Regression

In [31]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:
```
df=pd.read_csv('Student_Performance.csv')
df.head()
```

Out[2]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0

Preprocessing

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
Hours Studied                       0
Previous Scores                     0
Extracurricular Activities          0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
dtype: int64
```

In [4]:
```
df.duplicated().sum()
```

Out[4]:
```
127
```

In [5]:
```
df.dtypes
```

Out[5]:
```
Hours Studied                         int64
Previous Scores                       int64
Extracurricular Activities           object
Sleep Hours                           int64
Sample Question Papers Practiced      int64
Performance Index                   float64
dtype: object
```

In [6]:
```
df['Extracurricular Activities'].unique()
```

Out[6]:
```
array(['Yes', 'No'], dtype=object)
```

In [7]:
```
ohe = OneHotEncoder(sparse_output=False, drop='first')
encoded=ohe.fit_transform(df[['Extracurricular Activities']])
ohe.categories_
```

Out[7]:
```
[array(['No', 'Yes'], dtype=object)]
```

In [8]:
```
encoded
```

Out[8]:
```
array([[1.],
       [0.],
       [1.],
       ...,
       [1.],
       [1.],
       [0.]])
```

In [9]:
```
encoded_df = pd.DataFrame(encoded, columns=ohe.get_feature_names_out(['Extracurricu
encoded_df= encoded_df.astype(float)
encoded_df.head()
```

Out[9]:

	Extracurricular Activities_Yes
0	1.0
1	0.0
2	1.0
3	1.0
4	0.0

concatenation

In [10]:
```
df = pd.concat([df, encoded_df], axis=1)
df.head()
```

Out[10]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_Yes
0	7	99	Yes	9	1	91.0	1.0
1	4	82	No	4	2	65.0	0.0
2	8	51	Yes	7	2	45.0	1.0
3	5	52	Yes	5	2	36.0	1.0
4	7	75	No	8	5	66.0	0.0

In [11]:
```
df.drop('Extracurricular Activities',axis=1,inplace=True)
df.isnull().sum()
```

Out[11]:
```
Hours Studied                       0
Previous Scores                     0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
Extracurricular Activities_Yes      0
dtype: int64
```

In [12]:
```
df.describe()
```

Out[12]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_Yes
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	4.992900	69.445700	6.530600	4.583300	55.224800	0.494800
std	2.589309	17.343152	1.695863	2.867348	19.212558	0.499998
min	1.000000	40.000000	4.000000	0.000000	10.000000	0.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000	0.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000	0.000000
75%	7.000000	85.000000	8.000000	7.000000	71.000000	1.000000
max	9.000000	99.000000	9.000000	9.000000	100.000000	1.000000

correlation

In [13]:
```
df.corr()
```

Out[13]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_Yes
Hours Studied	1.000000	-0.012390	0.001245	0.017463	0.373730	0.003873
Previous Scores	-0.012390	1.000000	0.005944	0.007888	0.915189	0.008369
Sleep Hours	0.001245	0.005944	1.000000	0.003990	0.048106	-0.023284
Sample Question Papers Practiced	0.017463	0.007888	0.003990	1.000000	0.043268	0.013103
Performance Index	0.373730	0.915189	0.048106	0.043268	1.000000	0.024525
Extracurricular Activities_Yes	0.003873	0.008369	-0.023284	0.013103	0.024525	1.000000

linear regression model

In [14]:
```
X = df.drop('Performance Index',axis=1)
y = df['Performance Index']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

In [15]:
```
model= LinearRegression()
model.fit(X_train,y_train)
```

Out[15]:
```
▾ LinearRegression
LinearRegression()
```

In [16]:
```
# Perform 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_mse = -cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=kf)
cv_r2 = cross_val_score(model, X, y, scoring='r2', cv=kf)
```

mse & r2

In [17]:
```
y_predict = model.predict(X_test)
mse = mean_squared_error(y_test, y_predict)
r2 = r2_score(y_test, y_predict)
mse, r2
```

