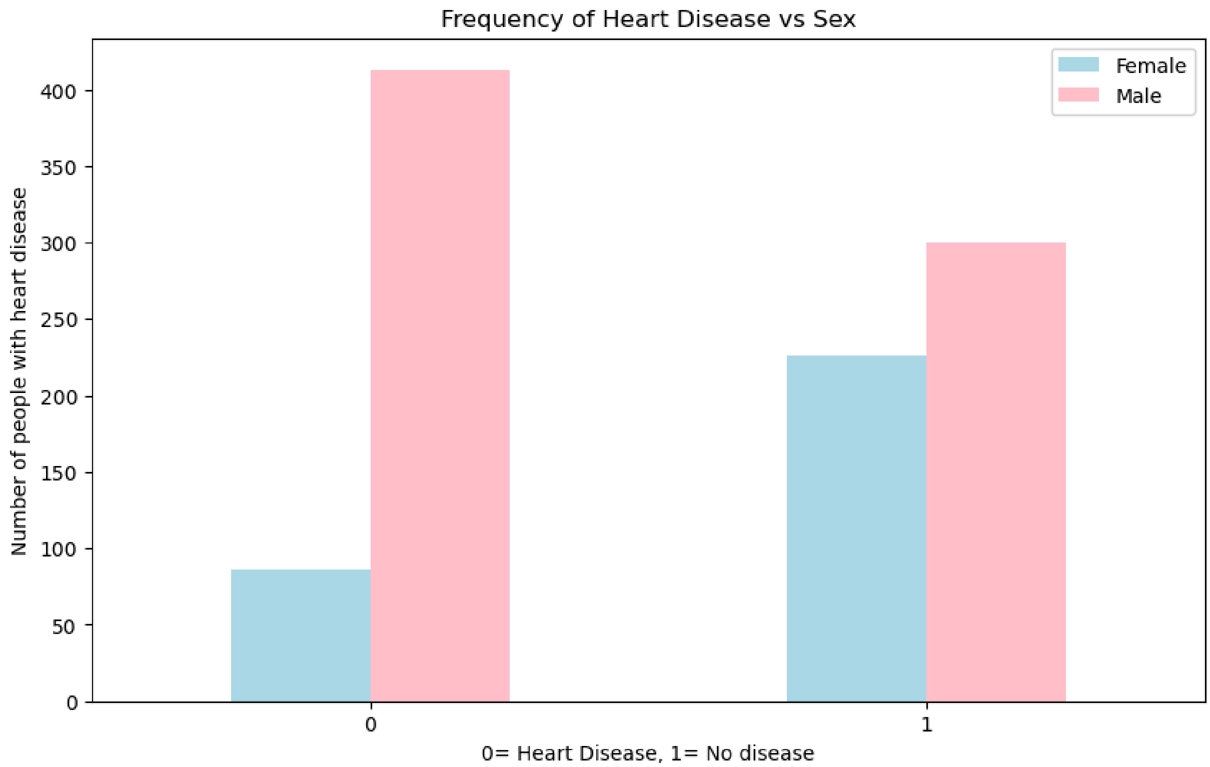


In [10]:
```
## Contingency table
pd.crosstab(df.target,df.sex)
```

Out[10]:

sex	0	1
target		
0	86	413
1	226	300

In [11]:
```
## Visualize the contingency table
pd.crosstab(df.target,df.sex).plot(kind='bar',figsize=(10,6),color=["lightblue","pi
plt.title("Frequency of Heart Disease vs Sex")
plt.xlabel("0= Heart Disease, 1= No disease")
plt.ylabel("Number of people with heart disease")
plt.legend(["Female","Male"])
plt.xticks(rotation=0);
```



