IAN DOUGLAS GACHUHI NJAI

P101/1364G/20

COM 404E – Project on Breast Cancer using ML

Dataset

https://www.kaggle.com/datasets/reihanenamdari/breast-cancer

In [ ]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

a. Load the Data

In [ ]:
```
# Load the breast cancer dataset
breast_cancer_data = pd.read_csv('Breast_Cancer.csv')
```

b. Display the Dataframe Information

In [ ]:
```
# Display the information about the dataset
print(breast_cancer_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4024 entries, 0 to 4023
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Age                     4024 non-null   int64 
 1   Race                    4024 non-null   object
 2   Marital Status          4024 non-null   object
 3   T Stage                 4024 non-null   object
 4   N Stage                 4024 non-null   object
 5   6th Stage               4024 non-null   object
 6   differentiate           4024 non-null   object
 7   Grade                   4024 non-null   object
 8   A Stage                 4024 non-null   object
 9   Tumor Size              4024 non-null   int64 
 10  Estrogen Status         4024 non-null   object
 11  Progesterone Status     4024 non-null   object
 12  Regional Node Examined  4024 non-null   int64 
 13  Reginol Node Positive   4024 non-null   int64 
 14  Survival Months         4024 non-null   int64 
 15  Status                  4024 non-null   object
dtypes: int64(5), object(11)
memory usage: 503.1+ KB
None
```

c. Display the first and last tuples of the data set

In [ ]:
```
# Display the first and last rows of the dataset
print("First 5 rows:")
breast_cancer_data.head()
```

```
First 5 rows:
```

Out[ ]:

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estrogen Status	I
0	68	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	4	Positive	
1	50	White	Married	T2	N2	IIIA	Moderately differentiated	2	Regional	35	Positive	
2	58	White	Divorced	T3	N3	IIIC	Moderately differentiated	2	Regional	63	Positive	
3	58	White	Married	T1	N1	IIA	Poorly differentiated	3	Regional	18	Positive	
4	47	White	Married	T2	N1	IIB	Poorly differentiated	3	Regional	41	Positive	

In [ ]:
```
print("Last 5 rows:")
breast_cancer_data.tail()
```

```
Last 5 rows:
```

Out[ ]:

	Age	Race	Marital Status	T Stage	N Stage	6th Stage	differentiate	Grade	A Stage	Tumor Size	Estroge Statu
4019	62	Other	Married	T1	N1	IIA	Moderately differentiated	2	Regional	9	Positiv
4020	56	White	Divorced	T2	N2	IIIA	Moderately differentiated	2	Regional	46	Positiv
4021	68	White	Married	T2	N1	IIB	Moderately differentiated	2	Regional	22	Positiv
4022	58	Black	Divorced	T2	N1	IIB	Moderately differentiated	2	Regional	44	Positiv
4023	46	White	Married	T2	N1	IIB	Moderately differentiated	2	Regional	30	Positiv

d. Display the descriptive statistics

In [ ]:

```
# Display descriptive statistics
breast_cancer_data.describe()
```

Out[ ]:

	Age	Tumor Size	Regional Node Examined	Reginol Node Positive	Survival Months
count	4024.000000	4024.000000	4024.000000	4024.000000	4024.000000
mean	53.972167	30.473658	14.357107	4.158052	71.297962
std	8.963134	21.119696	8.099675	5.109331	22.921430
min	30.000000	1.000000	1.000000	1.000000	1.000000
25%	47.000000	16.000000	9.000000	1.000000	56.000000
50%	54.000000	25.000000	14.000000	2.000000	73.000000
75%	61.000000	38.000000	19.000000	5.000000	90.000000
max	69.000000	140.000000	61.000000	46.000000	107.000000

e. Display the class label distribution

In [ ]:

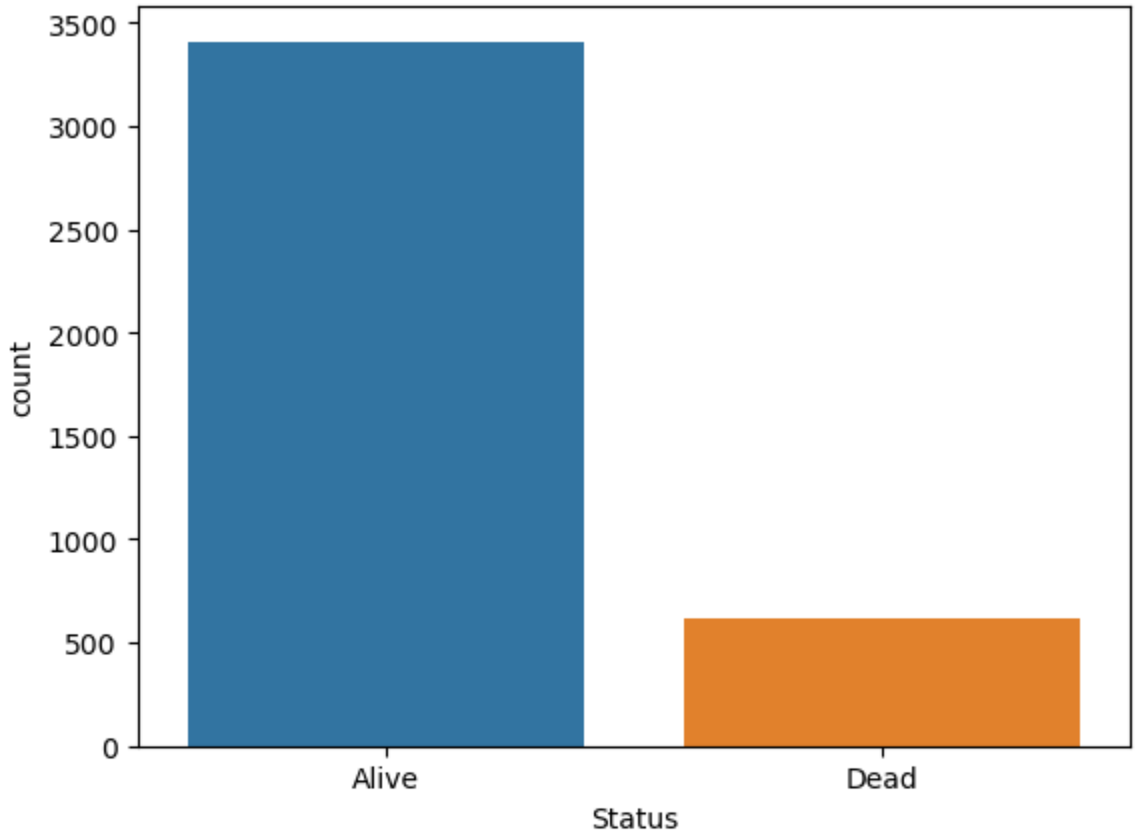
```
# Display the distribution of class labels
print("Class Label Distribution:")
print(breast_cancer_data['Status'].value_counts())
```

```
Class Label Distribution:
Status
Alive    3408
Dead      616
Name: count, dtype: int64
```

f. Display count plot for the class label

In [ ]:
```
# Count plot of the 'Status' label using seaborn
sns.countplot(x='Status', data=breast_cancer_data)
```

