indian_liver_patient Dataset analysing and predictions

In [ ]:
```
#importing library
```

In [47]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [48]:
```
#Looking up data
df = pd.read_csv("indian_liver_patient.csv")
df.head()P
```

Out[48]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	A
0	65	Female	0.7	0.1	187	16	
1	62	Male	10.9	5.5	699	64	
2	62	Male	7.3	4.1	490	60	
3	58	Male	1.0	0.4	182	14	
4	72	Male	3.9	2.0	195	27	

In [49]:
```
# Looking for missing values in the dataset
df.isnull().sum()
```

Out[49]:
```
Age                         0
Gender                      0
Total_Bilirubin             0
Direct_Bilirubin            0
Alkaline_Phosphotase        0
Alamine_Aminotransferase    0
Aspartate_Aminotransferase  0
Total_Protiens              0
Albumin                     0
Albumin_and_Globulin_Ratio  4
Dataset                     0
dtype: int64
```

In [100…
```
df.shape
```

Out[100…
```
(579, 11)
```

.

.

.

.

In [52]:
```
df.dtypes
```

Out[52]:
```
Age                           int64
Gender                       object
Total_Bilirubin             float64
Direct_Bilirubin            float64
Alkaline_Phosphotase          int64
Alamine_Aminotransferase      int64
Aspartate_Aminotransferase    int64
Total_Protiens              float64
Albumin                     float64
Albumin_and_Globulin_Ratio  float64
Dataset                       int64
dtype: object
```

In [53]:
```
df.describe()
```

Out[53]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase
count	583.000000	583.000000	583.000000	583.000000	583.000000
mean	44.746141	3.298799	1.486106	290.576329	80.713551
std	16.189833	6.209522	2.808498	242.937989	182.620356
min	4.000000	0.400000	0.100000	63.000000	10.000000
25%	33.000000	0.800000	0.200000	175.500000	23.000000
50%	45.000000	1.000000	0.300000	208.000000	35.000000
75%	58.000000	2.600000	1.300000	298.000000	60.500000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000