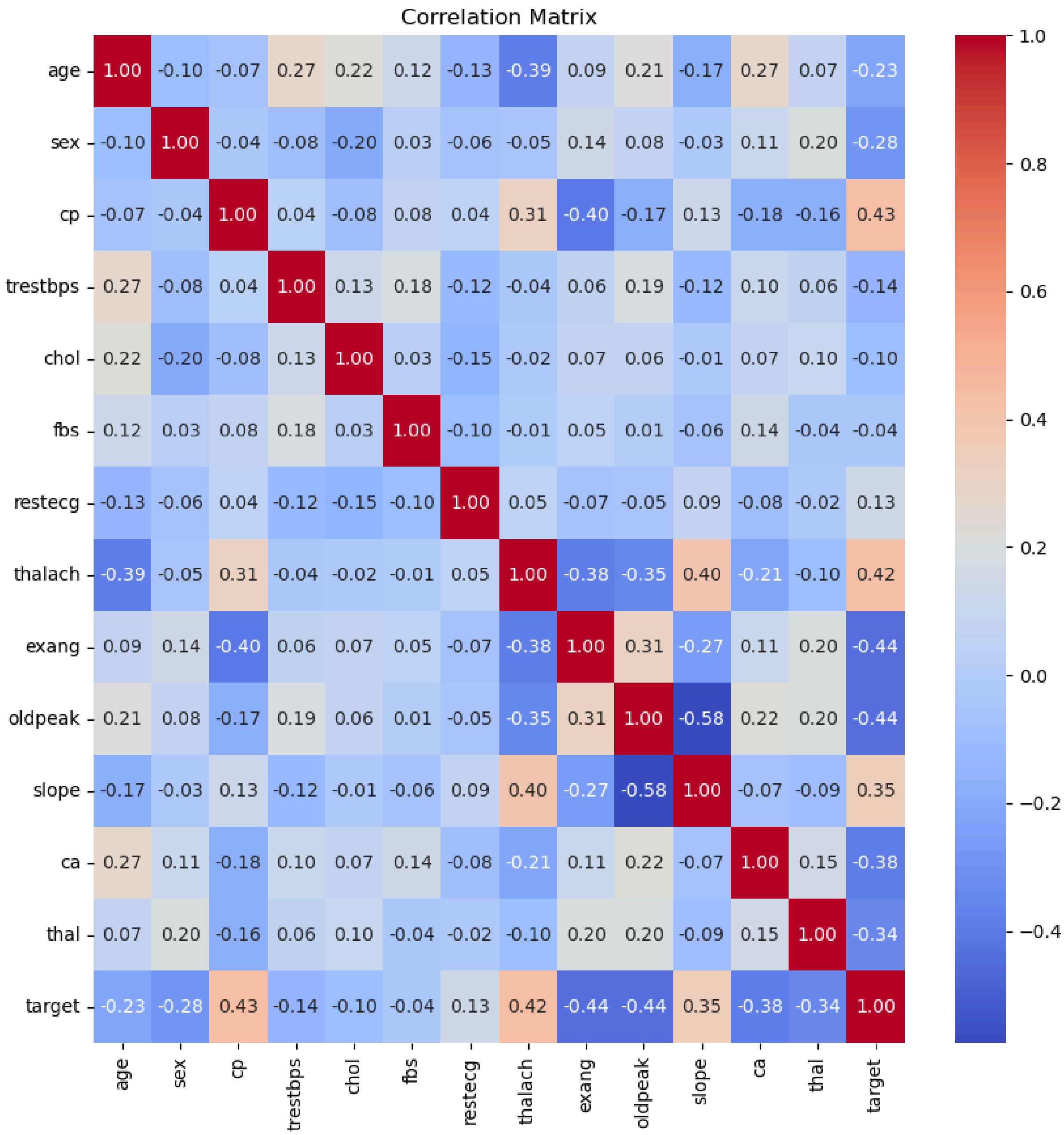
In [12]:

```python
## Building a Correlation Matrix
df.corr()
```

Out[12]:

	age	sex	cp	trestbps	chol	fbs	restecg	thal
age	1.000000	-0.103240	-0.071966	0.271121	0.219823	0.121243	-0.132696	-0.390
sex	-0.103240	1.000000	-0.041119	-0.078974	-0.198258	0.027200	-0.055117	-0.049
cp	-0.071966	-0.041119	1.000000	0.038177	-0.081641	0.079294	0.043581	0.306
trestbps	0.271121	-0.078974	0.038177	1.000000	0.127977	0.181767	-0.123794	-0.039
chol	0.219823	-0.198258	-0.081641	0.127977	1.000000	0.026917	-0.147410	-0.021
fbs	0.121243	0.027200	0.079294	0.181767	0.026917	1.000000	-0.104051	-0.008
restecg	-0.132696	-0.055117	0.043581	-0.123794	-0.147410	-0.104051	1.000000	0.048
thalach	-0.390227	-0.049365	0.306839	-0.039264	-0.021772	-0.008866	0.048411	1.000
exang	0.088163	0.139157	-0.401513	0.061197	0.067382	0.049261	-0.065606	-0.380
oldpeak	0.208137	0.084687	-0.174733	0.187434	0.064880	0.010859	-0.050114	-0.349
slope	-0.169105	-0.026666	0.131633	-0.120445	-0.014248	-0.061902	0.086086	0.395
ca	0.271551	0.111729	-0.176206	0.104554	0.074259	0.137156	-0.078072	-0.207
thal	0.072297	0.198424	-0.163341	0.059276	0.100244	-0.042177	-0.020504	-0.098
target	-0.229324	-0.279501	0.434854	-0.138772	-0.099966	-0.041164	0.134468	0.422

In [13]:
```
## Visualize the correlation matrix
cor_mat=df.corr()
plt.figure(figsize=(10,10))
sns.heatmap(cor_mat,annot=True,cmap="coolwarm", fmt = ".2f")
plt.title("Correlation Matrix")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```



Machine Learning

In [14]:
```
## Standardization using the MinMax Scaling
from sklearn.preprocessing import MinMaxScaler
scal=MinMaxScaler()
feat=['age', 'trestbps', 'thalach' ,'oldpeak' , 'chol']
```

```
df[feat] = scal.fit_transform(df[feat])
df.head()
```

Out[14]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
0	0.479167	1	0	0.292453	0.196347	0	1	0.740458	0	0.161290	2
1	0.500000	1	0	0.433962	0.175799	1	0	0.641221	1	0.500000	0
2	0.854167	1	0	0.481132	0.109589	0	1	0.412214	1	0.419355	0
3	0.666667	1	0	0.509434	0.175799	0	1	0.687023	0	0.000000	2
4	0.687500	0	0	0.415094	0.383562	1	1	0.267176	0	0.306452	1

In [15]:

```
## Creating Features and Target variable
X=df.drop("target",axis=1).values
Y=df.target.values
```

In [16]:

```
## Splitting the data into train and test sets
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,random_state=42,test_size=0.2)
```

In [17]:

```
## Create a function for evaluating metrics
from sklearn.metrics import accuracy_score,recall_score,f1_score,precision_score,ro

def evaluation(Y_test,Y_pred):
  acc=accuracy_score(Y_test,Y_pred)
  rcl=recall_score(Y_test,Y_pred)
  f1=f1_score(Y_test,Y_pred)


  metric_dict={'accuracy': round(acc,3),
               'recall': round(rcl,3),
               'F1 score': round(f1,3),

              }

  return print(metric_dict)
```

Fitting and Comparing different Models

In [18]:

```
## K-Nearest Neighbors
np.random.seed(42)
from sklearn.neighbors import KNeighborsClassifier
Knn_clf=  KNeighborsClassifier()
Knn_clf.fit(X_train,Y_train)
Knn_Y_pred=Knn_clf.predict(X_test)
Knn_score=Knn_clf.score(X_test,Y_test)
#print(Knn_score)
evaluation(Y_test,Knn_Y_pred)
```

```
{'accuracy': 0.859, 'recall': 0.883, 'F1 score': 0.863}
```

In [19]:
```
## Logistic Regression
np.random.seed(42)
from sklearn.linear_model import LogisticRegression
LR_clf=LogisticRegression()
LR_clf.fit(X_train,Y_train)
LR_Y_pred=LR_clf.predict(X_test)
LR_score=LR_clf.score(X_test,Y_test)
#print(LR_score)
evaluation(Y_test,LR_Y_pred)
```

```
{'accuracy': 0.8, 'recall': 0.874, 'F1 score': 0.814}
```

In [20]:
```
## Random Forest
np.random.seed(42)
from sklearn.ensemble import RandomForestClassifier
RF_clf=RandomForestClassifier(n_estimators=450)
RF_clf.fit(X_train,Y_train)
RF_score=RF_clf.score(X_test,Y_test)
RF_Y_pred=RF_clf.predict(X_test)
#print(RF_score)
evaluation(Y_test,RF_Y_pred)
```