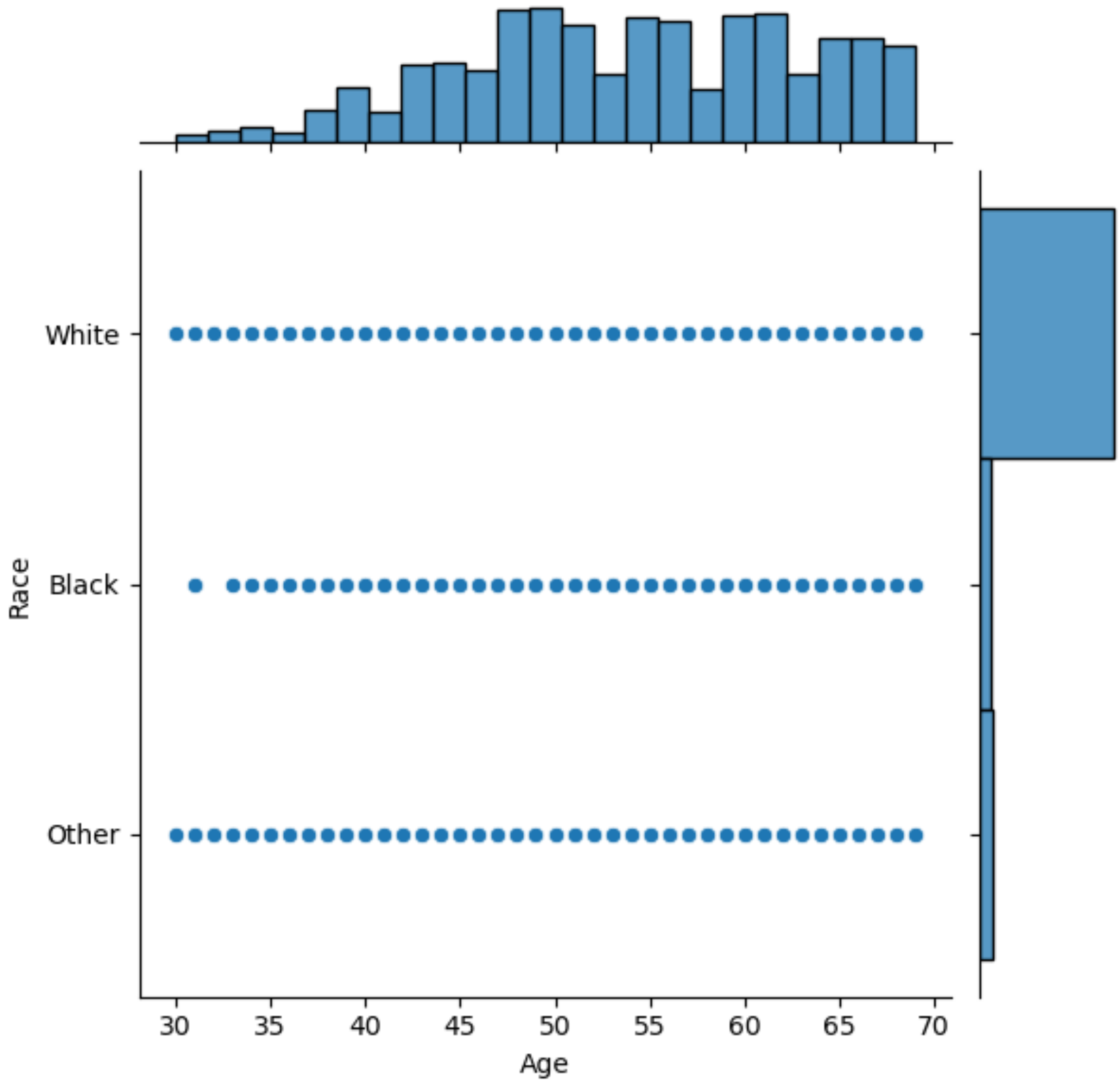
Out[ ]: <Axes: xlabel='Status', ylabel='count'>



g. Display a joint plot with any two variables of your choice

In [ ]:
```
# Display a joint plot with any two variables of your choice
sns.jointplot(x='Age', y='Race', data=breast_cancer_data)
```

Out[ ]: <seaborn.axisgrid.JointGrid at 0x7f641a575a10>



h. Determine based in the data set, whether you want to use category encoders

In [ ]:
```
# Checking data types for potential encoding
print("Data Types:")
print(breast_cancer_data.dtypes)
```

```
Data Types:
Age                          int64
Race                        object
Marital Status              object
T Stage                     object
N Stage                     object
6th Stage                   object
differentiate               object
Grade                       object
A Stage                     object
Tumor Size                   int64
Estrogen Status             object
Progesterone Status         object
Regional Node Examined       int64
Reginol Node Positive        int64
Survival Months              int64
Status                      object
dtype: object
```