In [55]:
```
# Re-naming the columns
df.rename(columns={'Dataset':'Outcome'}, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Gender                      583 non-null    object 
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64  
 5   Alamine_Aminotransferase    583 non-null    int64  
 6   Aspartate_Aminotransferase  583 non-null    int64  
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Outcome                     583 non-null    int64  
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

In [56]:
```
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 2})
```

.

In [57]:
```
# Having a look at the dataset after the numerical transformation
df.head()
```

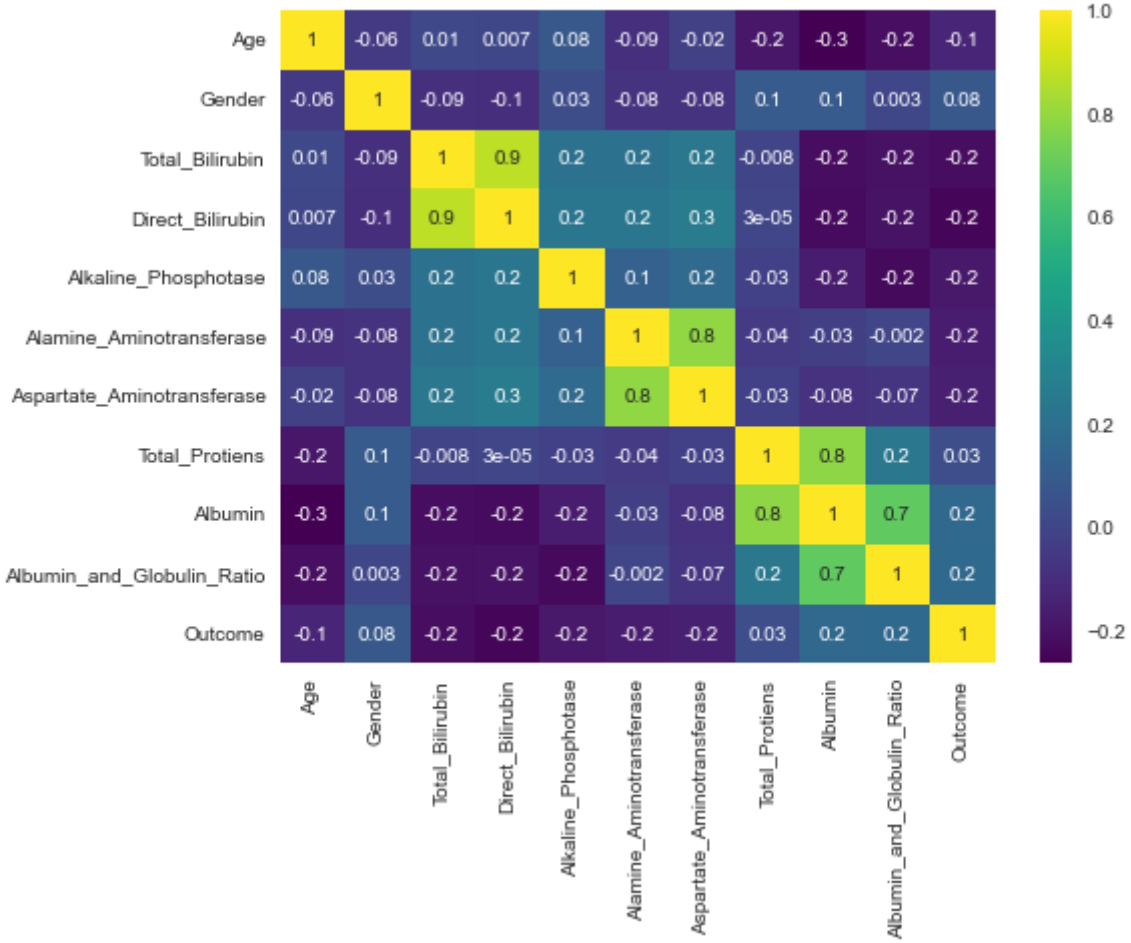
Out[57]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	A
0	65	2	0.7	0.1	187	16	
1	62	1	10.9	5.5	699	64	
2	62	1	7.3	4.1	490	60	
3	58	1	1.0	0.4	182	14	
4	72	1	3.9	2.0	195	27	

In [58]:
```
# Dropping the missing values
df = df.dropna()
```

In [99]:
```
# Having a look at the correlation matrix

fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, fmt='.1g', cmap="viridis");
```



In [60]:
```
print ('Total Unhealthy Livers : {} '.format(df.Outcome.value_counts()[1]))
print ('Total Healthy Livers : {} '.format(df.Outcome.value_counts()[2]))
```

```
Total Unhealthy Livers : 414
```

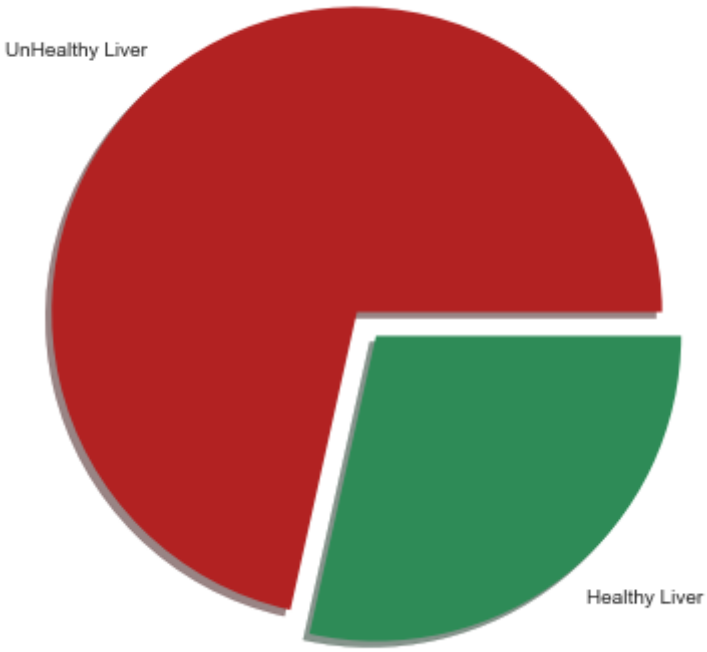
```
Total Healthy Livers : 165
```

In [61]:

```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(7,7))

plt.pie(x=df["Outcome"].value_counts(),
        colors=["firebrick","seagreen"],
        labels=["UnHealthy Liver","Healthy Liver"],
        shadow = True,
        explode = (0, 0.1)
        )

plt.show()
```



In [62]:

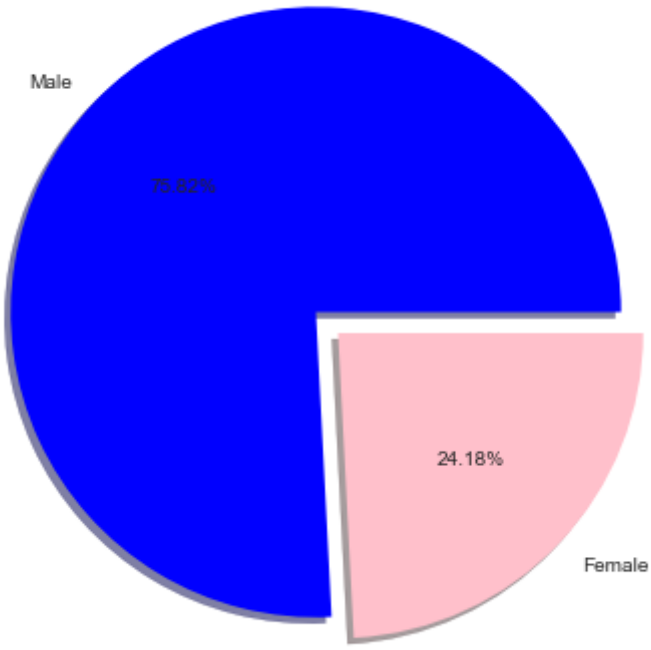
```
df.Gender.value_counts()
```

Out[62]:

```
1    439
2    140
Name: Gender, dtype: int64
```

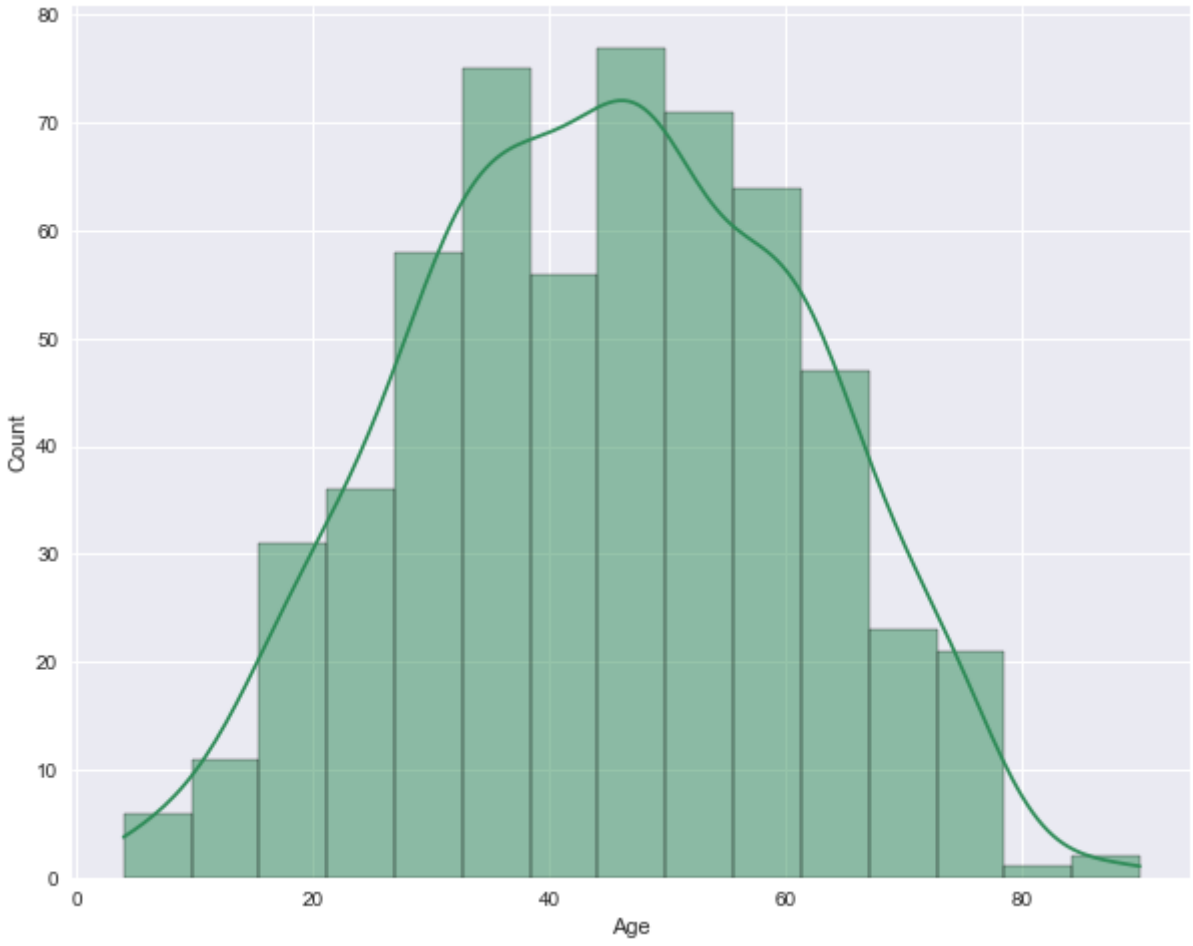
In [64]:

```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(7,7))
plt.pie(x=df["Gender"].value_counts(),
        colors=["blue","pink"],
        labels=["Male","Female"],
        shadow = True,
        autopct="%1.2f%%",
        explode = (0, 0.1)
        )
plt.show()
```



In [103…

```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(10,8))
sns.histplot(x=df["Age"], kde=True, color="seagreen");
```



In [101…

```
#Splitting the data into training and test datasets
# y data
y = df["Outcome"]
```

```
# X data
X = df.drop("Outcome", axis=1)
X.head()
```

Out[101…

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	A
0	65	2	0.7	0.1	187	16	
1	62	1	10.9	5.5	699	64	
2	62	1	7.3	4.1	490	60	
3	58	1	1.0	0.4	182	14	
4	72	1	3.9	2.0	195	27	

In [94]:
```
y.head()
```

Out[94]:
```
0    1
1    1
2    1
3    1
4    1
Name: Outcome, dtype: int64
```

In [71]:
```
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_stat
```

In [72]:
```
len(X_train), len(X_test)
```

Out[72]:
```
(463, 116)
```

In [73]:
```
# Scaling the data

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic regression

In [74]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[74]:
```
LogisticRegression()
```

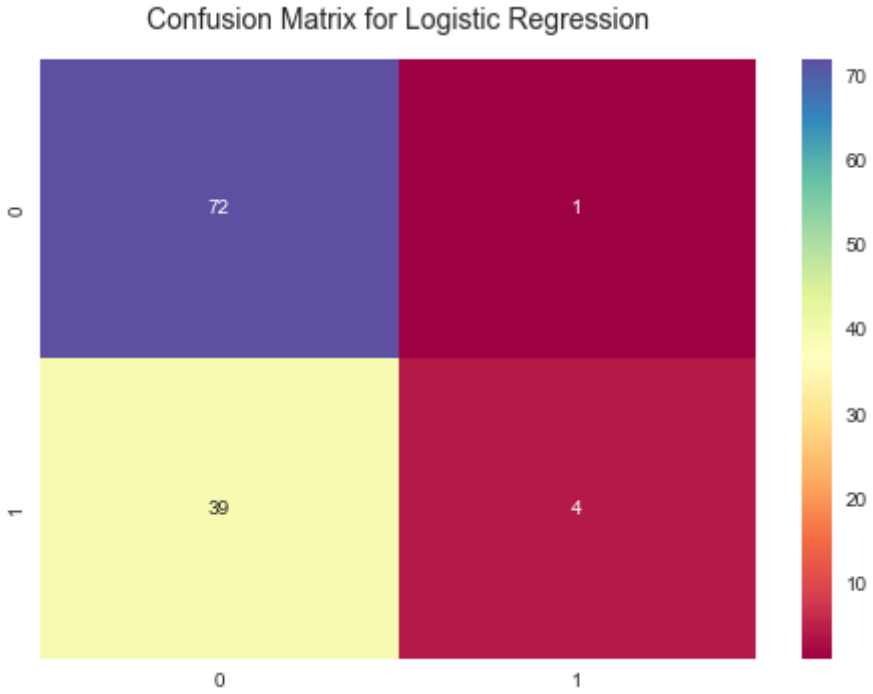
In [75]:
```
LogisticRegressionScore = lr.score(X_test, y_test)