```
{'accuracy': 0.985, 'recall': 0.971, 'F1 score': 0.985}
```

In [21]:
```
## Support Vector Machines
np.random.seed(42)
from sklearn.svm import SVC
SVC_clf=SVC()
SVC_clf.fit(X_train,Y_train)
SVC_score=SVC_clf.score(X_test,Y_test)
SVC_Y_pred=SVC_clf.predict(X_test)
#print(SVC_score)
evaluation(Y_test,SVC_Y_pred)
```

```
{'accuracy': 0.883, 'recall': 0.932, 'F1 score': 0.889}
```

Comparing the performance of different models

In [22]:
```
## Model comparison
model_comp = pd.DataFrame({'Model': ['Logistic Regression','Random Forest',
                    'K-Nearest Neighbour','Support Vector Machine'], 'Accuracy': [L
                    RF_score*100,Knn_score*100,SVC_score*100]})
model_comp
```

Out[22]:

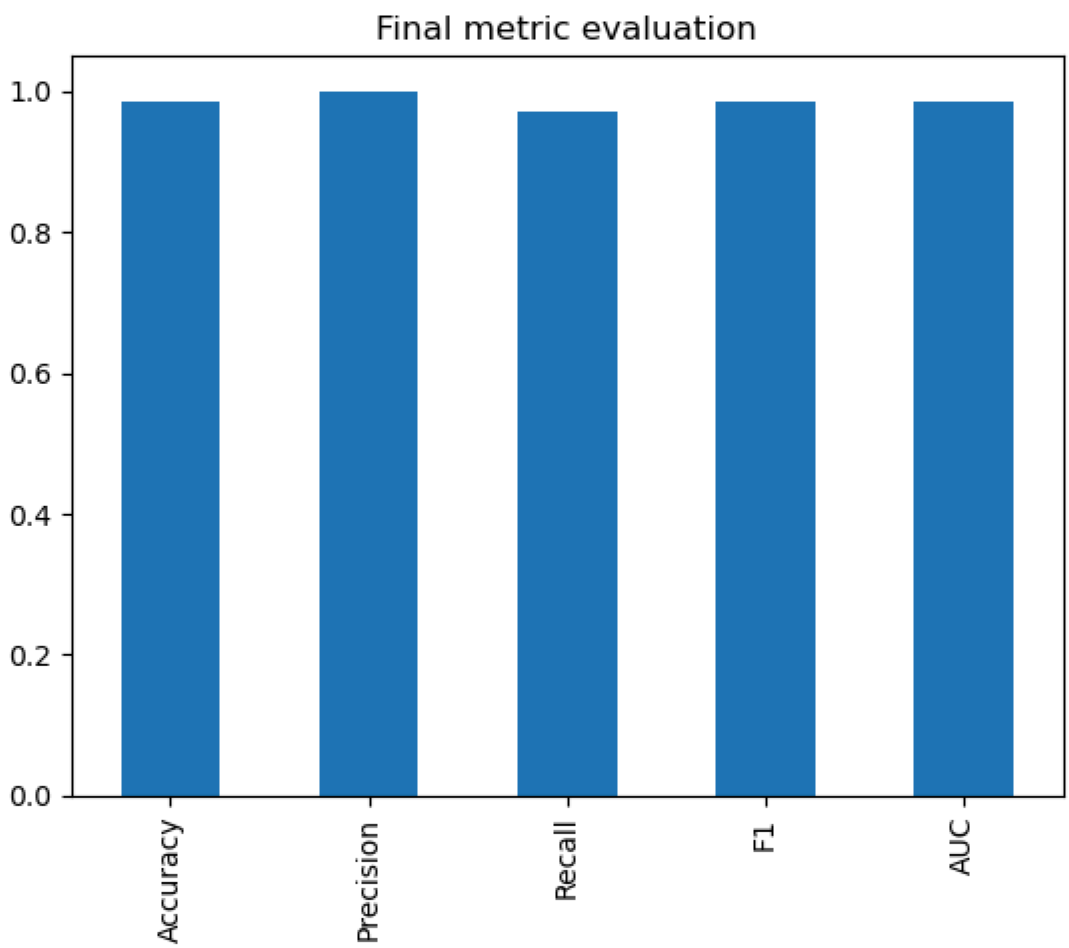
	Model	Accuracy
0	Logistic Regression	80.000000
1	Random Forest	98.536585
2	K-Nearest Neighbour	85.853659
3	Support Vector Machine	88.292683

In [26]:
```
print(" Best evaluation parameters achieved with Random Forest:")
evaluation(Y_test, RF_Y_pred)
```