In [ ]:

```python
# Encoding all categorical variables
label_encoder = LabelEncoder()

# Iterate through the variables in the dataframe for encoding
for col in breast_cancer_data.columns:
    breast_cancer_data[col] = label_encoder.fit_transform(breast_cancer_data
```

i. Split the data such that 25% is reserved for testing

In [ ]:

```python
# Split the data into training and testing sets
X = breast_cancer_data.drop('Status', axis=1)
y = breast_cancer_data['Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
```

j. show the shape of the training set and the test set

In [ ]:

```python
# Display the shape of the training and test sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (3018, 15) (3018,)
Test set shape: (1006, 15) (1006,)
```

k. Train a Model Using KNN

In [ ]:

```python
# Train a KNN model
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
```

Out[ ]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

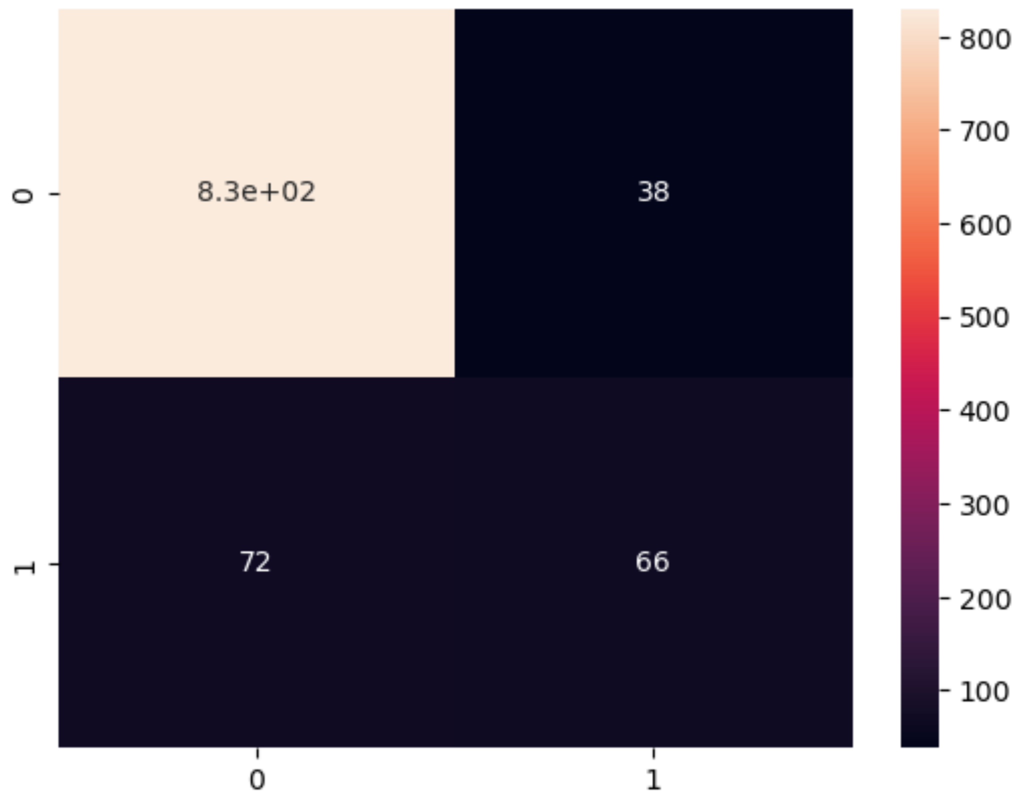
In [ ]:
```
# Predictions and Evaluation
y_pred_knn = knn_classifier.predict(X_test)

# Confusion Matrix
print("Confusion Matrix (KNN):")
print(confusion_matrix(y_test, y_pred_knn))
```

```
Confusion Matrix (KNN):
[[830  38]
 [ 72  66]]
```

In [ ]:
```
# Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True)
```

Out[ ]: <Axes: >



In [ ]:
```
# Classification Accuracy
print("Classification Accuracy (KNN):", accuracy_score(y_test, y_pred_knn))
```

```
Classification Accuracy (KNN): 0.8906560636182903
```