Out[17]:
```
(4.224022760753756, 0.9884855999665682)
```

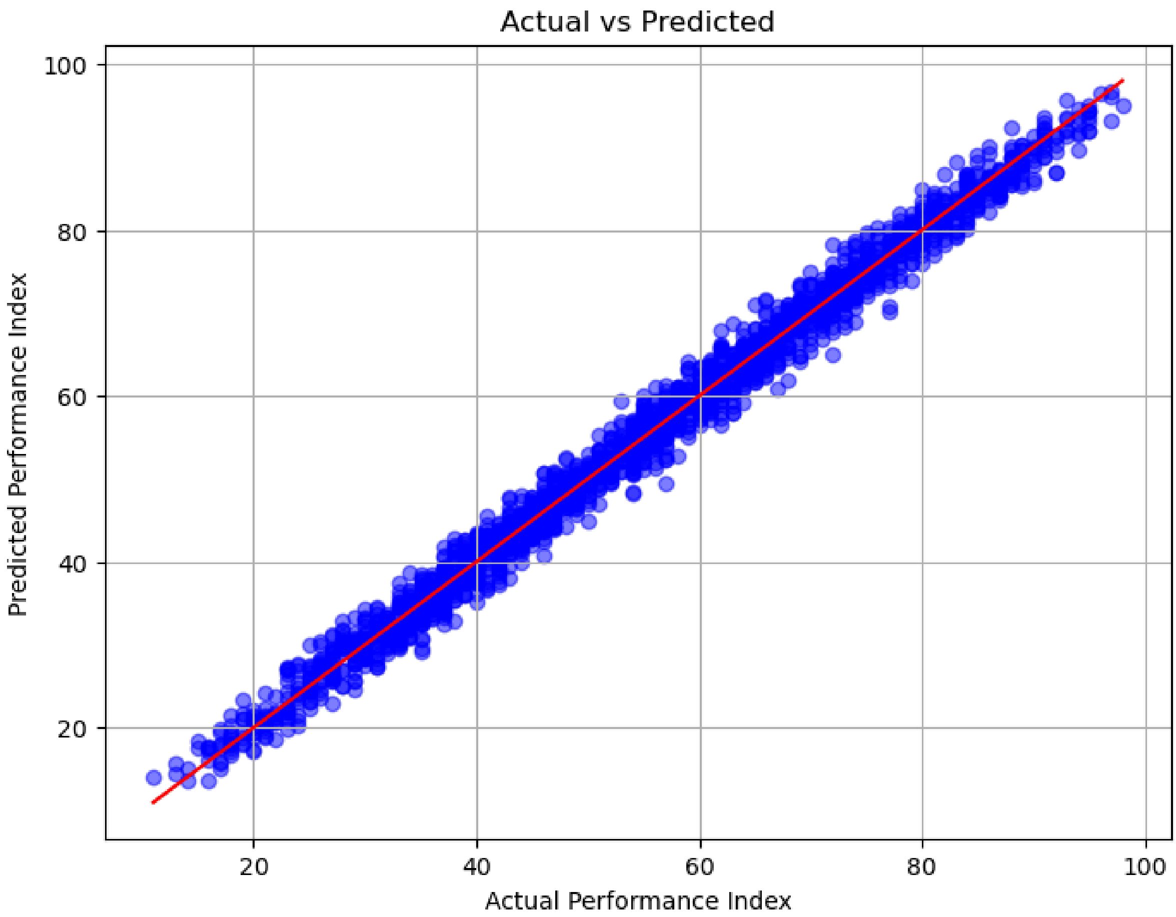
accuracy score

In [18]:
```
r2Score=r2_score(y_test,y_predict)
mse = mean_squared_error(y_test, y_predict)
print("Accuracy_R\u00b2 : ",r2Score)
print("Accuracy_MSE : ",mse)
```

```
Accuracy_R² :  0.9884855999665682
Accuracy_MSE :  4.224022760753756
```

regression plot

In [19]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_predict,marker='o' ,color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lines
plt.title('Actual vs Predicted')
plt.xlabel('Actual Performance Index')
plt.ylabel('Predicted Performance Index')
plt.grid(True)
plt.show()
```



Classification

In [39]:
```
# Importing the required libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import seaborn as sns
```

In [40]:
```
cancer = load_breast_cancer()
X_cancer, y_cancer = cancer.data, cancer.target
```

In [44]:
```
#Split the dataset into training and test sets
X_train_cancer, X_test_cancer, y_train_cancer, y_test_cancer = train_test_split(X_c
```

In [46]:
```
#implementing naive bayes
nb= GaussianNB()
nb.fit(X_train_cancer, y_train_cancer)
```

Out[46]:
```
▾ GaussianNB
GaussianNB()
```

In [47]:
```
# print the accuracy
gnb_pred= nb.predict(X_test_cancer)
print("Accuracy of Gaussian Naive Bayes: ",
```

```
      accuracy_score(y_test_cancer, gnb_pred))
# print other performance metrics
print("Precision of Gaussian Naive Bayes: ",
      precision_score(y_test_cancer, gnb_pred, average='weighted'))
print("Recall of Gaussian Naive Bayes: ",
      recall_score(y_test_cancer, gnb_pred, average='weighted'))
print("F1-Score of Gaussian Naive Bayes: ",
      f1_score(y_test_cancer, gnb_pred, average='weighted'))
```

```
Accuracy of Gaussian Naive Bayes:  0.9415204678362573
Precision of Gaussian Naive Bayes:  0.941391481684838
Recall of Gaussian Naive Bayes:  0.9415204678362573
F1-Score of Gaussian Naive Bayes:  0.9413171134406007
```

In [48]:

```
# DECISION TREE CLASSIFIER
dt= DecisionTreeClassifier(random_state=0)
# train the model
dt.fit(X_train_cancer, y_train_cancer)
# make predictions
dt_pred= dt.predict(X_test_cancer)
```

In [49]:

```
# print the accuracy
print("Accuracy of Decision Tree Classifier: ",
      accuracy_score(y_test_cancer, dt_pred))
# print other performance metrics
print("Precision of Decision Tree Classifier: ",
      precision_score(y_test_cancer, dt_pred, average='weighted'))
print("Recall of Decision Tree Classifier: ",
      recall_score(y_test_cancer, dt_pred, average='weighted'))
print("F1-Score of Decision Tree Classifier: ",
      f1_score(y_test_cancer, dt_pred, average='weighted'))
```

```
Accuracy of Decision Tree Classifier:  0.9239766081871345
Precision of Decision Tree Classifier:  0.928608650338718
Recall of Decision Tree Classifier:  0.9239766081871345
F1-Score of Decision Tree Classifier:  0.9247041047595065
```

In [50]:

```
# Perform 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_mse = -cross_val_score(model, X_cancer, y_cancer, scoring='neg_mean_squared_erro
cv_r2 = cross_val_score(model, X_cancer, y_cancer, scoring='r2', cv=kf)
```

In [51]:

```
# Evaluation metrics
gnb_conf_matrix = confusion_matrix(y_test_cancer, gnb_pred)
dt_conf_matrix = confusion_matrix(y_test_cancer, dt_pred)
```

In [52]:

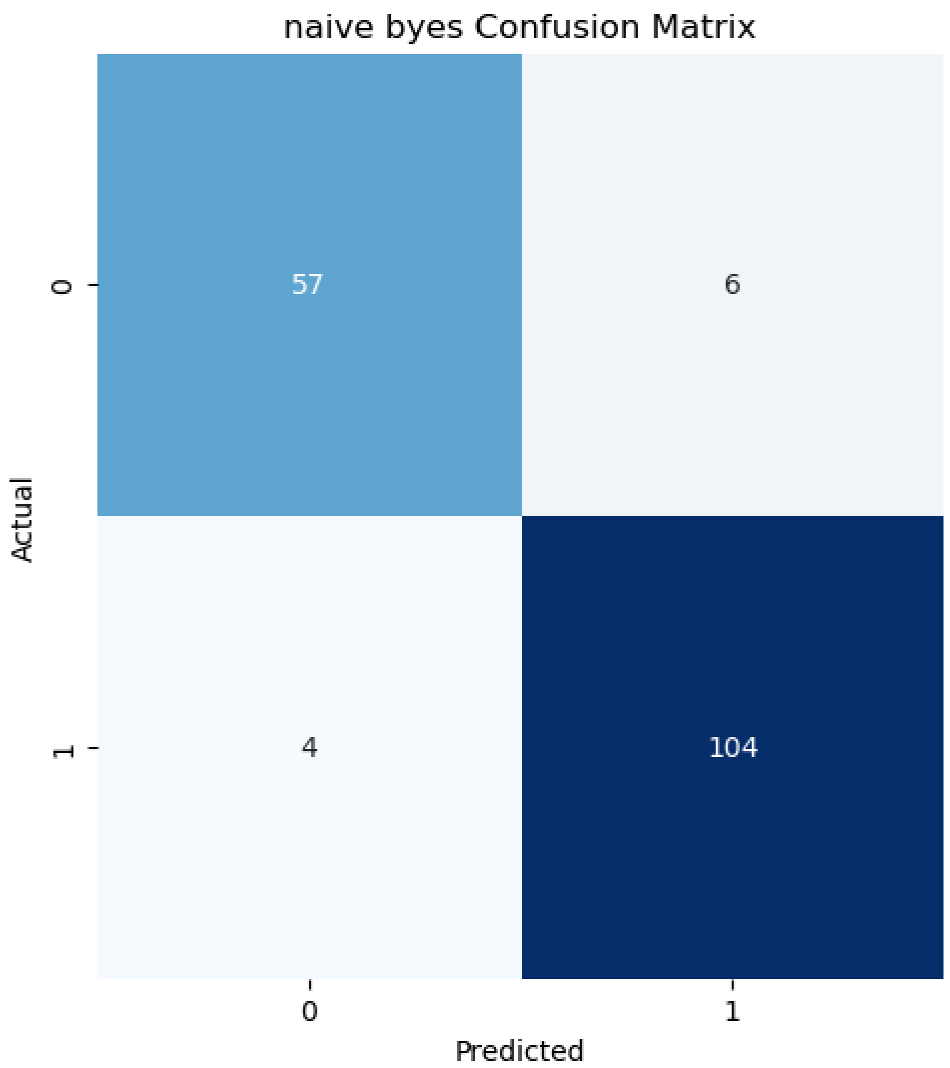
```
def evaluate_classification(y_true, y_pred):
    cm= confusion_matrix(y_true, y_pred)
    tn= cm[0, 0]
    fp= cm[0, 1]
    fn= cm[1, 0]
    tp= cm[1, 1]
    accuracy= accuracy_score(y_true, y_pred)
    specificity= tn/(tn+fp) if (tn+fp)>0 else 0
    error_rate= 1-accuracy
    print("Confusion Matrix:\n", cm)
    print(f"Error Rate: {error_rate:.4f}")
    print(f"Specificity: {specificity:.4f}")
    print(f"True Positives (TP): {tp}")
    print(f"True Negatives (TN): {tn}")
    print(f"False Positives (FP): {fp}")
    print(f"False Negatives (FN): {fn}")
```

In [53]:
```
print("\nNaive byes Evaluation:")
evaluate_classification(y_test_cancer, gnb_pred)
```

```
Naive byes Evaluation:
Confusion Matrix:
 [[ 57   6]
 [  4 104]]
Error Rate: 0.0585
Specificity: 0.9048
True Positives (TP): 104
True Negatives (TN): 57
False Positives (FP): 6
False Negatives (FN): 4
```

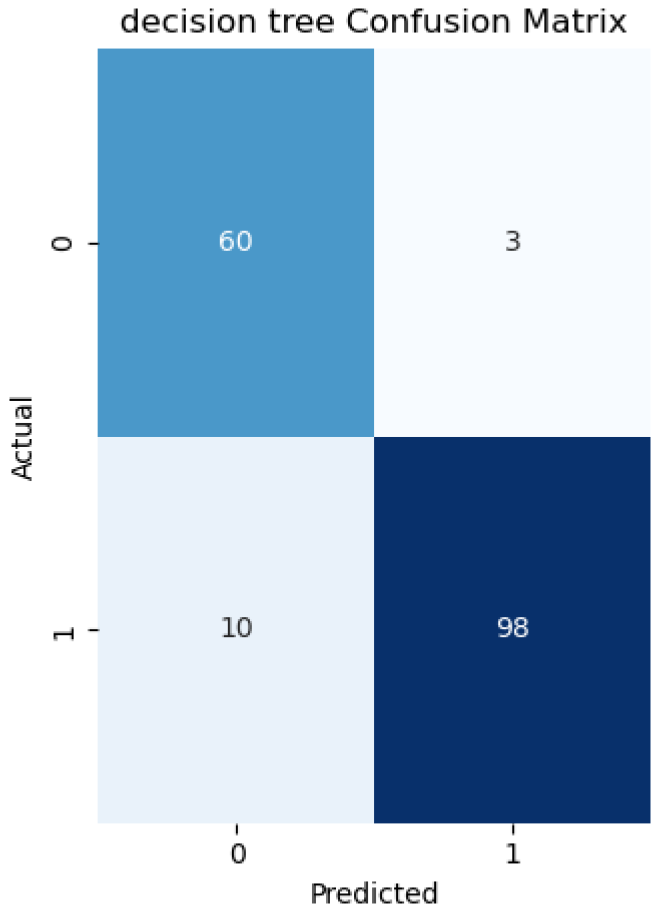
In [54]:
```
# Visualize Confusion Matrices
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.heatmap(gnb_conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("naive byes Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