```
 Best evaluation parameters achieved with Random Forest:
{'accuracy': 0.985, 'recall': 0.971, 'F1 score': 0.985}
```

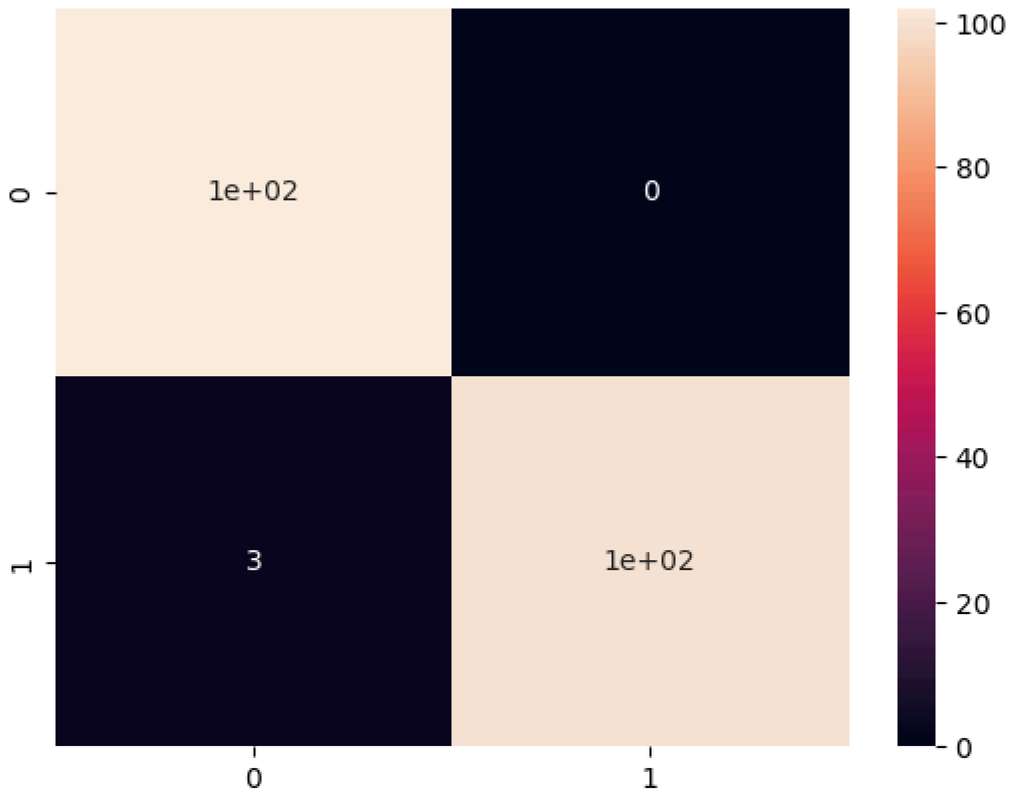
Visualization of the Models Results

In [27]:
```
final_metrics={'Accuracy': RF_clf.score(X_test,Y_test),
                   'Precision': precision_score(Y_test,RF_Y_pred),
                   'Recall': recall_score(Y_test,RF_Y_pred),
                   'F1': f1_score(Y_test,RF_Y_pred),
                   'AUC': roc_auc_score(Y_test,RF_Y_pred)}

metrics=pd.DataFrame(final_metrics,index=[0])

metrics.T.plot.bar(title='Final metric evaluation',legend=False);
```



Create the confusion matrix of the best model

In [28]:
```
from sklearn.metrics import confusion_matrix

fig,ax=plt.subplots()
ax=sns.heatmap(confusion_matrix(Y_test,RF_Y_pred),annot=True,cbar=True);
```



Let's save our model using pickle

In [30]:
```
## Lets save our model using pickle
import pickle as pkl
pkl.dump(RF_clf,open("finali_model.p","wb"))
```