print("Accuracy obtained by Logistic Regression model:",LogisticRegressionScore*100)
```

```
Accuracy obtained by Logistic Regression model: 65.51724137931035
```

In [77]:
```
# Having a look at the confusion matrix
```

```
from sklearn.metrics import confusion_matrix, classification_report

y_pred_lr = lr.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cf_matrix, annot=True, cmap="Spectral")
plt.title("Confusion Matrix for Logistic Regression", fontsize=14,y=1.03);
```



In [78]:
```
# Having a look at the classification report of Logistic Regression

from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

           1       0.65      0.99      0.78        73
           2       0.80      0.09      0.17        43

    accuracy                           0.66       116
   macro avg       0.72      0.54      0.47       116
weighted avg       0.70      0.66      0.55       116
```

K Neighbors Classifier

In [95]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(3)
knn.fit(X_train,y_train)
```

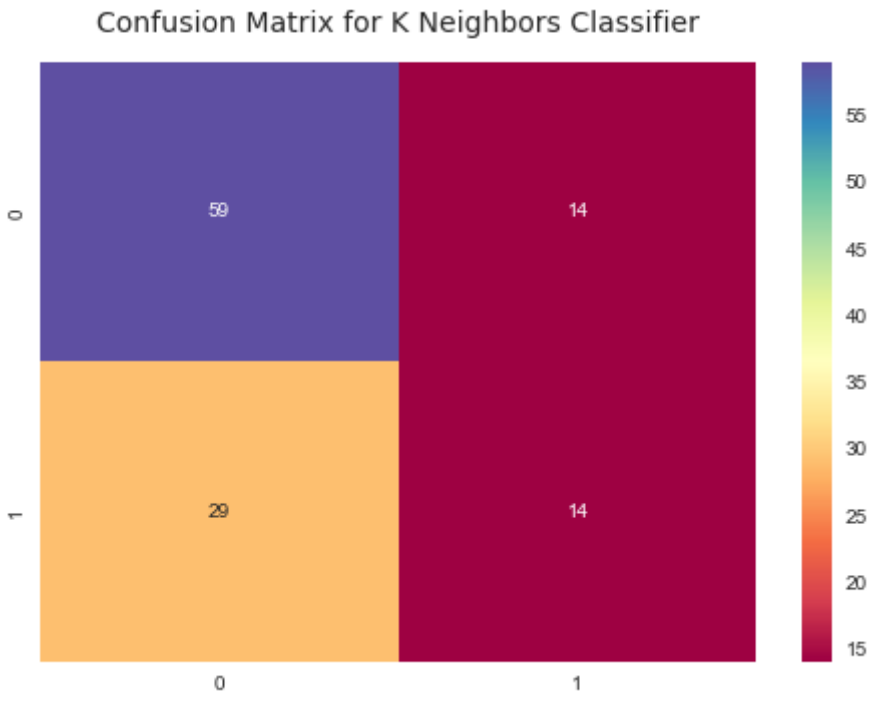
Out[95]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [96]:
```
KNeighborsClassifierScore = knn.score(X_test, y_test)
print("Accuracy obtained by K Neighbors Classifier model:",KNeighborsClassifierScore
```