Out[54]:
```
Text(120.72222222222221, 0.5, 'Actual')
```



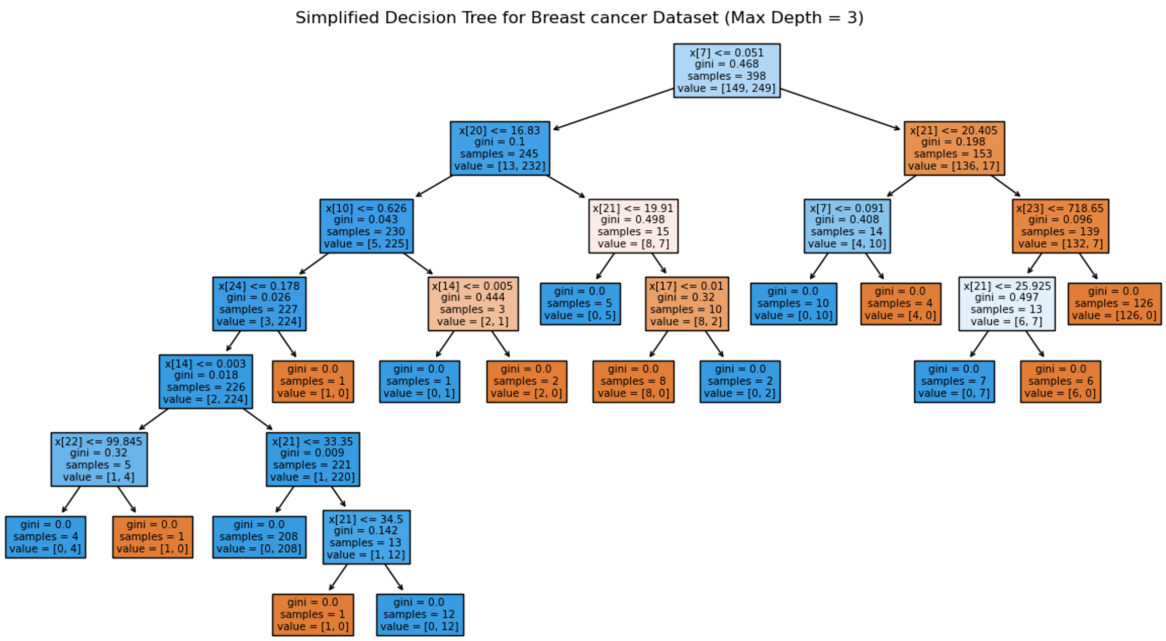
In [55]:
```
plt.subplot(1, 2, 2)
sns.heatmap(dt_conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title("decision tree Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```python
plt.tight_layout()
plt.show()
```



In [56]:
```python
# Importing the required libraries for plotting the decision tree
from sklearn.tree import plot_tree

# Plotting the decision tree
plt.figure(figsize=(15, 8))
plot_tree(dt, filled=True)
plt.title("Simplified Decision Tree for Breast cancer Dataset (Max Depth = 3)")
plt.show()
```



In [ ]:

```
```