In [ ]:
```
# Training Accuracy
print("Training Accuracy (KNN):", knn_classifier.score(X_train, y_train))
```

```
Training Accuracy (KNN): 0.9135188866799204
```

I. Train a Model Using SVM

In [ ]:
```
# Train an SVM model
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC(kernel='linear')
```

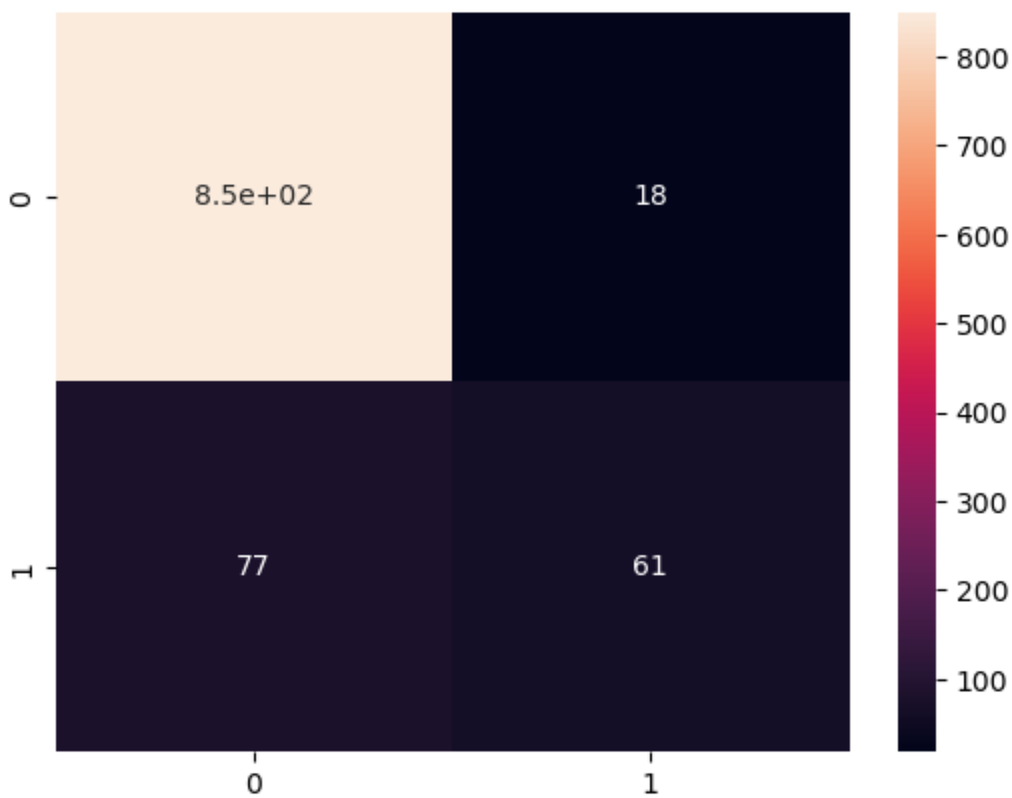
In [ ]:
```
# Predictions and Evaluation
y_pred_svm = svm_classifier.predict(X_test)

# Confusion Matrix
print("Confusion Matrix (SVM):")
print(confusion_matrix(y_test, y_pred_svm))
```

```
Confusion Matrix (SVM):
[[850  18]
 [ 77  61]]
```

In [ ]:
```
# Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True)
```

Out[ ]: <Axes: >



In [ ]:
```
# Classification Accuracy
print("Classification Accuracy (SVM):", accuracy_score(y_test, y_pred_svm))
```

```
Classification Accuracy (SVM): 0.9055666003976143
```

In [ ]:
```
# Training Accuracy
print("Training Accuracy (SVM):", svm_classifier.score(X_train, y_train))
```

```
Training Accuracy (SVM): 0.8933068257123923
```

m. Train a Model Using Decision Tree

In [ ]:
```
# Train a Decision Tree model
decision_tree_classifier = DecisionTreeClassifier()
decision_tree_classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

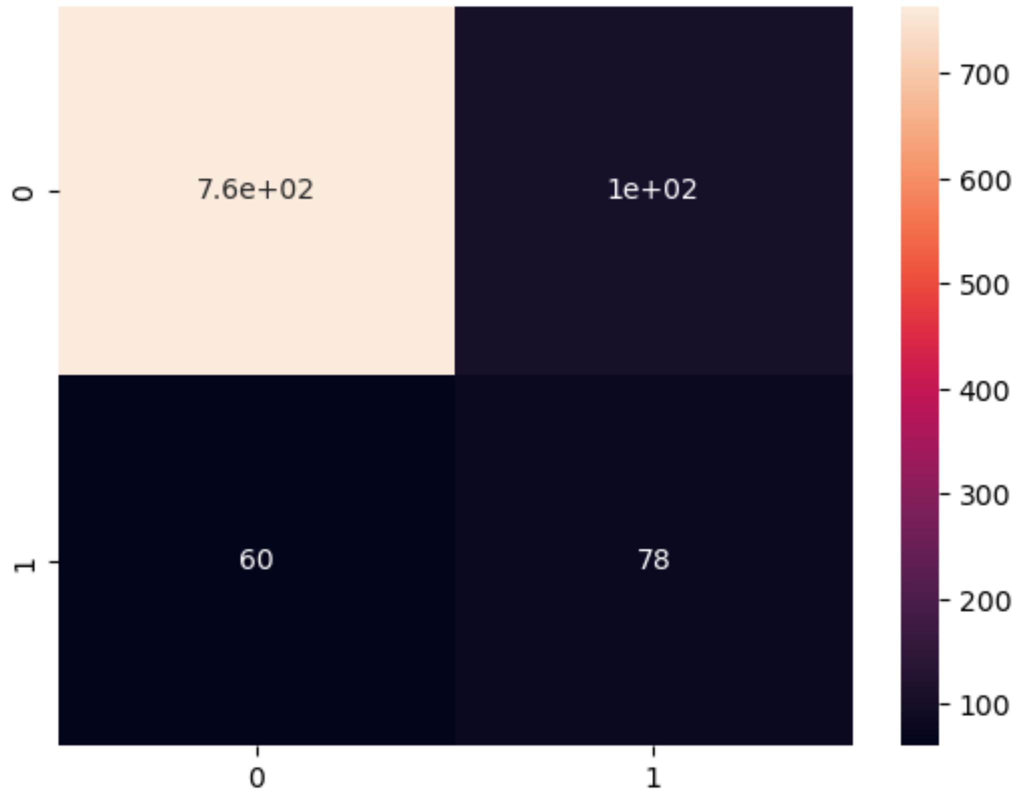
In [ ]:
```
# Predictions and Evaluation
y_pred_dt = decision_tree_classifier.predict(X_test)

# Confusion Matrix
print("Confusion Matrix (Decision Tree):")
print(confusion_matrix(y_test, y_pred_dt))
```

```
Confusion Matrix (Decision Tree):
[[764 104]
 [ 60  78]]
```

In [ ]:
```
# Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred_dt), annot=True)
```

Out[ ]: <Axes: >



In [ ]:
```
# Classification Accuracy
print("Classification Accuracy (Decision Tree):", accuracy_score(y_test, y_p
```

```
Classification Accuracy (Decision Tree): 0.8369781312127237
```

In [ ]:
```
# Training Accuracy
print("Training Accuracy (Decision Tree):", decision_tree_classifier.score(X
```

```
Training Accuracy (Decision Tree): 1.0
```

n. Train a Model Using Random Forest

In [ ]:
```
# Train a Random Forest model
random_forest_classifier = RandomForestClassifier()
random_forest_classifier.fit(X_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

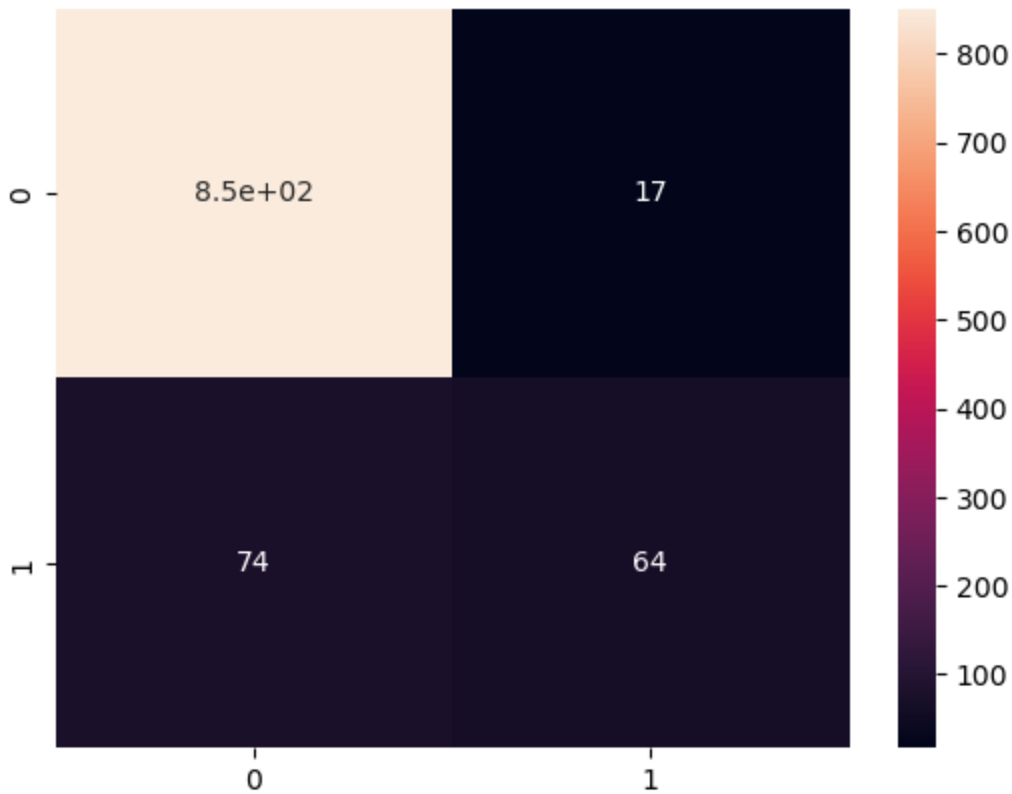
In [ ]:
```
# Predictions and Evaluation
y_pred_rf = random_forest_classifier.predict(X_test)

# Confusion Matrix
print("Confusion Matrix (Random Forest):")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Confusion Matrix (Random Forest):
[[851  17]
 [ 74  64]]
```

In [ ]:
```
# Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True)
```

Out[ ]:
```
<Axes: >
```



In [ ]:
```
# Classification Accuracy
print("Classification Accuracy (Random Forest):", accuracy_score(y_test, y_p
```

```
Classification Accuracy (Random Forest): 0.9095427435387674
```

In [ ]:
```
# Training Accuracy
print("Training Accuracy (Random Forest):", random_forest_classifier.score(X
```

```
Training Accuracy (Random Forest): 1.0
```

o. Demonstrate by a way of a plot which ML algorithm performs better from your results above

In [ ]:
```
# Create a list of models
models = ['KNN', 'SVM', 'Decision Tree', 'Random Forest']

# List of accuracy scores for each model
accuracy_scores = [
    knn_classifier.score(X_test, y_test),
    svm_classifier.score(X_test, y_test),
    decision_tree_classifier.score(X_test, y_test),
    random_forest_classifier.score(X_test, y_test)
]

# Print the accuracy scores
print("Accuracy Scores:")
print(accuracy_scores)
```

```
Accuracy Scores:
[0.8906560636182903, 0.9055666003976143, 0.8369781312127237, 0.909542743538
7674]
```

In [ ]:
```
# scatter plot of the accuracy scores
plt.scatter(models, accuracy_scores)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f6419178b50>
```