```
Accuracy obtained by K Neighbors Classifier model: 62.93103448275862
```

In [97]:
```
# Having a look at the confusion matrix
y_pred_knn = knn.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cf_matrix, annot=True, cmap="Spectral")
plt.title("Confusion Matrix for K Neighbors Classifier", fontsize=14, fontname="Helv
```

Out[97]:
```
Text(0.5, 1.03, 'Confusion Matrix for K Neighbors Classifier')
```



In [98]:
```
# Classification report of K Neighbors Classifier

print(metrics.classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           1       0.67      0.81      0.73        73
           2       0.50      0.33      0.39        43

    accuracy                           0.63       116
   macro avg       0.59      0.57      0.56       116
weighted avg       0.61      0.63      0.61       116
```

DecisionTreeClassifier

In [84]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

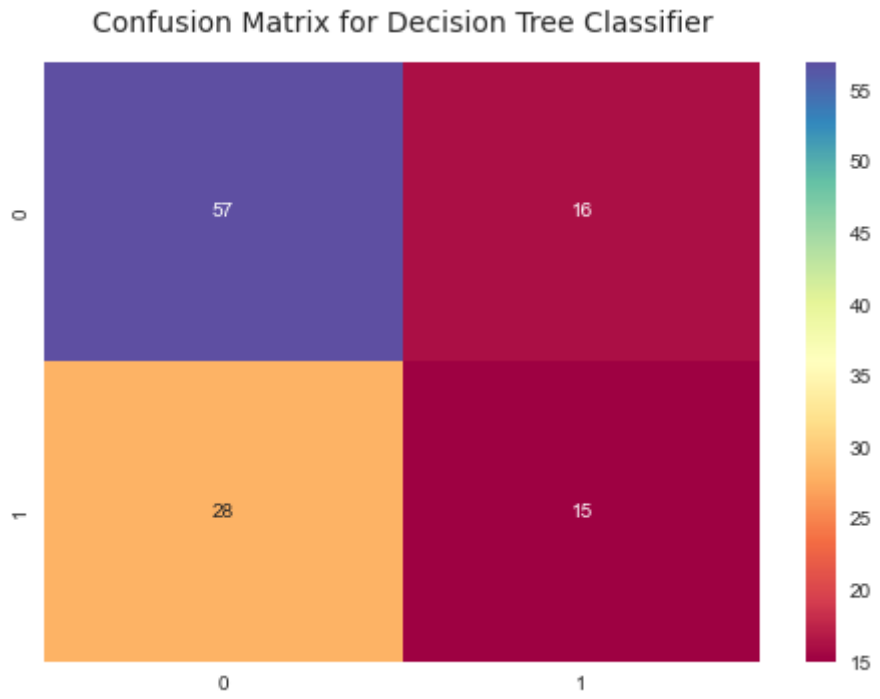
Out[84]:
```
DecisionTreeClassifier()
```

In [85]:
```
DecisionTreeClassifierScore = dtc.score(X_test,y_test)
print("Accuracy obtained by Decision Tree Classifier model:",DecisionTreeClassifierS
```

```
Accuracy obtained by Decision Tree Classifier model: 62.06896551724138
```

In [86]:
```
# Confusion matrix
```

```
y_pred_dtc = dtc.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_dtc)
sns.heatmap(cf_matrix, annot=True, cmap="Spectral")
plt.title("Confusion Matrix for Decision Tree Classifier", fontsize=14, fontname="He
```



In [87]:

```
# Classification Report of Decision Tree Classifier

print(metrics.classification_report(y_test, y_pred_dtc))
```

```
              precision    recall  f1-score   support

           1       0.67      0.78      0.72        73
           2       0.48      0.35      0.41        43

    accuracy                           0.62       116
   macro avg       0.58      0.56      0.56       116
weighted avg       0.60      0.62      0.60       116
```

In [ ]: