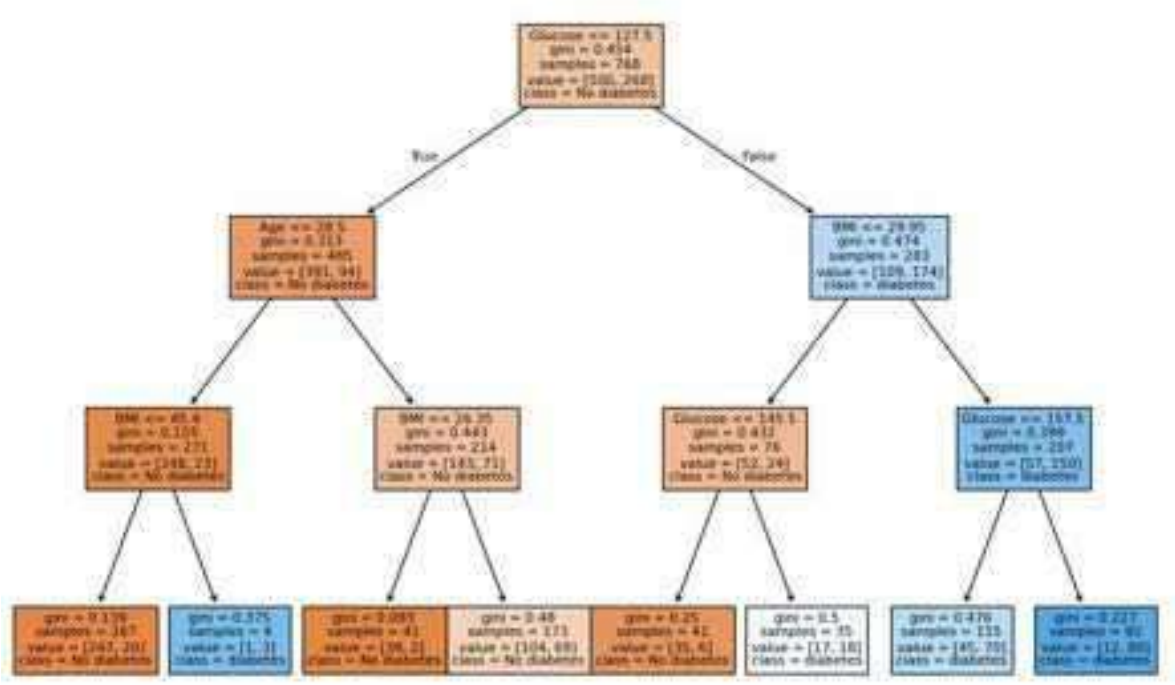
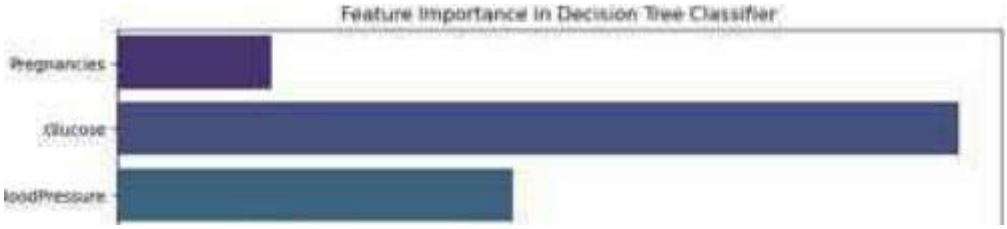
In [1]:
```
# 1
# Prepare decision tree classifier in python using sklearn Library for data se
# diabetes.csv - (pregnant, glucose, bp, skin, insulin, bmi, pedigree, age, La
# Use all features except Label as independent variable.
# Use complete dataset for training and Limit the depth of tree upto 3 Levels
# tree.plotTree()method.
import pandas as pd
from sklearn.tree import DecisionTreeClassifier,plot_tree
import matplotlib.pyplot as plt

data=pd.read_csv('Diabetes.csv')
x=data.drop(columns='Outcome')
y=data['Outcome']

clf=DecisionTreeClassifier(max_depth=3,random_state=42)
elf.fit(x,y)

plt.figure(figsize=(12,8))
plot_tree(clf,feature_names=x.columns,class_names=['No  diabetes','diabetes'J,f
plt.show()
```



In [2]:

```
# 2
# Using the diabetes dataset, implement a Decision Tree Classifier and
# determine the importance of each feature in predicting diabetes.
# Generate different plots to justify your model.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, accuracy_score, confusion_rr

# Load the dataset
df = pd.read_csv("Diabetes.csv")

# Split dataset into features and target variable
X = df.drop(columns=["Outcome"])
y = df["Outcome"]

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

#Traina Decision Tree Classifier
elf= DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)

# Model performance
accuracy= accuracy_score(y_test, y_pred)
report= classification_report(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)

# Feature Importance Plot
plt.figure(figsize=(10,  6))
sns.barplot(x=clf.feature_importances_, y=X.columns, palette="viridis")
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance in Decision Tree Classifier")
plt.show()

# Confusion Matrix
plt.figure(figsize=(6,  4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Visualizing the Decision Tree
plt.figure(figsize=(12,  8))
plot_tree(clf, feature_names=X.columns, class_names=["No Diabetes", "Diabetes"
plt.title("Decision Tree Visualization")
plt.show()
```

```
Model Accuracy: 0.75

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.76      0.79        99
           1       0.62      0.73      0.67        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154
```



In [3]:
```
#3
# Implement a Decision Tree Classifier and compare its performance with a SVM
# Display the accuracy of both models.
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification r

# Load the Iris dataset
iris= datasets.load_iris()
X, y = iris.data, iris.target

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

#Traina Decision Tree Classifier
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)
y_pred_dt = dt_clf.predict(X_test)
dt_accuracy  = accuracy_score(y_test, y_pred_dt)

# Train an SVM Classifier
svm_clf = SVC(kernel="linear", random_state=42)
svm_clf.fit(X_train, y_train)
y_pred_svm = svm_clf.predict(X_test)
svm_accuracy = accuracy_score(y_test, y_pred_svm)

# Print Accuracy
print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")
print(f"SVM Accuracy: {svm_accuracy:.2f}")
```

```
Decision Tree Accuracy: 1.00
SVM Accuracy: 1.00
```

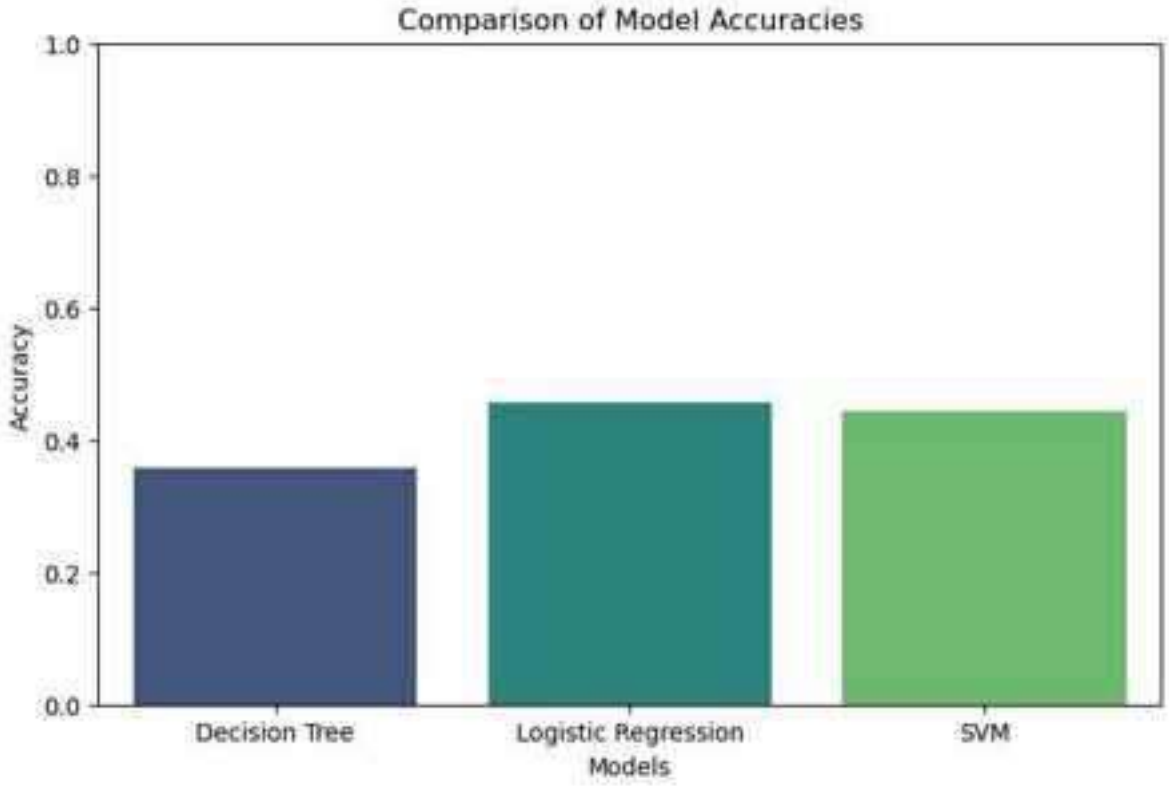
In [4]:
```
# 4
# Build decision tree classifier for iris data set. One with maximum Leaf node
# another one with minimum sample per Leaf as 5. Compare accuracy of both mode

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Load the Iris dataset
iris= load_iris()
X =iris.data# Features (sepal Length, sepal width, petal Length, petal width
y =iris.target# Target (setosa, versicolor, virginica)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randc
# 1. Create the first Decision Tree Classifier with a maximum of 8 Leaf nodes
dtc_max_leaf = DecisionTreeClassifier(max_leaf_nodes=8, random_state=42)
dtc_max_leaf.fit(X_train, y_train)

# 2. Create the second Decision Tree Classifier with a minimum of 5 samples pe
dtc_min_samples_leaf = DecisionTreeClassifier(min_samples_leaf=5, random_state
dtc_min_samples_leaf.fit(X_train, y_train)

# Evaluate the models using accuracy score
y_pred_max_leaf = dtc_max_leaf.predict(X_test)
y_pred_min_samples_leaf = dtc_min_samples_leaf.predict(X_test)
accuracy_max_leaf = accuracy_score(y_test, y_pred_max_leaf)
accuracy_min_samples_leaf = accuracy_score(y_test, y_pred_min_samples_leaf)

# Visualize the first decision tree (Max Leaf Nodes= 8)
plt.figure(figsize=(12, 8))
plot_tree(dtc_max_leaf, filled=True, feature_names=iris.feature_names, class n
plt.title("Decision Tree Classifier (Max Leaf Nodes= 8)")
plt.show()

# Visualize the second decision tree (Min Samples Per Leaf= 5)
plt.figure(figsize=(12, 8))
plot_tree(dtc_min_samples_leaf, filled=True, feature_names=iris.feature_names,
plt.title("Decision Tree Classifier (Min Samples Per Leaf= 5)")
plt.show()
```

In [5]:
```
#5
# Implement a Decision Tree Classifier and compare its performance with a
# Logistic Classifier on the diabetes dataset. Develop Python code to display
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
df = pd.read_csv("Diabetes.csv")

# Split dataset into features and target variable
X = df.drop(columns=["Outcome"])
y = df["Outcome"]

# Split data into training and testing sets (80% train 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

#Traina Decision Tree Classifier
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)
y_pred_dt = dt_clf.predict(X_test)
dt_accuracy = accuracy_score(y_test, y_pred_dt)

#Traina Logistic Regression Classifier
lr_clf = LogisticRegression(max_iter=1000, random_state=42)
lr_clf.fit(X_train, y_train)
y_pred_lr = lr_clf.predict(X_test)
lr_accuracy = accuracy_score(y_test, y_pred_lr)

# Print Accuracy
print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")
print(f"Logistic Regression Accuracy: {lr_accuracy:.2f}")
```

```
Decision Tree Accuracy: 0.75
Logistic Regression Accuracy: 0.75
```

In [6]:
```
#6
# Develop a random dataset using 3 columns and 350  rows.
# Generate the data frame and apply decision tree classifier to train the mode
# Generate plot of comparing the accuracy for different models.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Generate random dataset
np.random.seed(42)
data= {
    ''Featurel": np.random.rand(350) * 10, # Random values between 0 and 10
    ''Feature2": np.random.rand(350) * 20, # Random values between 0 and 20
    "Feature3": np.random.rand(350) * 30, # Random values between 0 and 30
    "Target": np.random.choice([0, 1], size=350)  # Binary classification
}

# Create DataFrame
df = pd.DataFrame(data)

# Split dataset into features and target variable
X = df.drop(columns=["Target"])
y = df["Target"]

# Split data into training and testing sets (80% trainJ 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

#Traina Decision Tree Classifier
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)
y_pred_dt = dt_clf.predict(X_test)
dt_accuracy  =  accuracy_score(y_test, y_pred_dt)

#Traina Logistic Regression Classifier
lr_clf = LogisticRegression(max_iter=1000, random_state=42)
lr_clf.fit(X_train, y_train)
y_pred_lr = lr_clf.predict(X_test)
lr_accuracy  =  accuracy_score(y_test, y_pred_lr)

# Train an SVM Classifier
svm_clf = SVC(kernel="linear", random_state=42)
svm_clf.fit(X_train, y_train)
y_pred_svm = svm_clf.predict(X_test)
svm_accuracy = accuracy_score(y_test, y_pred_svm)

# Compare Accuracies
model_names = ["Decision Tree", "Logistic Regression", "SVM"]
accuracies= [dt_accuracy, lr_accuracy, svm_accuracy]

# Plot accuracy comparison
plt.figure(figsize=(8,  5))
```

```
sns.barplot(x=model_names, y=accuracies, palette=''viridis")
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.ylim(0, 1) # Accuracy ranges between e and 1
plt.title("Comparison of Model Accuracies")
plt.show()

# Print accuracy scores
print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")
print(f"Logistic Regression  Accuracy: {lr_accuracy:.2f}")
print(f"SVM Accuracy: {svm_accuracy:.2f}")
```



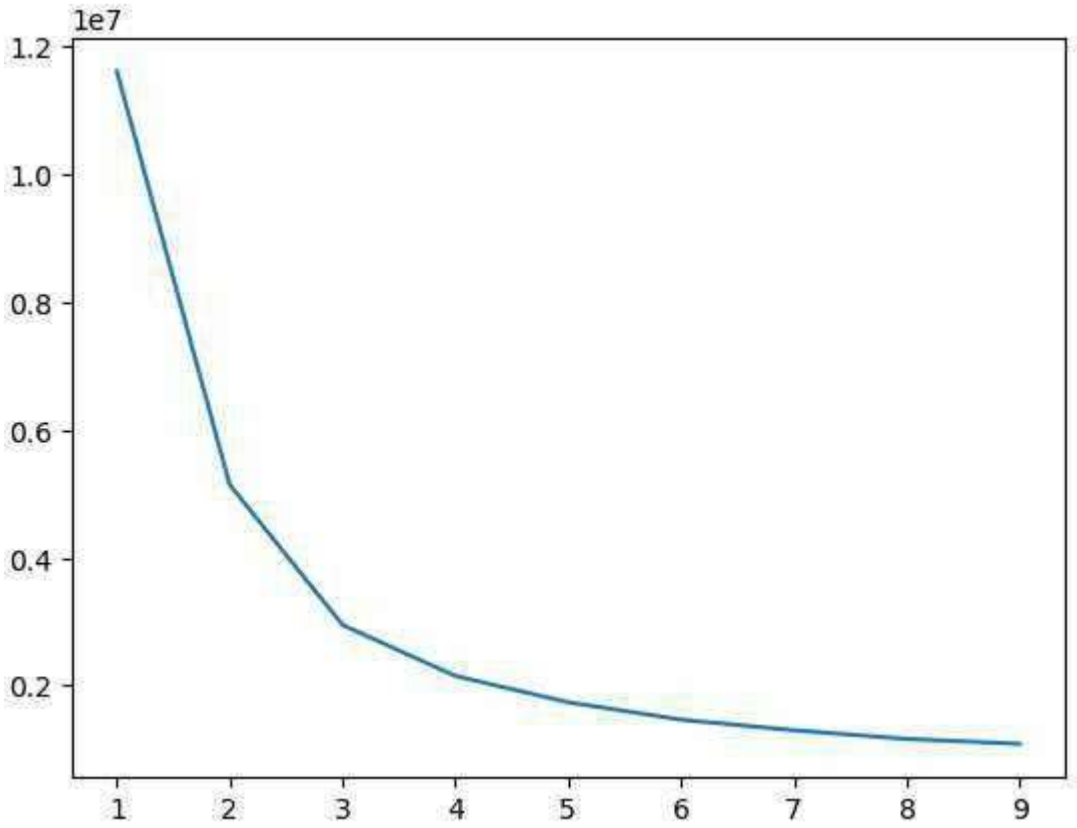

```
Decision Tree Accuracy: 0.36
Logistic Regression Accuracy: 0.46
SVM Accuracy: 0.44
```

In [7]:
```
# 7
# Using the diabetes dataset, implement a Decision Tree Classifier and
# plot ccp_alpha of built tree against any one OT parameter (node count or max

# Import necessary Libraries
from sklearn.datasets import load iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Load the Iris dataset
iris= load_iris()
X =iris.data# Features (sepal Length, sepal width, petal Length, petal width
y =iris.target# Target variable (setosa, versicolor, virginica)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randc

# Create a Decision Tree Classifier
dtc = DecisionTreeClassifier(random_state=42)
# Fit the classifier to the training data
dtc.fit(X_train, y_train)

# Get the cost-complexity pruning path (ccp_alpha values)
path= dtc.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas # The alpha values
impurities= path.impurities# Corresponding impurities at each alpha value

# Print ccp_alpha values
print("ccp_alpha values:")
print(ccp_alphas)
# Optionally, print the impurities at each ccp_alpha value
print("\nimpurities at each ccp_alpha value:")
print(impurities)
```

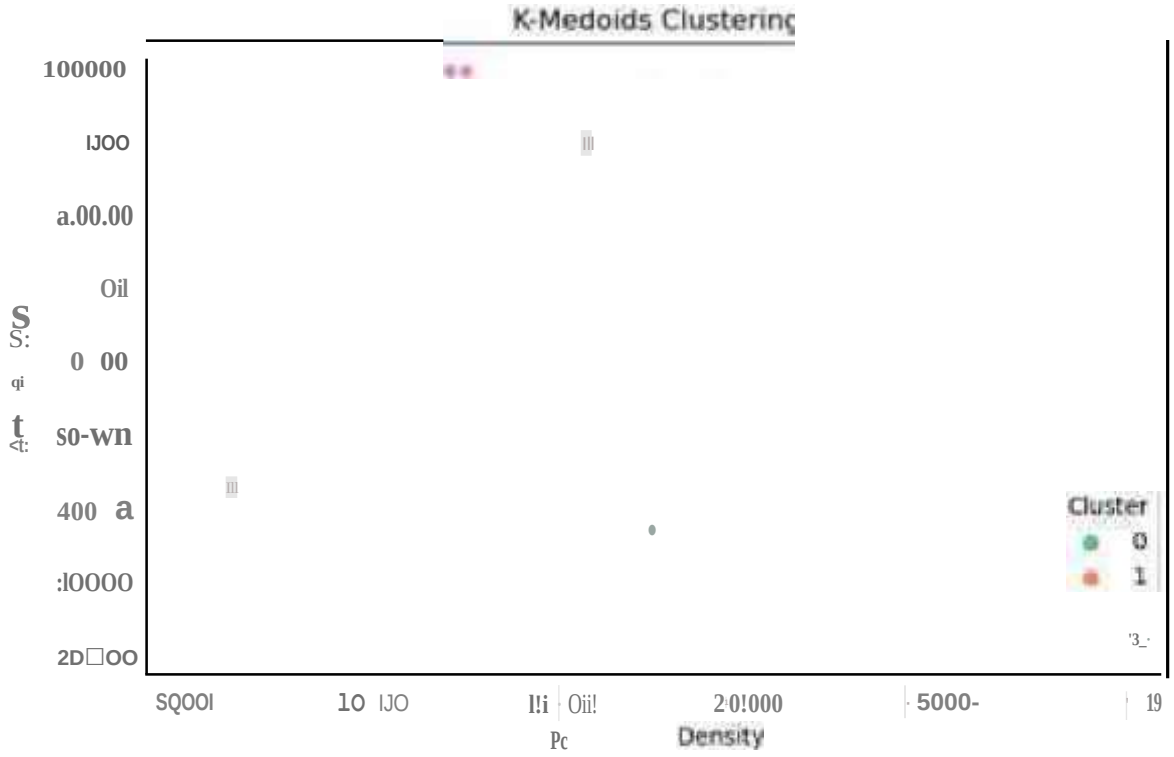
```
ccp_alpha values:
[0.         0.00923521 0.01269841 0.01269841 0.01847042 0.02705804
 0.25028684 0.31210884]

Impurities at each ccp_alpha value:
[0.         0.01847042 0.04386724 0.05656566 0.07503608 0.10209411
 0.35238095 0.6644898]
```

In [8]:
```
# 8
# Write a Python script that implements the Elbow Method
# to determine the optimal number of clusters for a generated dataset. (Genera
import numpy as np
X = np.random.rand(14, 2) * 10
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
elbow=[]
for i in range(1,10):
  km=KMeans(n_clusters=i)
  km.fit(x)
  elbow.append(km.inertia_)
plt.plot(range(1,10),elbow)
```

Out[8]: [<matplotlib.lines.Line2D at 0x1d5ad7b82b0>]



In [12]:
```
# 9
# Generate a random dataset of urban areas with features such as
# population density, average income, and amenities, (min. 100 data).
# Apply K-medoids clustering to identify neighbourhoods with similar character
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt
import seaborn as sns

# Generate random dataset (100 urban areas)
np.random.seed(42)
data= {
    "Latitude": np.random.uniform(40.5, 40.9, 100),
    "Longitude": np.random.uniform(-74.0, -73.7, 100),
    "Population  Density": np.random.randint(5000, 30000, 100),
    "Average  Income": np.random.randint(20000, 100000, 100),
    "Amenities": np.random.randint(5, 50, 100),
}

df = pd.DataFrame(data)

# Standardize features
X = df[["Population Density", "Average Income", "Amenities"]]
X_scaled = StandardScaler().fit_transform(X)

# Apply K-Medoids clustering
kmedoids = KMedoids(n_clusters=4, random_state=42, init="random")
df["Cluster"] = kmedoids.fit_predict(X_scaled)

# Plot clusters
plt.figure(figsize=(8,  5))
sns.scatterplot(x=df["Population Density"], y=df["Average Income"], hue=df["Cl
plt.xlabel("Population Density")
plt.ylabel("Average Income")
plt.title("K-Medoids Clustering")
plt.legend(title="Cluster")
plt.show()
```


K-Medoids Clustering
100000
IJOO
a.00.00
Oil
0 00
S0-wn
400 a
:lOOOO
2D□OO
SQOOI
lO IJO
l!i Oii!
Pc
2·0!000
Density
5000-
19
Cluster
0
1

In [13]:
```
#10
# Generate a random dataset containing customer information (e.g. age spendi
# apply K-medoids clustering to segment customers into distinct groups.
# Visualize the clusters and summarize the characteristics of each segment
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn extra.cluster import KMedoids

# Generate random dataset (100 customers)
np.random.seed(42)
data= {
    "Age": np.random.randint(18, 70, 100),
    "Spending  Score": np.random.randint(1, 100, 100)
}

df = pd.DataFrame(data)

# Standardize features
X_scaled = StandardScaler().fit_transform(df)

# Apply K-Medoids clustering
kmedoids = KMedoids(n_clusters=4, random_state=42, init="random")
df["Cluster"] = kmedoids.fit_predict(X_scaled)

# Plot clusters
plt.figure(figsize=(8,  5))
sns.scatterplot(x=df["Age"], y=df["Spending Score"], hue=df["Cluster"], palett
plt.xlabel("Age")
plt.ylabel("Spending  Score")
plt.title("Customer Segmentation using K-Medoids")
plt.legend(title="Cluster")
plt.show()

# Summarize characteristics of each segment
summary= df.groupby("Cluster").agg({"Age": ["mean", "min", "max"], "Spending
print(summary)
```

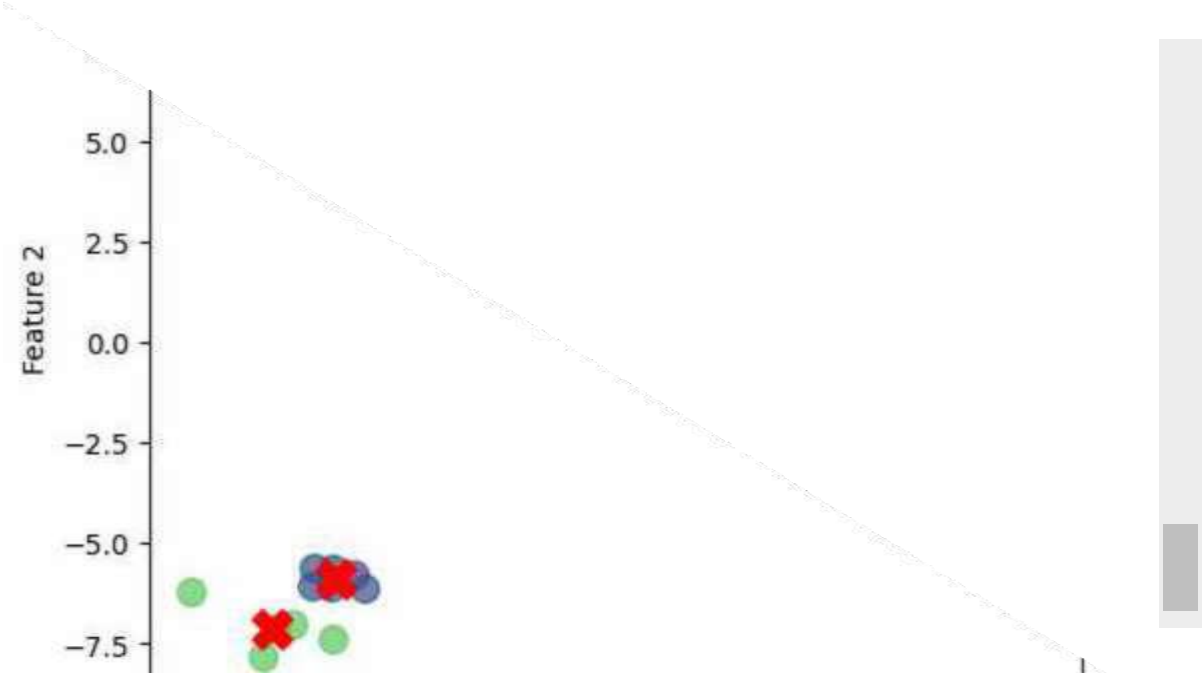
In [14]:
```
#11
# You are working with an e-commerce company that wants to segment its custome
# Generate random dataset that includes features Like total spending frequenc
# Develop a Python program to perform K-means clustering.
# Visualize the clusters and summarize the characteristics of each segment.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Generate random dataset (100 customers)
np.random.seed(42)
data= {
    "Total Spending": np.random.randint(100, 5000, 100),
    "Frequency of Purchases": np.random.randint(1, 50, 100),
    "Average Basket Size": np.random.randint(10, 500, 100)
}

df = pd.DataFrame(data)

# Standardize features
X_scaled = StandardScaler().fit_transform(df)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df["Cluster"] = kmeans.fit_predict(X_scaled)

# Plot clusters
plt.figure(figsize=(8,  5))
sns.scatterplot(x=df["Total Spending"], y=df["Frequency of Purchases"], hue=df
plt.xlabel("Total Spending")
plt.ylabel("Frequency of Purchases")
plt.title("Customer Segmentation using K-Means")
plt.legend(title="Cluster")
plt.show()

# Summarize characteristics of each segment
summary= df.groupby("Cluster").agg({"Total Spending": ["mean", "min", "max"],
                                    "Frequency of Purchases":["mean", "min",
                                    "Average Basket Size":["mean", "min", "rr

print(summary)
```

```
C:\Users\VRAJ\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1429:    Us
erWarning: KMeans is known to have a memory leak on Windows with MKL, when th
ere are less chunks than available threads. You can avoid it by setting thee
nvironment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

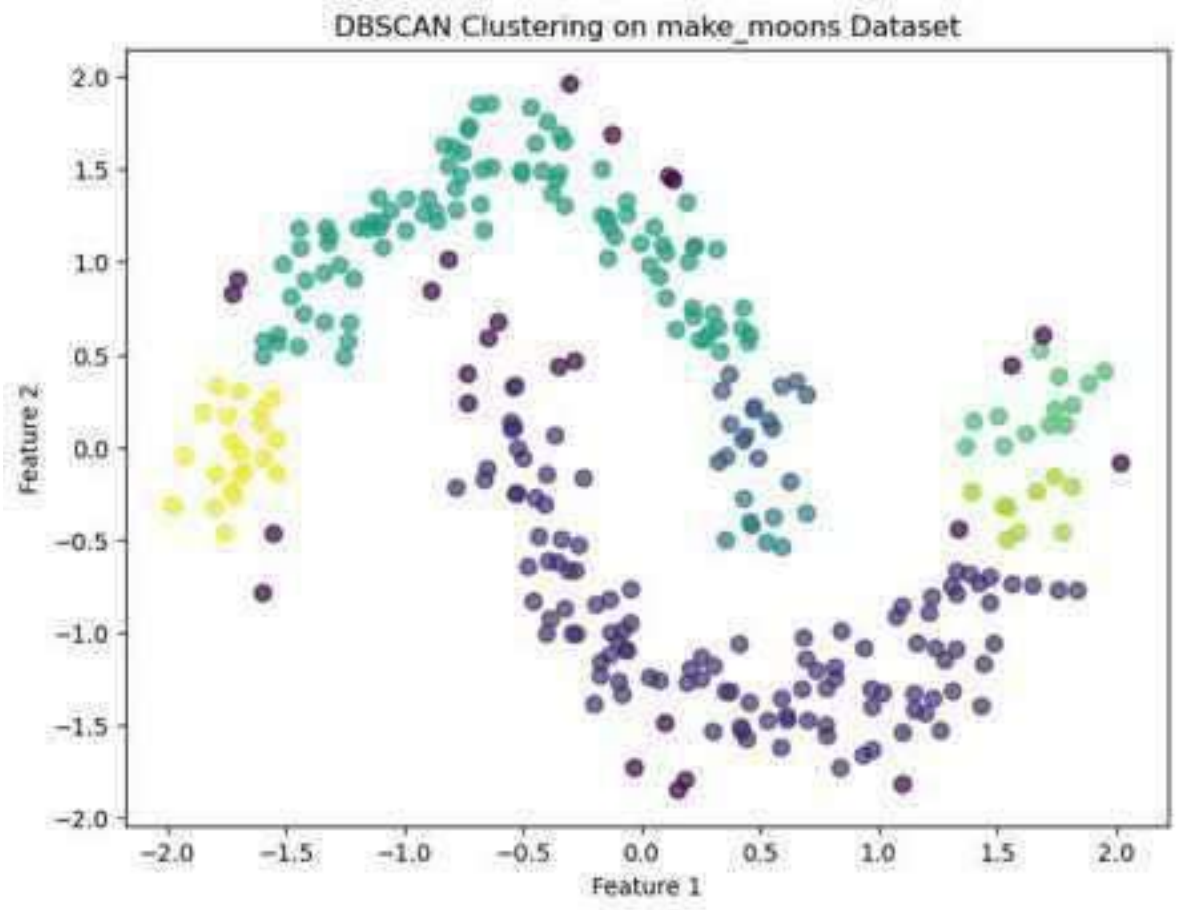
In [17]:
```
# 12
# Use make_blobs to generate 300 samples. Analyse the data and apply KMeans cl
# generate the graph for better representation.
# Show what happens if you change n_clusters count from 2 to 7(2,3,4,5,6,7)?
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Step 1: Generate synthetic dataset with 2 features
X, y = make_blobs(n_samples=300, centers=2, cluster_std=0.60, random_state=42)

# Step 2: Apply KMeans clustering
kmeans = KMeans(n_clusters=2)  # Specify number of clusters
kmeans.fit(X)

# Step 3: Get the cluster centers and Labels
centers= kmeans.cluster_centers
labels= kmeans.labels

# Step 4: Visualize the clusters and the cluster centers
plt.figure(figsize=(8,  6))

# Plot data points
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7)

# Plot the cluster centers
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='(

plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

In [23]:
```
#13
# A grocery store wants to analyze customer purchase patterns to improve produ
# Generate random transaction data (products purchased together)
# apply K-means clustering to group similar transactions.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Generate synthetic transaction data (300 transactions  2 product categories)
np.random.seed(42)
X = np.random.randint(0, 10, size=(300, 2))  # Products purchased together

# Apply KMeans clustering
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

# Get the cluster centers and Labels
centers= kmeans.cluster_centers
labels= kmeans.labels

# Visualize the clusters and the cluster centers
plt.figure(figsize=(8,  6))

# Plot data points
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7)

# Plot the cluster centers
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='(

plt.title('K-Means Clustering for Grocery Transactions')
plt.xlabel('Product Category 1 Purchases')
plt.ylabel('Product Category 2 Purchases')
plt.legend()
plt.show()
```

In [24]:
```
#14
# Generate random dataset with 35 points.
# Apply KMeans clustering and generate the plot with each cluster points with
# What is the impact of varying cluster count?
# Justify your understanding with suitable graphs.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic dataset with 35 points and 2 features
X, _ = make_blobs(n_samples=35, centers=3, cluster_std=0.80, random_state=42)

# Function to apply KMeans clustering and plot results
def plot_kmeans_clusters(X, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    labels= kmeans.fit_predict(X)
    centers= kmeans.cluster_centers

    plt.figure(figsize=(6,  5))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=100, alpha=0.7)
    plt.scatter(centers[:, 0], centers[:,  1], s=200, c='red', marker='X', labe
    plt.title(f'K-Means Clustering (k={n_clusters})')
    plt.xlabel('Feature  1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

# Plot for different cluster counts
for k in [2, 3, 4, 5]:
    plot_kmeans_clusters(X,  k)
```



In [25]:
```
#15
# Develop a Python program to implement DBSCAN using the sklearn Library. Use
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler

# Generate synthetic dataset (moons shape)
X, _ = make_moons(n_samples=300, noise=0.1, random_state=42)

# Standardize the dataset
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.2, min_samples=5)
labels= dbscan.fit_predict(X_scaled)

# Plot the DBSCAN clusters
plt.figure(figsize=(8,  6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', s=50, al
plt.title('DBSCAN Clustering on make moons Dataset')
plt.xlabel('Feature  1')
plt.ylabel('Feature  2')
plt.show()
```



In [37]:
```
# 16
# Generate a random dataset with 5 columns and use 3rd and 4th column as data
# Consider eps=5 and minPts=5.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generate random dataset (100 samples, 5 columns)
np.random.seed(42)
data= np.random.randint(0, 100, size=(100, 5))

# Extract 3rd and 4th columns for clustering
X = data[:, 2:4]

# Standardize the dataset
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=5, min_samples=5)
labels= dbscan.fit_predict(X_scaled)

# Plot the DBSCAN clusters
plt.figure(figsize=(8,  6))
plt.scatter(X[:, 0], X[:, 1],c=labels, cmap='viridis', s=50, alpha=0.7)
#   plt.scatter(X[labels==-1,0),X[labels==-1,1),s=20,color='black')
# plt.scatter(X[labels==0,0),X[labels==0,1),s=20,color='blue')
# plt.scatter(X[labels==1,0),X[labels==1,1],s=20,color='red')
# plt.scatter(X[labels==2,0),X[labels==2,1),s=20,color='grey')
# plt.scatter(X[labels==3,0),X[labels==3,1),s=20,color='green')
plt.title('DBSCAN Clustering using 3rd and 4th Columns')
plt.xlabel('Feature 3')
plt.ylabel('Feature 4')
plt.show()
```

In [38]:
```
# 17
# Use DBSCAN to cluster customers based on their purchasing behaviour
#(ex.frequency of purchases, average transaction amount)(Generate random dat
# Develop a program that analyses the clusters to identify different customer
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generate random dataset for 1ee customers
np.random.seed(42)
data= {
    "CustomerID": np.arange(1, 101),
    "Frequency_of_Purchases": np.random.randint(1, 50, 100),  # Number of pure
    "Avg_Transaction_Amount": np.random.randint(10, 500, 100)  # Average amour,
}

df = pd.DataFrame(data)

# Extract relevant features for clustering
X = df[["Frequency_of_Purchases", "Avg_Transaction_Amount"]]

# Standardize the dataset
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=1, min_samples=5)
df["Cluster"] = dbscan.fit_predict(X_scaled)

# Plot the clusters
plt.figure(figsize=(8,  6))
plt.scatter(df["Frequency_of_Purchases"], df["Avg_Transaction_Amount"], c=df["
plt.xlabel("Frequency of Purchases")
plt.ylabel("Average Transaction Amount")
plt.title("DBSCAN Clustering of Customers")
plt.colorbar(label="Cluster")
plt.show()

# Analyze customer segments
cluster_summary = df.groupby("Cluster")[["Frequency_of_Purchases", "Avg_Transa
print(cluster_summary)
```

In [40]:
```
# 18
# Generate a data frame containing two features(Spending,features) for 166 dat
# Apply DBSCAN and do parametric analysis for eps value and show the impact on
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generate random dataset with 166 data points
np.random.seed(42)
df = pd.DataFrame({
    "Spending": np.random.randint(10, 500, 100),
    "Features": np.random.randint(1, 50, 100)
})

# Extract features for clustering
X = df[["Spending", "Features"]]

# Standardize the dataset
X_scaled = StandardScaler().fit_transform(X)

# Different epsilon values for analysis
eps_values = [0.5, 1, 2, 5]

plt.figure(figsize=(12,  8))

for i, eps in enumerate(eps_values, 1):
    dbscan = DBSCAN(eps=eps, min_samples=5)
    labels= dbscan.fit_predict(X_scaled)

    # Plot clustering result
    plt.subplot(2, 2, i)
    plt.scatter(df["Spending"], df["Features"], c=labels, cmap='viridis', s=5e
    plt.xlabel("Spending")
    plt.ylabel("Features")
    plt.title(f"DBSCAN Clustering  (eps={eps})")

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

In [41]:
```
#19
# Generate a random dataset with 4 columns and use 1st and 3rd column as data
# What if eps is changed from 1 to 5?
# How it will affect the performance of the model? What is the impact of minPt
# Justify it with graph by changing minPts.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generate random dataset (100 samples, 4 columns)
np.random.seed(42)
df = pd.DataFrame({
    "Feature1": np.random.randint(10, 100, 100),
    "Feature2": np.random.randint(20, 200, 100),
    "Feature3": np.random.randint(5, 50, 100),
    "Feature4": np.random.randint(1, 10, 100)
})

# Extract 1st and 3rd columns for clustering
X = df[["Feature1", "Feature3"]]

# Standardize the dataset
X_scaled = StandardScaler().fit_transform(X)

# Analyze the impact of changing eps values
eps_values = [1, 2, 3, 5]
plt.figure(figsize=(12,  8))

for i, eps in enumerate(eps_values, 1):
    dbscan = DBSCAN(eps=eps, min_samples=5)
    labels= dbscan.fit_predict(X_scaled)

    plt.subplot(2, 2, i)
    plt.scatter(df["Feature1"], df["Feature3"], c=labels, cmap='viridis', s=5E
    plt.xlabel("Feature1")
    plt.ylabel("Feature3")
    plt.title(f"DBSCAN Clustering  (eps={eps})")

plt.tight_layout()
plt.show()

# Analyze the impact of changing minPts values
min_samples_values = [3, 5, 7, 10]
plt.figure(figsize=(12,  8))

for i, min_pts in enumerate(min_samples_values, 1):
    dbscan = DBSCAN(eps=2, min_samples=min_pts)
    labels= dbscan.fit_predict(X_scaled)

    plt.subplot(2, 2, i)
    plt.scatter(df["Feature1"], df["Feature3"], c=labels, cmap='viridis', s=5E
    plt.xlabel("Feature1")
    plt.ylabel("Feature3")
    plt.title(f"DBSCAN Clustering  (minPts={min_pts})")
```

In [47]:
```
#20
# Generate random dataset for maLL_customer dataset with features Like Custome
# Consider minimum 200 random samples for analysis. Apply DBSCAN as well as ag
# generate the dendrogram. Plot points in each cluster for DBSCAN with differe
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN, AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
import scipy.cluster.hierarchy as she

# Generate random dataset (200 samples)
np.random.seed(42)
df = pd.DataFrame({
    "CustomerID": np.arange(1, 201),
    "Genre": np.random.choice(["Male", "Female"], 200),
    "Age": np.random.randint(18, 70, 200),
    "Annual_Income": np.random.randint(15000, 100000, 200),
    "Spending_Score": np.random.randint(1, 100, 200)
})

# Convert categorical 'Genre' column to numerical
df["Genre"] =  df["Genre"].map({"Male": 0, "Female": 1})

# Extract relevant features for clustering
X = df[["Age", "Annual_Income", "Spending_Score"]]

# Standardize the dataset
X_scaled = StandardScaler().fit_transform(X)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=1, min_samples=5)
df["DBSCAN_Cluster"] = dbscan.fit_predict(X_scaled)

# Plot DBSCAN Clustering
plt.figure(figsize=(8,  6))
plt.scatter(df[ "Annual_Income"], df["Spending_Score"], c=df["DBSCAN_Cluster"],
plt.xlabel("Annual Income")
plt.ylabel("Spending  Score")
plt.title("DBSCAN Clustering of Mall Customers")
plt.colorbar(label="Cluster")
plt.show()

# Apply Agglomerative Clustering
agg_clustering = AgglomerativeClustering(n_clusters=5)
df["Agglomerative_Cluster"] = agg_clustering.fit_predict(X_scaled)

# Generate Dendrogram for Agglomerative Clustering
plt.figure(figsize=(10,  6))
linked= shc.linkage(X_scaled, method='ward')
shc.dendrogram(linked)
plt.title("Dendrogram for Agglomerative Clustering")
plt.xlabel("Customers")
plt.ylabel("Distance")
plt.show()
```

In [48]:
```
# 21
#Write a Python code to perform k-fold cross-validation on the dataset using
# Use the following steps:
#•Import necessary Libraries.
#•Prepare the dataset synthetically using make_classification.
# • Preprocess the data if necessary (e.g., scaling).
#•Implement k-fold cross-validation with 10 folds.
#•Train the SVM classifier on the training set and evaluate it on the tests
#•Print the accuracy for each fold and the average accuracy across all folds
# Dataset Description:
#•The dataset consists of 100 samples with 20 features.
# The target variable is binary (0 or 1).
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_score
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler

# Step 1: Generate synthetic dataset
X, y = make_classification(n_samples=100, n_features=20, n_classes=2, randoms

# Step 2: Preprocess the data (Standardization)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Initialize SVM classifier
svm_classifier = SVC(kernel='linear')

# Step 4: Implement 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
scores= cross_val_score(svm_classifier, X_scaled, y, cv=kf, scoring='accuracy

# Step 5: Print the accuracy for each fold and the average accuracy
for i, score in enumerate(scores, 1):
    print(f"Fold {i}: Accuracy= {score:.4f}")

print(f"\nAverage Accuracy: {np.mean(scores):.4f}")
```

```
Fold 1: Accuracy  0.9000
Fold 2: Accuracy = 0.9000
Fold 3: Accuracy = 0.9000
Fold 4: Accuracy = 1.0000
Fold 5: Accuracy = 1.0000
Fold 6: Accuracy = 1.0000
Fold 7: Accuracy = 1.0000
Fold 8: Accuracy = 0.9000
Fold 9: Accuracy = 1.0000
Fold 10: Accuracy = 1.0000

Average Accuracy: 0.9600
```

In [49]:
```
#22
#Write a Python code to perform bootstrap sampling on a dataset using a Rando
# The dataset consists of features and Labels as specified below.
# Dataset Description:
#•The dataset consists of 150 samples with 10 features.
#•The target variable is categorical with three classes (0, 1, 2).
# Instructions:
#•Import the necessary Libraries.
#•Load or create a synthetic dataset with the specified characteristics.
#•Implement bootstrap sampling to create multiple samples from the original
#•Train a Random Forest Classifier on each bootstrap sample.
#•Evaluate the model on the original dataset and print the accuracy for each
#•Calculate and print the average accuracy across all bootstrap samples.
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.utils import resample

# Step 1: Generate synthetic dataset
X, y = make_classification(n_samples=150, n_features=10, n_classes=3, n_inforrr

# Convert to DataFrame for easier manipulation
df = pd.DataFrame(X, columns=[f"Feature_{i}" for i in range(10)])
df["Target"] = y

# Step 2: Perform bootstrap sampling and train Random Forest
n_bootstrap_samples = 10  # Number of bootstrap iterations
accuracies = []

for i in range(n_bootstrap_samples):
    # Create a bootstrap sample
    df_sample = resample(df, replace=True, n_samples=len(df), random_state=i)

    # Split into features and target
    X_sample = df_sample.drop(columns=["Target"])
    y_sample = df_sample["Target"]

    # Train Random Forest Classifier
    rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_classifier.fit(X_sample, y_sample)

    # Evaluate on the original dataset
    y_pred = rf_classifier.predict(X)
    accuracy= accuracy_score(y, y_pred)
    accuracies.append(accuracy)

    print(f"Bootstrap Sample {i+1}: Accuracy= {accuracy:.4f}")

# Step 3: Print average accuracy
print(f"\nAverage Accuracy across Bootstrap Samples: {np.mean(accuracies):.4f}
```

In [50]:
```
# 23
# Write a Python code to perform Leave-One-Out Cross-Validation (LOOCV) on a a
# The dataset consists of features and Labels as specified below.
# Dataset Description:
#•The dataset consists of 100 samples with 5 features.
#•The target variable is binary (0 or 1).
# Instructions:
#     • Import the necessary Libraries.
#•Load or create a synthetic dataset with the specified characteristics.
#•Implement Leave-One-Out Cross-Validation (LOOCV).
#•Train a Decision Tree Classifier on each training set and evaluate it on t
# Print the accuracy for each iteration and the average accuracy across all it
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import accuracy_score

# Step 1: Generate synthetic dataset
X, y = make_classification(n_samples=100, n_features=S, n_classes=2, random st

# Step 2: Initialize Leave-One-Out Cross-Validation
loo= LeaveOneOut()
accuracies= []

# Step 3: Perform LOOCV with Decision Tree Classifier
for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train Decision Tree Classifier
    model= DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)

    # Evaluate on Left-out sample
    y_pred = model.predict(X_test)
    accuracy= accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

    print(f"Iteration {len(accuracies)}: Accuracy   {accuracy:.4f}")

# Step 4: Print average accuracy
print(f"\nAverage Accuracy  across  LOOCV  iterations:  {np.mean(accuracies):.4f}"
```

In [51]:
```
# 24
# Write a Python code to perform Leave-One-Out Cross-Validation (LOOCV) on a a
# The dataset consists of features and Labels as specified below.
# Dataset Description:
#•The dataset consists of 150 samples with 10 features.
#•The target variable is categorical with three classes (0 1  2).
#•Introduce some noise to the dataset to make the classification task more
# Instructions:
#•Import the necessary  Libraries.
#•Load or create a synthetic dataset with the specified characteristics (mak
#•Implement Leave-One-Out Cross-Validation (LOOCV).
#•Train a Decision Tree Classifier on each training set and evaluate it on t
#•Calculate and print the accuracy for each iteration as well as the confus
#•Calculate and print the average accuracy across all iterations.
# Print confusion matrix and classificarion report.
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import accuracy_score, confusion_matrix, classification r

# Generate synthetic dataset with noise
X, y = make_classification(
    n_samples=150, n_features=10, n_classes=3, n_informative=7, n_redundant=2,
    n_clusters_per_class=1, flip_y=0.1, random_state=42
)

# Initialize Leave-One-Out Cross-Validation
loo= LeaveOneOut()
accuracies=[]
y_true, y_pred_all = [], []

# Perform LOOCV
for train_index, test_index in loo.split(X):
    # Split dataset
    X_train, X test= X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train Decision Tree Classifier
    model= DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)

    # Evaluate on Left-out sample
    y_pred = model.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))

    # Store true Labels and predictions for final confusion matrix
    y_true.append(y_test[0])
    y_pred_all.append(y_pred[0])

# Calculate and print overall accuracy
average_accuracy = np.mean(accuracies)
print(f"\nAverage Accuracy  across  LOOCV  iterations:  {average_accuracy:.4f}")

# Compute confusion matrix
conf_matrix  =  confusion_matrix(y_true,  y_pred_all)
```

```
print("\nConfusion  Matrix:\n", conf_matrix)

# Print classification report
class_report = classification_report(y_true, y_pred_all)
print("\nClassification Report:\n", class_report)
```

```
Average Accuracy across LOOCV iterations: 0.6667

Confusion Matrix:
 [[31 11  8]
 [15 32  2]
 [12  2 37]]

Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.62      0.57        50
           1       0.71      0.65      0.68        49
           2       0.79      0.73      0.76        51

    accuracy                           0.67       150
   macro avg       0.68      0.67      0.67       150
weighted avg       0.68      0.67      0.67       150
```

In [52]:
```
# 25
# Write a Python code to perform Stratified K-Fold Cross-Validation on a datas
# The dataset consists of features and Labels as specified below.
# Dataset Description:
#•The dataset consists of 200  samples with 15 features.
#•The target variable is categorical with four classes (BJ  1J  2J  3).
# Instructions:
#•Import the necessary  Libraries.
#•Load or create a synthetic dataset with the specified characteristicsJ  ens
#•Implement Stratified K-Fold Cross-Validation with 5 folds.
#•Train a Random Forest Classifier on each training set and evaluate it on t
#•Calculate and print the accuracy for each foldJ  as well as the average ace
# Generate and visualize the classification report for the final modeLJ
# including precisionJ  recaLLJ  and Fl-score for each class.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report

# Generate synthetic dataset with class imbalance
X, y = make_classification(
    n_samples=200, n_features=lS, n_classes=4, n_informative=10,
    n_redundant=3, weights=[0.5, 0.2, 0.2, 0.1], random_state=42
)

# Initialize Stratified K-Fold
skf = StratifiedKFold(n_splits=S, shuffle=True, random_state=42)
accuracies=[]
y_true_all, y_pred_all = [], []

# Perform Stratified K-Fold Cross-Validation
for fold, (train_index, test_index) in enumerate(skf.split(X, y), 1):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train Random Forest Classifier
    model= RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # Predict on validation set
    y_pred = model.predict(X_test)
    ace= accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    print(f"Fold {fold}  Accuracy:  {acc:.4f}")

    # Store true and predicted Labels for final classification report
    y_true_all.extend(y_test)
    y_pred_all.extend(y_pred)

# Calculate and print average accuracy
average_accuracy = np.mean(accuracies)
print(f"\nAverage Accuracy  across  Stratified  K-Folds:  {average_accuracy:.4f}")
```

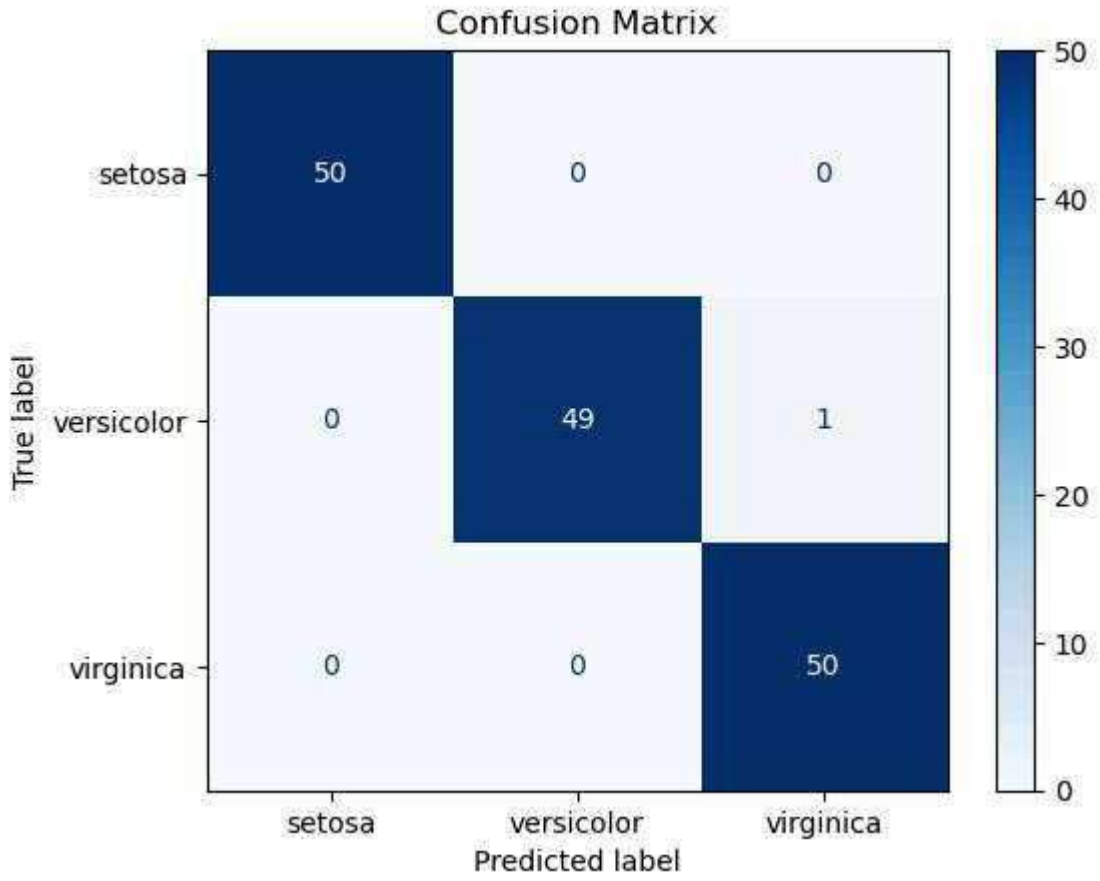
```
# Generate classification report
class_report = classification_report(y_true_all, y_pred_all)
print("\nClassification Report:\n", class_report)

# Visualize classification report
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

cm= confusion_matrix(y_true_all, y_pred_all)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix")
plt.show()
```

```
Fold 1 Accuracy: 0.7250
Fold 2 Accuracy: 0.7250
Fold 3 Accuracy: 0.7500
Fold 4 Accuracy: 0.7250
Fold 5 Accuracy: 0.6000

Average Accuracy across Stratified K-Folds: 0.7050

Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.90      0.78        99
           1       0.62      0.50      0.56        40
           2       0.77      0.68      0.72        40
           3       1.00      0.24      0.38        21

    accuracy                           0.70       200
   macro avg       0.77      0.58      0.61       200
weighted avg       0.73      0.70      0.68       200
```

In [53]:
```
#26
# Write a Python code to perform a Grid Search with Cross-Validation on the Ir
# a Support Vector Machine (SVM) classifier.
# The goal is to find the best hyperparameters for the SVM model.Dataset Descr
#•The Iris dataset consists of 156 samples with 4 features.
#•The target variable is categorical with three classes (Setosa, Versicolor,
# Instructions:
#•Import the necessary Libraries.
#•Load the Iris dataset from sklearn.datasets.
#•Split the dataset into features and Labels.
#•Implement a Grid Search with 5-fold Cross-Validation to find the best hype
# Consider the following hyperparameters: C: [6.1, 1, 16, 166) kernel: ['Linea
#•Print the best hyperparameters found by the Grid Search.
#•Evaluate the best model on the entire dataset and print the accuracy.
# Generate and visualize the confusion matrix for the best model.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion rr

#  Load the Iris dataset
iris= datasets.load_iris()
X, y = iris.data, iris.target  # Features and Labels

# Define the parameter grid for Grid Search
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Initialize SVM classifier
svm = SVC()

# Perform Grid Search with 5-Fold Cross-Validation
grid_search = GridSearchCV(svm, param_grid, cv=S, scoring='accuracy', n_jobs=-
grid_search.fit(X, y)

# Print the best hyperparameters found
print(f"Best  Hyperparameters:  {grid_search.best_params_}")

# Train the best model on the entire dataset
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X)

# Evaluate and print accuracy
accuracy= accuracy_score(y, y_pred)
print(f"Accuracy of Best Model: {accuracy:.4f}")

# Generate and print classification report
print("\nClassification  Report:\n", classification_report(y,  y_pred))

# Generate and visualize confusion matrix
cm= confusion_matrix(y,  y_pred)
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix")
plt.show()
```

```
Best Hyperparameters: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy of Best Model: 0.9933

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      0.98      0.99        50
           2       0.98      1.00      0.99        50

    accuracy                           0.99       150
   macro avg       0.99      0.99      0.99       150
weighted avg       0.99      0.99      0.99       150
```



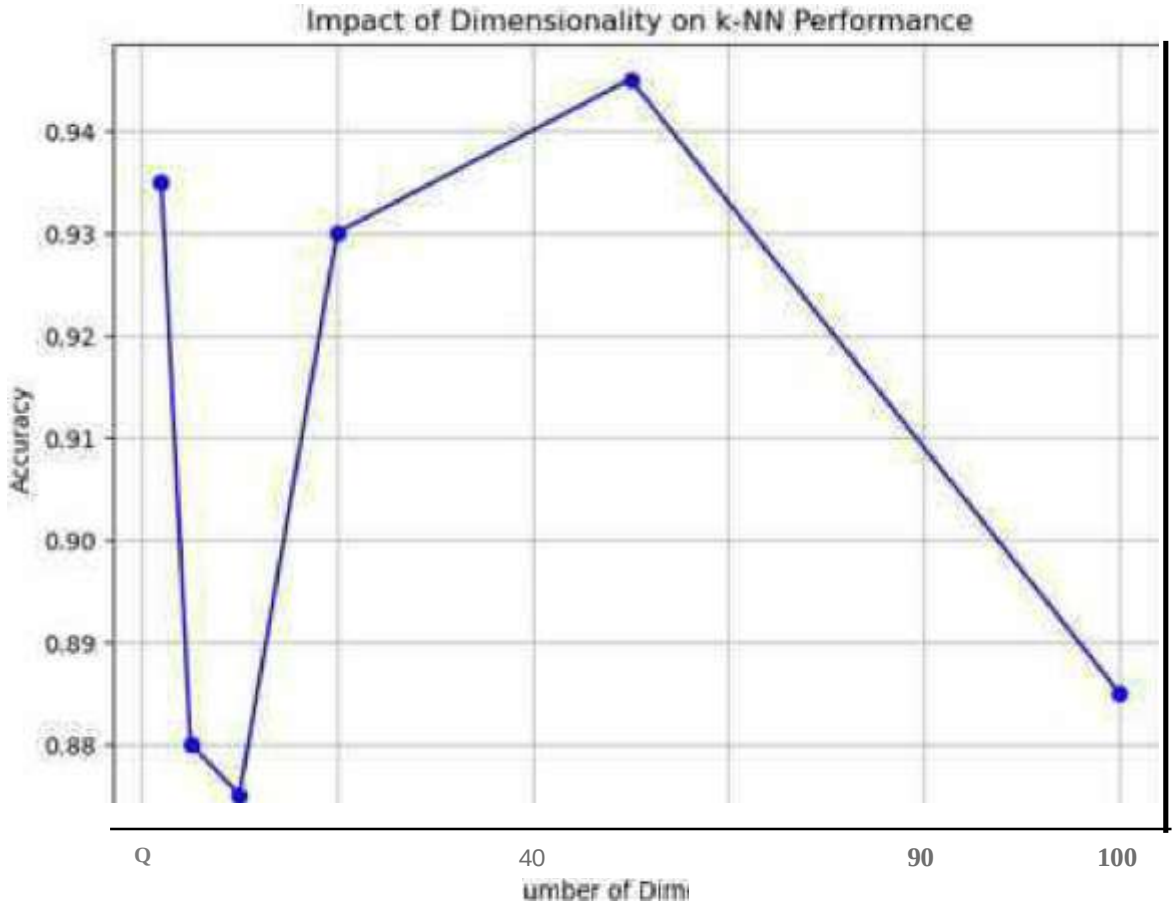
In [55]:

```
# 27
# Write a Python program to apply Principal Component Analysis (PCA) on the Ir
# and reduce its dimensionality to 2 dimensions.
# You should use sklearn's PCA implementation.
# After reducing the dimensions, visualize the transformed data in a 20 scatte
# coloring the points based on their respective classes.
# Dataset Description:
#•The Iris dataset consists of 150 samples with 4 features.
#•The target variable is categorical with three classes (Setosa, Versicolor,
# Instructions:
#•Import the necessary Libraries.
#•Load the Iris dataset from sklearn.datasets.
#•Separate the features and the target variable.
#•Standardize the features (mean= e, variance= 1).
#•Apply PCA to reduce the dimensionality of the dataset to 2 dimensions.
# Visualize the transformed data in a 20 scatter plot, using different colors
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris= load_iris()
X = iris.data
y = iris.target
feature  names= iris.feature  names

# Standardize the data (important for PCA)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensions to 2
pea= PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['species'] = y
# print(pca_df)
# Create a DataFrame for the reduced data
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['species'] = y

# Plot the results
plt.figure(figsize=(8, 6))
for species in np.unique(y):
    plt.scatter(pca_df[pca_df['species']  ==  species]['PC1'],
                pca_df[pca_df['species']  ==  species]['PC2'],
                label=iris.target_names[species])

plt.title('PCA of Iris Dataset (2D)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
```

In [57]:
```
# 28
# Write a Python program that demonstrates the curse of dimensionality
# using the k-Nearest Neighbors (k-NN) classifier with k=5.
# The program should create a synthetic dataset with varying dimensions and
# evaluate the performance of the k-NN classifier as the number of dimensions
#•Generate a synthetic dataset with a fixed number of samples (e.g., 1666) a
#•Use the k-NN classifier to classify the data.
#•Measure and print the accuracy of the classifier for each dimensionality.
# Visualize the accuracy as a function of dimensionality.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Fixed number of samples
n_samples = 1000

# Varying number of dimensions
dimensions= [2, 5, 10, 20, 50, 100]
accuracies=[]

for dim in dimensions:
    # Ensure n_informative, n_redundant, and n_repeated are valid
    n_informative = max(2, dim// 2)  # At Least 2 informative features
    n_redundant = max(0, dim - n_informative - 1)  # Adjust redundant features
    n_repeated = 0  # No repeated features

    # Generate synthetic dataset with specified dimensions
    X, y = make_classification(n_samples=n_samples, n_features=dim,
                               n_informative=n_informative, n_redundant=n_redu
                               n_repeated=n_repeated, n_classes=2, random_stat

    # Split into training and testing sets (86-26 split)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    # Apply k-NN classifier with k=5
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)

    # Predict on the test set
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy= accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

    # Print accuracy for each dimensionality
    print(f"Dimensionality:{dim}, Accuracy: {accuracy:.4f}")

# Visualize the accuracy as a function of dimensionality
plt.figure(figsize=(8, 6))
plt.plot(dimensions, accuracies, marker='o', linestyle='-', color='b')
plt.xlabel("Number of Dimensions")
plt.ylabel("Accuracy")
plt.title("Impact of Dimensionality on k-NN Performance")
```

```
plt.grid(True)
plt.show()
```

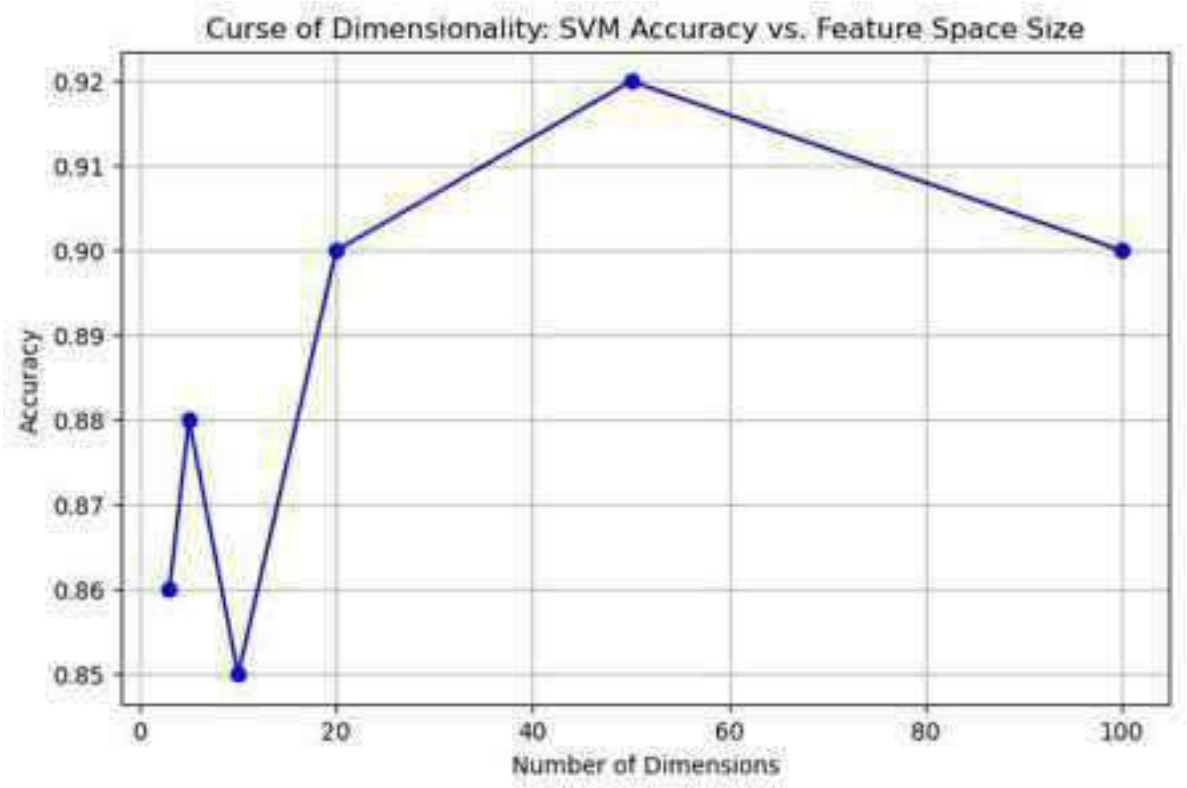
```
Dimensionality: 2, Accuracy: 0.9350
Dimensionality: 5, Accuracy: 0.8800
Dimensionality: 10, Accuracy: 0.8750
Dimensionality: 20, Accuracy: 0.9300
Dimensionality: 50, Accuracy: 0.9450
Dimensionality: 100, Accuracy: 0.8850
```



In [2]:
```
# 29
# You are using a Support Vector Machine (SVM) classifier for a dataset that h
# If you increase the dimensionality from 3 features to 12 features, explain h
# What are the potential challenges that arise from this increase in dimension
# Instructions:
#•Provide detailed answers to both questions, incorporating concepts from ma
#•Use diagrams or examples where appropriate to illustrate your points.
# Discuss the implications of high-dimensional spaces on
# model performance, including overfitting, computational complexity, and the
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Fix random seed for reproducibility
np.random.seed(42)

# Define number of samples and dimensions to test
num_samples = 500
dimensions= [3, 5, 10, 20, 50, 100]  # Different feature space sizes
accuracy_scores = []

for dim in dimensions:
    # Ensure n_informative is at Least 2 to avoid ValueError
    n_informative = max(2, min(dim - 1, int(np.log2(2 * 2))))

    # Generate dataset with proper constraints
    X, y = make_classification(n_samples=num_samples, n_features=dim,
                               n_informative=n_informative,
                               n_redundant=dim - n_informative - 1, # Some rE
                               n_clusters_per_class=1, # Avoid excessive clus
                               n_classes=2, random_state=42)

    # Split into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r

    # Standardize features
    scaler= StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train an SVM classifier
    elf= SVC(kernel='linear', random_state=42)
    clf.fit(X_train, y_train)

    # Predict and compute accuracy
    y_pred = clf.predict(X_test)
    ace= accuracy_score(y_test, y_pred)
    accuracy_scores.append(acc)

    print(f"Dim:{dim}, Accuracy: {acc:.4f}")

# Plot accuracy vs. dimensions
plt.figure(figsize=(8,  5))
```

```
plt.plot(dimensions, accuracy_scores, marker='o', linestyle='-', color='b')
plt.xlabel('Number of Dimensions')
plt.ylabel('Accuracy')
plt.title('Curse of Dimensionality: SVM Accuracy vs. Feature Space Size')
pH.grid()
plt.show()
```

```
Dim: 3, Accuracy: 0.8600
Dim: 5, Accuracy: 0.8800
Dim: 10, Accuracy: 0.8500
Dim: 20, Accuracy: 0.9000
Dim: 50, Accuracy: 0.9200
Dim: 100, Accuracy: 0.9000
```



In [3]:

```
#30
#Create a synthetic dataset using make_blobs with the following specification
#•The dataset should have 4 features and 4 centers (clusters).
#•Initialize the PCA object with n_components=3 to reduce the dataset to 3 a
#•Use Matplotlib to plot the original 40 dataset in a 30 scatter plot and
# the PCA-reduced 30 dataset in another 30 scatter plot side by side.
# Instructions:
#•Import the necessary Libraries (numpy, matplotlib, and sklearn).
#•Generate the synthetic dataset using make blobs.
#•Apply PCA to reduce the dataset to 3 dimensions.
#•Create a figure with two subplots:
#•The first subplot should display the original 30 representation of the dat
# (you can use any three of the four features for this plot).
#•The second subplot should display the PCA-reduced 30 dataset.
#•Ensure that the points in both plots are color-coded based on their respec
# Add appropriate titles and Labels to the plots for clarity.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA

# Step 1: Generate a synthetic dataset with 4 features and 4 clusters
n_samples = 500
n features= 4
n  clusters= 4

X, y = make_blobs(n_samples=n_samples, n_features=n_features, centers=n cluste

# Step 2: Apply PCA to reduce the dataset from 40 to 30
pea= PCA(n_components=3)
X_pca = pca.fit_transform(X)

# Step 3: Create a figure with two subplots
fig=  plt.figure(figsize=(12, 6))

# Plot the original 40 dataset (using first 3 features)
axl = fig.add_subplot(121, projection='3d')
axl.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap='viridis', edgecolor='k')
axl.set_title("Original 4D Dataset (First 3 Features)")
axl.set_xlabel("Feature 1")
axl.set_ylabel("Feature  2")
axl.set_zlabel("Feature  3")

# Plot the PCA-reduced 30 dataset
ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=y, cmap='viridis', edgecc
ax2.set_title("PCA-Reduced 3D Dataset")
ax2.set_xlabel("Principal  Component  1")
ax2.set_ylabel("Principal  Component  2")
ax2.set_zlabel("Principal  Component  3")

# Show the plots
plt.tight_layout()
plt.show()
```

In [4]:
```
# 31
# Use the Breast Cancer dataset to train a random forest classifier.
# The model will be trained to accurately classify tumors as benign or maligna
# The dataset is imbalanced, with a significant majority of benign samples.
# Evaluate the model's performance using accuracy, precision, recall, and Fl-s
# Import necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, fl _

# Step 1: Load the Breast Cancer dataset
cancer_data = load_breast_cancer()
X = cancer data.data  # Features
y = cancer_data.target  # Target (e  = malignant, 1 = benign)

# Convert to DataFrame for better understanding
df  = pd.DataFrame(X, columns=cancer_data.feature_names)
df['target'] = y

# Step 2: Split the dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.2, randc

# Step 3: Standardize the features (mean= e, variance= 1)
scaler= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Train the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 5: Make predictions
y_pred = rf_model.predict(X_test)

# Step 6: Evaluate the model
accuracy= accuracy_score(y_test, y_pred)
precision= precision_score(y_test, y_pred)
recall= recall_score(y_test, y_pred)
fl= fl_score(y_test, y_pred)

# Display results
print(f"Accuracy:{accuracy:.4f}")
print(f"Precision:{precision:.4f}")
print(f"Recall:{recall:.4f}")
print(f"Fl-score:{fl:.4f}")
print("\nClassification  Report:\n", classification_report(y_test,  y_pred))

# Step 7: Visualizing the Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Mali
```

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Step 8: Feature Importance Visualization
feature_importances = rf_model.feature_importances_
feature_names= cancer_data.feature_names
sorted_idx = np.argsort(feature_importances)[::-1][:10] # Top 10 features

plt.figure(figsize=(10,  5))
sns.barplot(x=feature_importances[sorted_idx],  y=np.array(feature_names)[sortE
plt.xlabel("Feature Importance")
plt.ylabel("Feature  Name")
plt.title("Top 10 Important  Features in Breast Cancer Classification")
plt.show()
```

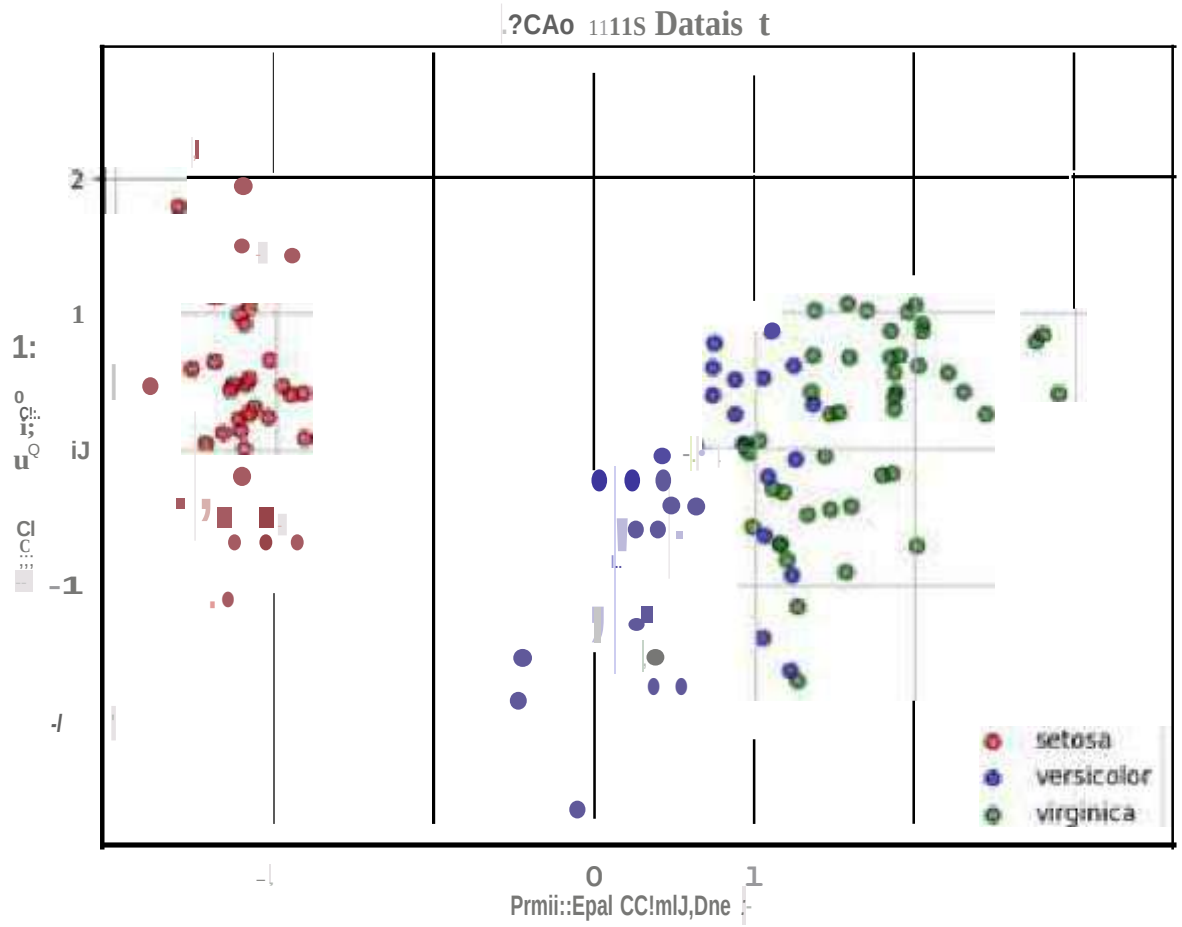
```
Accuracy: 0.9561
Precision: 0.9589
Recall: 0.9722
Fl-score: 0.9655

Classification Report:
              precision    recall  fl-score   support

           0       0.95      0.93      0.94        42
           1       0.96      0.97      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

In [8]:
```
# 32
#Use the Wine dataset, to develop a k-nearest neighbors (k-NN) classifier to
# types of wine based on their chemical properties.
# You will evaluate the model's performance using various metrics, including a
#•Load the Wine dataset from a CSV file
#•checking for missing values, understanding the distribution of classes, an
#•Normalize or standardize the features if necessary to improve the performa
#•split the dataset into training and testing sets (e.g., 80% training and 2
#•Train the classifier on the training dataset.
#•Use the trained model to make predictions on the test dataset.
#•Calculate and print the following performance metrics:
# o Accuracy
# o Precision (for each class)
# o Recall (for each class)
# o Fl-score (for each class)
# Import necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, fl _

# Step 1: Load the Wine dataset from CSV
file_path = "wine.csv"  # Change this if necessary
df = pd.read_csv(file_path)

# Step 2: Explore the dataset
print("\nFirst 5 rows of the dataset:")
print(df.head())

# Check for missing values
print("\nMissing values in the dataset:")
print(df.isnull().sum())

# Class distribution
print("\nClass distribution:")
print(df['Wine'].value_counts())

# Step 3: Separate features and target
X = df.drop(columns=['Wine']) # Assuming 'target' column contains class Label
y = df['Wine']

# Step 4: Standardize the features
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 5: Split the dataset into training and testing sets (80% train, 20% tes
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2

# Step 6: Train the k-NN classifier (using k=S as default)
knn = KNeighborsClassifier(n_neighbors=S)
knn.fit(X_train, y_train)

# Step 7: Make predictions on the test dataset
```

```
y_pred = knn.predict(X_test)

# Step 8: Evaluate the model
accuracy= accuracy_score(y_test, y_pred)
precision= precision_score(y_test, y_pred, average=None)  # Per class
recall= recall_score(y_test, y_pred, average=None)  # Per class
fl= fl_score(y_test, y_pred, average=None) # Per class

# Print metrics
print("\nModel Performance:")
print(f"Accuracy:{accuracy:.4f}")
print( f"Precision: {precision}")
print(f"Recall:{recall}")
print(f"Fl-score:{fl}")
print("\nClassification  Report:\n", classification_report(y_test,  y_pred))

# Step 9: Visualizing the Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.uni
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

In [9]:
```
# 33
# Apply Principal Component Analysis (PCA) to the Iris dataset to reduce its a
# You will visualize the PCA-reduced dataset in a 20 scatter plot using Matplo
# instructions:
# Load the Iris dataset
# Briefly explore the dataset to understand its structure. Print the shape of
# the first few rows to get an overview of the features and target variable.
# Initialize the PCA object to reduce the dataset to 2 dimensions.
# Fit the PCA model to the Iris dataset and transform the dataset
# Use Matplotlib to create a scatter plot of the PCA-reduced dataset.
# Color the points based on their respective species (Setosa, Versicolor, Virg
# visualize how well the PCA has separated the different classes.
# Add appropriate titles and Labels to the axes of the plot.
# Include a Legend to indicate which colors correspond to which species.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris= datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target Labels
target_names = iris.target_names  # Class names

# Explore the dataset
print("Dataset shape:", X.shape)
print("First 5 rows:\n", pd.DataFrame(X, columns=iris.feature_names).head())

# Standardize the dataset (important for PCA)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce to 2 dimensions
pea= PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a scatter plot of PCA results
plt.figure(figsize=(8, 6))
colors = ['red', 'blue', 'green']

for i, target in enumerate(np.unique(y)):
    plt.scatter(X_pca[y == target, 0], X_pca[y == target, 1],
                color=colors[i], label=target_names[i], alpha=0.7, edgecolors=

# Plot settings
plt.xlabel("Principal Component  1")
plt.ylabel("Principal Component 2")
plt.title("PCA of Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

```
Dataset shape: (150, 4)
First 5 rows:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```



In [10]:
```
# 34
# Apply Linear Discriminant Analysis (LDA) to the Iris dataset to reduce its a
# You will visualize the LOA-reduced dataset in a 2D scatter plot using Matplo
# instructions:
# Load the Iris dataset
# Briefly explore the dataset to understand its structure. Print the shape of
# the first few rows to get an overview of the features and target variable.
# Initialize the PCA object to reduce the dataset to 2 dimensions.
# Fit the LDA model to the Iris dataset and transform the dataset
# Use Matplotlib to create a scatter plot of the LOA-reduced dataset.
# Color the points based on their respective species (Setosa, Versicolor, Virg
# visualize how well the LDA has separated the different classes.
# Add appropriate titles and Labels to the axes of the plot.
# Include a Legend to indicate which colors correspond to which species.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from  sklearn.discriminant_analysis  import  LinearDiscriminantAnalysis  as  LDA

# Load the Iris dataset
iris= datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target Labels
target_names = iris.target_names  # Class names

# Explore the dataset
print("Dataset shape:", X.shape)
print("First 5 rows:\n", pd.DataFrame(X, columns=iris.feature_names).head())

# Apply LDA to reduce dimensions to 2
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X, y)

# Create a scatter plot of LDA results
plt.figure(figsize=(8, 6))
colors = ['red', 'blue', 'green']

for i, target in enumerate(np.unique(y)):
    plt.scatter(X_lda[y == target, 0], X_lda[y == target, 1],
                color=colors[i], label=target_names[i], alpha=0.7, edgecolors=

# Plot settings
plt.xlabel("LDA Component  1")
plt.ylabel("LDA Component 2")
plt.title("LDA of Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

In [11]:
```
# 35
# Apply Linear Discriminant Analysis (LDA) to the Wine dataset to reduce its a
# You will visualize the LOA-reduced dataset in a 2D scatter plot and evaluate
# Dataset: You will use the Wine dataset, which consists of 178 samples of win
#      each described by 13 features representing different chemical properties
#     The target variable indicates the type of wine, which can take on one of
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Wine dataset
wine= datasets.load_wine()
X = wine.data  # Features (13 chemical properties)
y = wine.target  # Target Labels (wine classes: 0, 1, 2)
target_names = wine.target_names  # Class names

# Explore dataset
print("Dataset shape:", X.shape)
print("First 5 rows:\n", pd.DataFrame(X, columns=wine.feature_names).head())

# Standardize the features (LDA is sensitive to scale)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LDA to reduce dimensions to 2
lda = LDA(n_components=2)
X_lda  =  lda.fit_transform(X_scaled, y)

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, r

#Traina Random Forest classifier
elf= RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualizing LOA-reduced data
plt.figure(figsize=(8, 6))
colors = ['red', 'blue', 'green']

for i, target in enumerate(np.unique(y)):
    plt.scatter(X_lda[y == target, 0], X_lda[y == target, 1],
                color=colors[i], label=target_names[i], alpha=0.7, edgecolors=

# Plot settings
```

```
plt.xlabel("LDA Component 1")
plt.ylabel("LDA Component 2")
plt.title("LDA of Wine Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

```
Dataset shape: (178, 13)
First 5 rows:
    alcohol  malic acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue
\
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines   praline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
4                          2.93    735.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

In [14]:
```
#36
# apply Linear Discriminant Analysis (LDA) to the Iris dataset using the Seiki
# You will preprocess the data using Label encoding, perform LDA to reduce the
# and visualize the results in a 2D scatter plot.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Load the Iris dataset
iris= datasets.load_iris()
X = iris.data  # Features (4D)
y = iris.target  # Target Labels (e, 1, 2)
target_names = iris.target_names  # Class names

# Standardize the features (LDA is affected by scale)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LDA to reduce dimensionality to 2D
lda = LDA(n_components=2)
X_lda  =  lda.fit_transform(X_scaled, y)

# Visualizing LOA-transformed data in a scatter plot
plt.figure(figsize=(8, 6))
colors = ['red', 'blue', 'green']

for i, target in enumerate(np.unique(y)):
    plt.scatter(X_lda[y == target, 0], X_lda[y == target, 1],
                color=colors[i], label=target_names[i], alpha=0.7, edgecolors=

# Plot settings
plt.xlabel("LDA Component  1")
plt.ylabel("LDA Component 2")
plt.title("LDA of Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
```

In [15]:
```
#37
#Objective: The goal of this practical exam is to apply Linear Discriminant A
#     the Breast Cancer dataset using the Scikit-Learn Library.
#     You will reduce the dimensionality of the dataset from multiple features
#     visualize the results using Matplotlib.
#•Load the Breast Cancer dataset
#•initialize the LOA to reduce the dataset to 1 dimension.
#•Fit the LOA model to the Breast Cancer dataset and transform the dataset t
# Visualization
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the breast cancer dataset
data= load_breast_cancer()
X, y = data.data, data.target

# Standardize the data
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LOA with 1 component
lda = LinearDiscriminantAnalysis(n_components=1)
X_lda = lda.fit_transform(X_scaled,y)

# Plot the LOA-reduced dataset
plt.figure(figsize=(8,  5))
plt.scatter(X_lda, np.zeros_like(X_lda), c=y, cmap='coolwarm', alpha=0.7, edgE

#The x-coordinates come from X_Lda (Likely a 10 projection of your data such
#They-coordinates are all 0 (effectively placing the points along a horizonta
#The color of each point corresponds to its class Label (y) with the colors c
#The transparency of the points is set to 70% (alpha=0.7)  and the points have

plt.xlabel('LDA Component  1')
plt.title('LDA Reduction of Breast Cancer Dataset')
plt.yticks([])
plt.colorbar(label='Class Label')
plt.show()
```

In [16]:
```
#38
# Dataset: You will use the Iris dataset, which consists of 156 samples of iri
#     each described by 4 features (sepal Length, sepal width, petal Length, a
#     The target variable indicates the species of the iris flower (Setosa, Ve
#     apply both Linear Discriminant Analysis (LOA) and Principal Component An
#     You will reduce the dimensionality of the dataset using both techniques
#     You will then compare the effectiveness of LOA and PCA in terms of class
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Load the Iris dataset
iris= datasets.load_iris()
X = iris.data  # Features (40)
y = iris.target  # Target Labels (6, 1, 2)
target_names = iris.target_names  # Class names
colors = ['red', 'blue', 'green']

# Standardize the features (important for PCA)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA (40  20)
pea= PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Apply LOA (40  20)
lda = LDA(n_components=2)
X_lda  =  lda.fit_transform(X_scaled, y)

# Create subplots
fig,  axes= plt.subplots(1, 2, figsize=(12, 6))

# PCA Plot
axes[0].scatter(X_pca[y -- 0, 0], X_pca[y  -- 0, 1], color=colors[0], label=tar
axes[0].scatter(X_pca[y -- 1, 0], X_pca[y  -- 1, 1], color=colors[1], label=tar
axes[0].scatter(X_pca[y -- 2, 0], X_pca[y  -- 2, 1], color=colors[2], label=tar
axes[0].set_title("PCA of Iris Dataset")
axes[0].set_xlabel("Principal  Component  1")
axes[0].set_ylabel("Principal Component 2")
axes[0].legend()
axes[0].grid(True)

# LOA Plot
axes[1].scatter(X_lda[y -- 0, 0], X_lda[y  -- 0, 1], color=colors[0], label=tar
axes[1].scatter(X_lda[y -- 1, 0], X_lda[y  -- 1, 1], color=colors[1], label=tar
axes[1].scatter(X_lda[y -- 2, 0], X_lda[y  -- 2, 1], color=colors[2], label=tar
axes[1].set_title("LDA of Iris Dataset")
axes[1].set_xlabel("Linear Discriminant  1")
axes[1].set_ylabel("Linear Discriminant 2")
axes[1].legend()
axes[1].grid(True)

plt.show()
```

In [17]:
```
# 39
# develop a Linear regression model to predict house prices based on various f
# You will use a dataset that contains information about houses, including fea
# number of bedrooms, square footage, and Location. You will evaluate the mode
# Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.
# Show performance of same model when PCA reduced data set is used.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2 score
from sklearn.decomposition import PCA

# Load dataset
df = pd.read_csv('HousePrices.csv')

# Drop unnecessary column
df = df.drop(columns=['Home'])

# One-hot encode categorical variables
df = pd.get_dummies(df, columns=['Brick', 'Neighborhood'], drop_first=True)

# Split features and target
X = df.drop(columns=['Price'])
y = df['Price']

# Standardize features
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2

# Train Linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predictions and evaluation
y_pred = lr.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Linear Regression  without  PCA:")
print(f"MAE:{mae:.2f}, MSE: {mse:.2f}, R² {r2:.4f}")

# Apply PCA (retain 95% variance)
pea= PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train Linear regression with PCA-transformed data
lr_pca = LinearRegression()
lr_pca.fit(X_train_pca, y_train)
y_pred_pca = lr_pca.predict(X_test_pca)

# Evaluate PCA model
```

```
mae_pca =mean_absolute_error(y_test, y_pred_pca)
mse_pca = mean_squared_error(y_test, y_pred_pca)
r2_pca = r2_score(y_test, y_pred_pca)

print("\nlinear    Regression    with    PCA:")
print(f"MAE:{mae_pca:.2f}, MSE: {mse_pca:.2f}, R²: {r2_pca:.4f}")
print(f"PCA reduced features to {pca.n_components_} components")
```

```
Linear Regression without PCA:
MAE: 8901.29, MSE: 114170418.45, R²: 0.8063

Linear Regression with PCA:
MAE: 8754.73, MSE: 139629218.23, R²: 0.7631
PCA reduced features to 6 components
```

In [18]:
```
# 40
# apply Principal Component Analysis (PCA) to a house price prediction dataset
# then use a regression model to predict house prices. You will evaluate them
# performance using metrics such as Mean Absolute Error (MAE) Mean Squared Er
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2 score

# Load dataset
df = pd.read_csv('HousePrices.csv')

# Drop unnecessary column
df = df.drop(columns=['Home'])

# One-hot encode categorical variables
df = pd.get_dummies(df, columns=['Brick', 'Neighborhood'], drop_first=True)

# Split features and target
X = df.drop(columns=['Price'])
y = df['Price']

# Standardize features
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA (retain 95% variance)
pea= PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, r

# Train regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predictions and evaluation
y_pred = lr.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print results
print("Linear Regression with PCA:")
print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"R²: {r2:.4f}")
print(f"PCA reduced features to {pca.n_components_} components")
```

In [19]:
```
# 41
# Consider SociaL_Netwok_Ads.csv dataset - (UserID, Gender, Age, EstimatedSaLa
# Use Age and EstimatedSaLary as input features and Purchased as target featur
# Split test data set 36% of complete dataset. Build two models of support vec
# python using skLearn Library, one for Linear and another for RBF kernel with
#  Predict test Labels and print test accuracy.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset
file_path = "Social_Network_Ads.csv"
df = pd.read_csv(file_path)

# Select features and target variable
X = df[['Age', 'EstimatedSalary']]
y = df['Purchased']

# Standardize features
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA (retain 95% variance)
pea= PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.3, r

# Train SVM with Linear kernel
svc_linear = SVC(kernel='linear', C=1.0)
svc_linear.fit(X_train, y_train)
y_pred_linear = svc_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)

# Train SVM with RBF kernel
svc_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
svc_rbf.fit(X_train, y_train)
y_pred_rbf = svc_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

# Print results
print("SVM with Linear Kernel:")
print(f"Accuracy:{accuracy_linear:.2f}")

print("\nSVM with RBF Kernel:")
print(f"Accuracy:{accuracy_rbf:.2f}")

print(f"PCA reduced features to {pca.n_components_} components")
```

In [20]:
```
# 42
# Develop a Support Vector Classifier to predict whether a tumor is malignant
# 30  features computed from a digitized image of a fine needle aspirate (FNA)
# The dataset contains 569 samples with binary Labels indicating tumor type. (
# Load the Dataset from sklearn.Split the dataset into training and testing se
# Visualize the Results.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification r

# Load the dataset
data= load_breast_cancer()
X = data.data
y = data.target

# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Standardize the features
scaler= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Support Vector Classifier
svc = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
svc.fit(X_train, y_train)

# Make predictions
y_pred = svc.predict(X_test)

# Evaluate the model
accuracy= accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")

# Confusion Matrix
cm= confusion_matrix(y_test,  y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Mal
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Test Accuracy: 0.98
```

In [21]:
```
# 43
# Write a Python program to implement SVM classification for breast cancerprea
#     Data preprocessing using StandardScaler use kernel parameters for tuning
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset
data= load_breast_cancer()
X = data.data
y = data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Standardize features
scaler= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM model with kernel parameters
svc = SVC(kernel='rbf', C=1.0, gamma='scale')
svc.fit(X_train, y_train)

# Predict and evaluate
y_pred = svc.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)

# Print results
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Test Accuracy: 0.98
```

In [22]:
```
# 44
# The breast cancer dataset is available in the sklearn.datasets module and ca
# implement a Naive Bayes Classifier on the Breast Cancer Dataset using python
# Assume that all features of datasets are continuous variables and must be us
# Perform following task. Load data set, Split data for training and testing,
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the dataset
data= load_breast_cancer()
X = data.data
y = data.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Build the model
model= GaussianNB()
model.fit(X_train, y_train)

# Predict Labels for the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9736842105263158
```

In [26]:
```
#45
# Implement a Gaussian Naive Bayes classifier to predict whether a patient has
# metrics of PIMA.csv dataset. The dataset consists of information from 768 fe
# initially gathered by the National Institute of Diabetes and Digestive and K
# Target variable: Diabetes (binary, 6 or 1) Attributes: Pregnancies, OGTT (Or
# Blood pressure, Skin thickness, Insulin, BMI (Body Mass Index), Age, Pedigre
# Load the Dataset from a CSV file. Provide summary statistics for the dataset
# Split the data into training and testing sets (86% train, 26% test).
# Instantiate a Gaussian Naive Bayes model and fit it on the training data.
# Predict diabetes status for the test set. Discuss any biases in the dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the dataset
data= pd.read_csv('PIMA.csv') # Ensure PIMA.csv is in your working directory

# Display summary statistics
print(data.describe())

# Define features and target
X = data.drop(columns=['Outcome'])
y = data['Outcome']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Build the model
model= GaussianNB()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

In [27]:
```
# 46
# Build a Naive Bayes classifier to analyze the sentiment of movie reviews (po
# The IMDB_Dataset.csv typically contains two main columns: Review: The text o
# A Label indicating the sentiment of the review, usually categorized as 'posi
# Load the dataset and inspect the structure. Clean the text data by removing
# and tokenizing the reviews.Use CountVectorizer or TfidfVectorizer to convert
# numerical format suitable for model training. Split the dataset and Create a
# fit it on the training data. Predict the sentiment of the test reviews.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load dataset
data= pd.read_csv('IMDB_Dataset.csv')

# Preprocess text data
data['Review'] =  data['Review'].str.replace('[A\w\s]', '').str.lower()

# Convert text to numerical format
vectorizer  =  CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(data['Review'])
y  =  data['Sentiment'].map({'positive': 1, 'negative': 0})

# Split data
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.2, randc

# Build and evaluate the model
model= MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
~\anaconda3\lib\site-packages\pandas\io\common.py in get_handle(path_or_buf,
mode, encoding, compression, memory_map, is_text, errors, storage_options)
    784         if ioargs.encoding and "b" not in ioargs.mode:
    785             # Encoding
--> 786             handle= open(
    787                 handle,
    788                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'IMDB_Dataset.csv'
```

In [3]:

```
# 47
# Use sklearn.datasets.fetch_20newsgroups() dataset (all sets) to classify new
# respective categories using a Multinomial Naive Bayes classifier. Load the a
# Preprocess the text data using CountVectorizer to convert text documents int
# Split the dataset and Create a Multinomial Naive Bayes model. Fit the model
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load dataset
newsgroups= fetch_20newsgroups(subset='all')
X = newsgroups.data
y = newsgroups.target

# Convert text to numerical format
vectorizer = CountVectorizer(stop_words='english')
X_vec = vectorizer.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_vec, y, test_size=0.2, r
# Build and evaluate the model
model= MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8742705570291777
```

In [4]:
```
# 48
# Imagine a telecommunications company that wants to predict whether a custome
# based on various features such as age, account Length, and monthly charges.
# The company has historical data on customers, including whether they churned
# Age: (in years), Account_Length: (in months), Monthly_Charges: (in dollars)
# Churn: Target variable (1 if the customer churned, 0 otherwise)
# 'Age': [25, 34, 45, 29, 50, 38, 42, 35, 48, 55},
# 'Account_Length': [12, 24, 36, 18, 48, 30, 42, 24, 36, 60},
# 'Monthly_Charges': [70, 90, 80, 100, 60, 80, 90, 70, 80, 100},
# 'Churn': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1}
# build a Logistic regression model using python that can predict the probabil
# by spliting 80% of given data for training. Predict targets on trained model
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Data
data= {
'Age':[25, 34, 45, 29, 50, 38, 42, 35, 48, 55],
'Account_Length':[12, 24, 36, 18, 48, 30, 42, 24, 36, 60],
'Monthly_Charges':[70, 90, 80, 100, 60, 80, 90, 70, 80, 100],
'Churn':[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
}
df = pd.DataFrame(data)

X = df[['Age', 'Account_Length', 'Monthly_Charges']]
y = df['Churn']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc
# Model
model= LogisticRegression()
model.fit(X_train, y_train)
# Predictions
y_pred = model.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)
print(accuracy)
```

```
0.5
```

In [ ]:
```
# 49
# build a Logistic regression model that can predict whether a customer is Lik
# features in file named customer_churn.csv.
# customer_id: Unique customer ID  age: Customer age
# gender: Customer gender (male/female)
# account_Length: Length of the customer's account (in months)
# international_plan: Whether the customer has an international plan (yes/no)
# voice_mail_plan: Whether the customer has a voice mail plan (yes/no)
# number_vmail_messages: Number of voice mail messages
# total_day_calls: Total day calls
# total_night_calls: Total night calls
# total_intl_calls: Total international calls
#•churn: Whether the customer churned (yes/no)
# Load data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df = pd.read_csv("customer_churn.csv")

# Preprocess data
df = pd.get_dummies(df, drop_first=True)

# Split features and target
X = df.drop('churn', axis=1)
y = df['churn']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Model
model= LogisticRegression()
model.fit(X_train, y_train)

# Predictions and accuracy
y_pred = model.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)
print(accuracy)
```

In [6]:

```
#50
# Use iris dataset for creating a binary classification problem that predicts
# the species "Iris-Virginica" or not. Dataset Features: sepal_length, sepal_w
# species: Species of the iris flower (Iris-setosa, Iris-versicolor, Iris-virg
# Make Predictions and Print Probabilities with Alter Threshold to 0.6.
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load dataset
data= load_iris()
X = data.data
y = (data.target -- 2).astype(int)  # Binary classification: 1 if Iris-Virgini

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Standardize features
scaler= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression model
model= LogisticRegression()
model.fit(X_train, y_train)

# Predict probabilities
y_prob  =  model.predict_proba(X_test)[:, 1]

# Apply threshold 0.6
y_pred = (y_prob >= 0.6).astype(int)

# Evaluate model
accuracy= accuracy_score(y_test,  y_pred)

# Print results
print(f"Test Accuracy: {accuracy:.2f}")
print("Predicted Probabilities:", y_prob)
print("Predicted Labels with 0.6 Threshold:", y_pred)
```

In [11]:
```
# 51
# Consider Salary.csv  with years_of_experience and salary. Write a python co
# simple Linear regression with independent variable years_of_experience and a
# Use complete dataset to train model. Plot :fitted model along with trained a
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear model import LinearRegression

# Load dataset
file_path = "Salary.csv"
df = pd.read_csv(file_path)

df = df.dropna()  # Remove rows with NaN values


# Extract independent and dependent variables
X = df[['Vears of Experience']]
y = df['Salary']

# Train Linear Regression model
model= LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Plot fitted model with trained data points
plt.figure(figsize=(10,  5))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label='Fitted Line')
plt.xlabel("Vears of Experience")
plt.ylabel("Salary")
plt.title("Linear Regression - Salary vs Experience")
plt.legend()
plt.show()

# Residual plot
residuals= y - y_pred
plt.figure(figsize=(10, 5))
sns.residplot(x=X.values.flatten(), y=residuals, lowess=True, line_kws={"color
plt.xlabel("Vears of Experience")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

In [13]:
```
#52
# Consider a dataset HousePrices.csv with columns square_feet and price.
# Write a Python code to fit a polynomial regression model with square_feet as
# price as the dependent variable. Use the complete dataset to train the model
# Plot the fitted polynomial model along with the trained data points and crea
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Load dataset
file_path = "HousePrices.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Extract independent and dependent variables
X = df[['SqFt']]
y = df['Price']

# Transform features to polynomial degree 2
poly= PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Train Polynomial Regression model
model= LinearRegression()
model.fit(X_poly, y)

# Predictions
y_pred = model.predict(X_poly)

# Plot fitted polynomial model with trained data points
plt.figure(figsize=(10,  5))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(np.sort(X.values, axis=0), model.predict(poly.transform(np.sort(X.val
plt.xlabel("Square Feet")
plt.ylabel("Price")
plt.title("Polynomial Regression - House Prices")
plt.legend()
plt.show()

# Residual plot
residuals= y - y_pred
plt.figure(figsize=(10, 5))
sns.residplot(x=X.values.flatten(), y=residuals, lowess=True, line_kws={"color
plt.xlabel("Square Feet")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

In [ ]:
```
# 53
# Consider a dataset CarPrices.csv with columns age, mileage, and price.
# Write a Python code to fit a multiple Linear regression model with age and m
# and price as the dependent variable. Use the complete dataset to train them
# Plot the actual prices against the predicted prices and create a residual pl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear model import LinearRegression

# Load dataset
file_path = "CarPrices.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Extract independent and dependent variables
X = df[['age', 'mileage']]
y = df['price']

# Train Multiple Linear Regression model
model= LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Plot actual vs predicted prices
plt.figure(figsize=(10,  5))
plt.scatter(y, y_pred, color='blue', label='Predicted vs Actual')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='dashe
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Prices")
plt.legend()
plt.show()

# Residual plot
residuals= y - y_pred
plt.figure(figsize=(10, 5))
sns.residplot(x=y_pred, y=residuals, lowess=True, line_kws={"color": "red"})
plt.xlabel("Predicted Price")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

In [17]:
```
# 54
# Consider a dataset HousePrices.csv with features such as size (in square fee
# a target column price (in dollars). Write a Python code to implement Linear
# Gradient Descent to predict price based on size. Use the complete dataset to
# Plot the regression Line along with the training data points.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor

# Load dataset
file_path = "HousePrices.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Extract independent and dependent variables
X = df[['SqFt']].values  # Feature (square feet)
y = df['Price'].values   # Target (price)

# Standardize the feature
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train Linear Regression using Gradient Descent
model= SGDRegressor(max_iter=1000, learning_rate='optimal', eta0=0.01)
model.fit(X_scaled, y)

# Predictions
y_pred = model.predict(X_scaled)

# Plot regression Line with training data points
plt.figure(figsize=(10,  5))
plt.scatter(df['SqFt'], y, color='blue', label='Actual Data')
plt.plot(df['SqFt'], y_pred, color='red', label='Regression Line')
plt.xlabel("Size (Square Feet)")
plt.ylabel("Price  (Dollars)")
plt.title("Linear Regression using SGDRegressor")
plt.legend()
plt.show()
```

In [18]:
```
# 55
# Consider a dataset CarPrices.csv with features such as horsepower, age, and
# Write a Python code to fit a Linear regression model to predict price based
# evaluate the model using MSE, R², and MAE.
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score

# Load dataset
file_path = "CarPrices.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Extract independent and dependent variables
X = df[['horsepower', 'months_old']]
y = df['price']

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Linear Regression model
model= LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 2220314.77
Mean Absolute Error (MAE): 1168.43
R² Score: 0.83
```

In [23]:
```
#56
# Consider a dataset HouseData.csv with features such as num_rooms, square_fee
# a target column house_price. Write a Python code to fit a Linear regression
# house_price based on the features and evaluate the model using Mean Squared
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score
from sklearn.preprocessing import OneHotEncoder

# Load dataset
file_path = "HouseData.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Separate numerical and categorical features
X_numerical = df[['Bedrooms', 'SqFt']]
X_categorical = df[['Neighborhood']]
y = df['Price']

# One-hot encode categorical variables
encoder= OneHotEncoder(drop='first', sparse_output=False)
X encoded= encoder.fit_transform(X_categorical)

# Combine numerical and encoded categorical features
X = np.hstack((X_numerical, X_encoded))

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Linear Regression model
model= LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 276452473.43
Mean Absolute Error (MAE): 12598.91
R² Score: 0.53
```

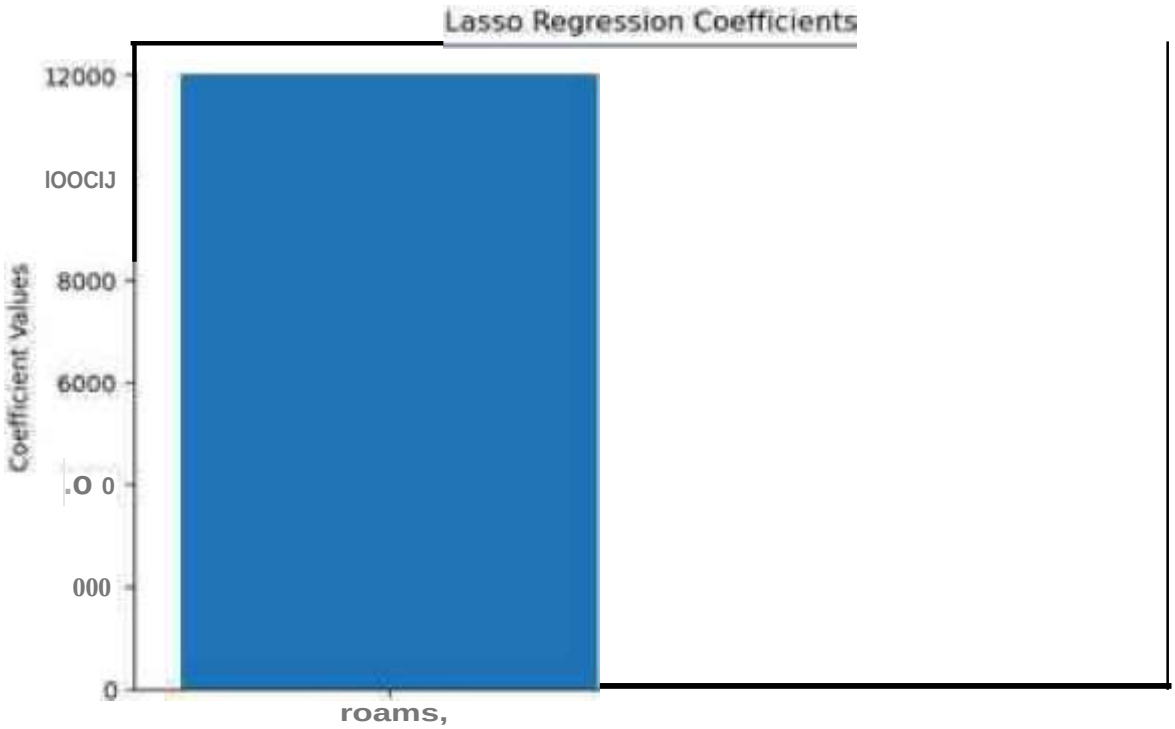
In [31]:
```
# 57
# Consider a dataset EmployeeData.csv with features such as years_of_experienc
# and a target column salary. Write a Python code to fit a Linear regression m
# the features and evaluate the model using Mean Squared Error (MSE) R-square
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score

# Load dataset
file_path = "Salary.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Convert categorical variables to numerical using one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True) # Converts categorical colun

# Extract independent and dependent variables
X = df_encoded.drop(columns=['Salary']) # Assuming 'salary' is the target col
y = df_encoded['Salary']

# Split dataset into training and testing sets (80% train 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Linear Regression model
model= LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")
```

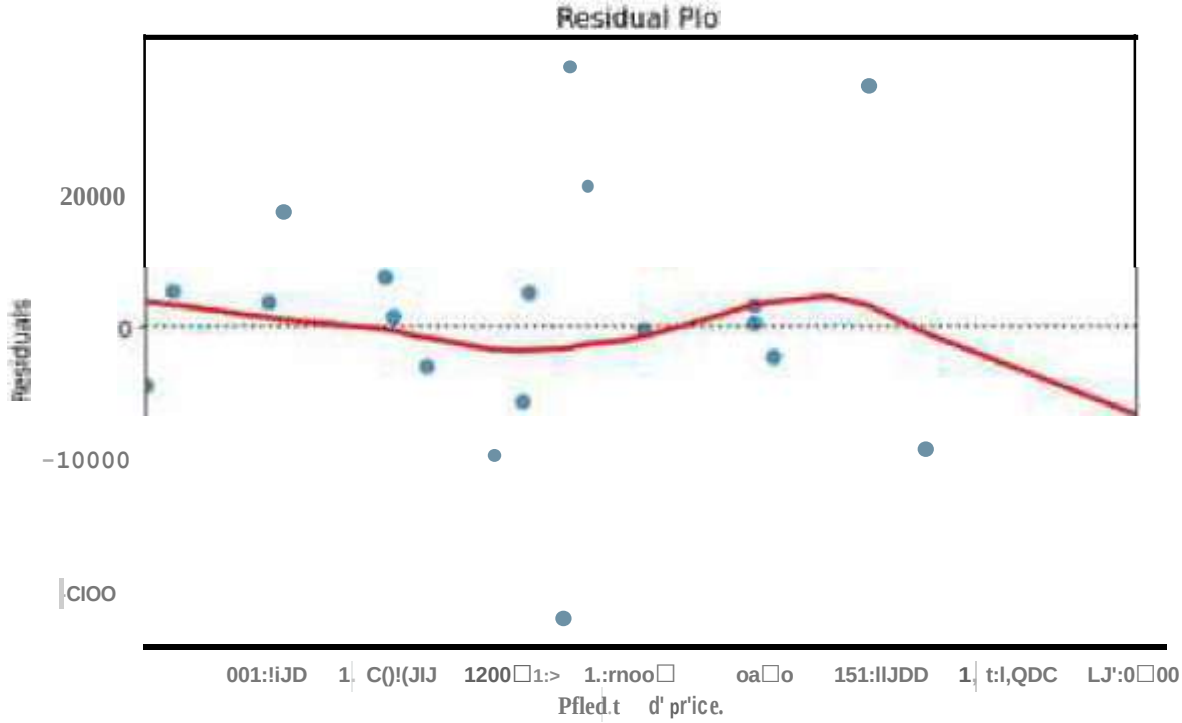
```
Mean Squared Error (MSE): 55031761374690635532685153402880.00
Mean Absolute Error (MAE): 1671690182167788.25
R² Score: -22953023618803744899072.00
```

In [32]:
```
# 58
# Consider a dataset HousingData.csv with columns num_rooms  area age and a
# Write a Python code to fit a Lasso regression model using num_rooms area a
# variables to predict price. Use the complete dataset to train the model.
# Plot the coefficients of the features and create a residual plot.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score

# Load dataset
file_path = "HousingData.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Define independent variables (features) and dependent variable (target)
X = df[['Bedrooms', 'SqFt', 'age']]
y = df['Price']

# Split dataset into training and testing sets (80% train  20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Lasso Regression model
lasso= Lasso(alpha=0.1)  # You can tune alpha for regularization strength
lasso.fit(X_train,  y_train)

# Predictions
y_pred = lasso.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")

# Plot feature coefficients
plt.figure(figsize=(8, 5))
plt.bar(X.columns, lasso.coef_)
plt.xlabel("Features")
plt.ylabel("Coefficient Values")
plt.title("Lasso Regression Coefficients")
plt.show()

# Residual plot
residuals= y_test - y_pred
plt.figure(figsize=(8,  5))
sns.residplot(x=y_pred, y=residuals, lowess=True, line_kws={"color": "red"})
plt.xlabel("Predicted Price")
```

```
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

```
Mean Squared Error (MSE): 350174259.25
Mean Absolute Error (MAE): 14814.16
R² Score: 0.41
```

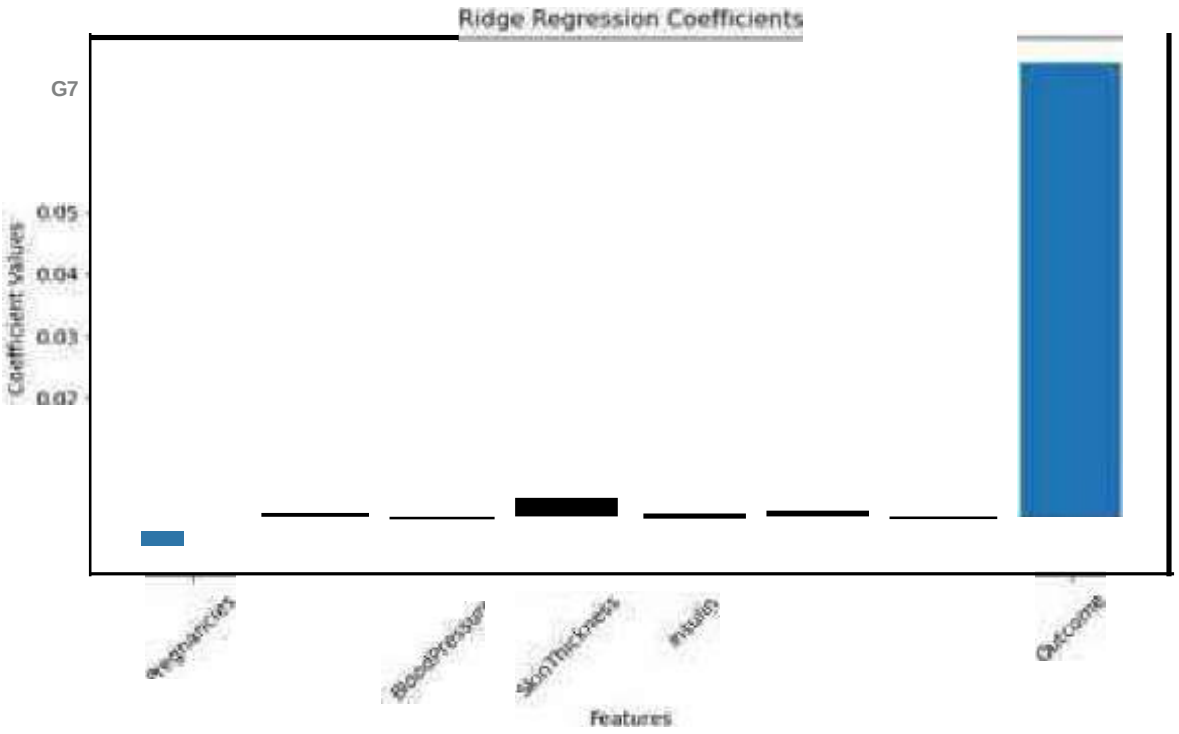


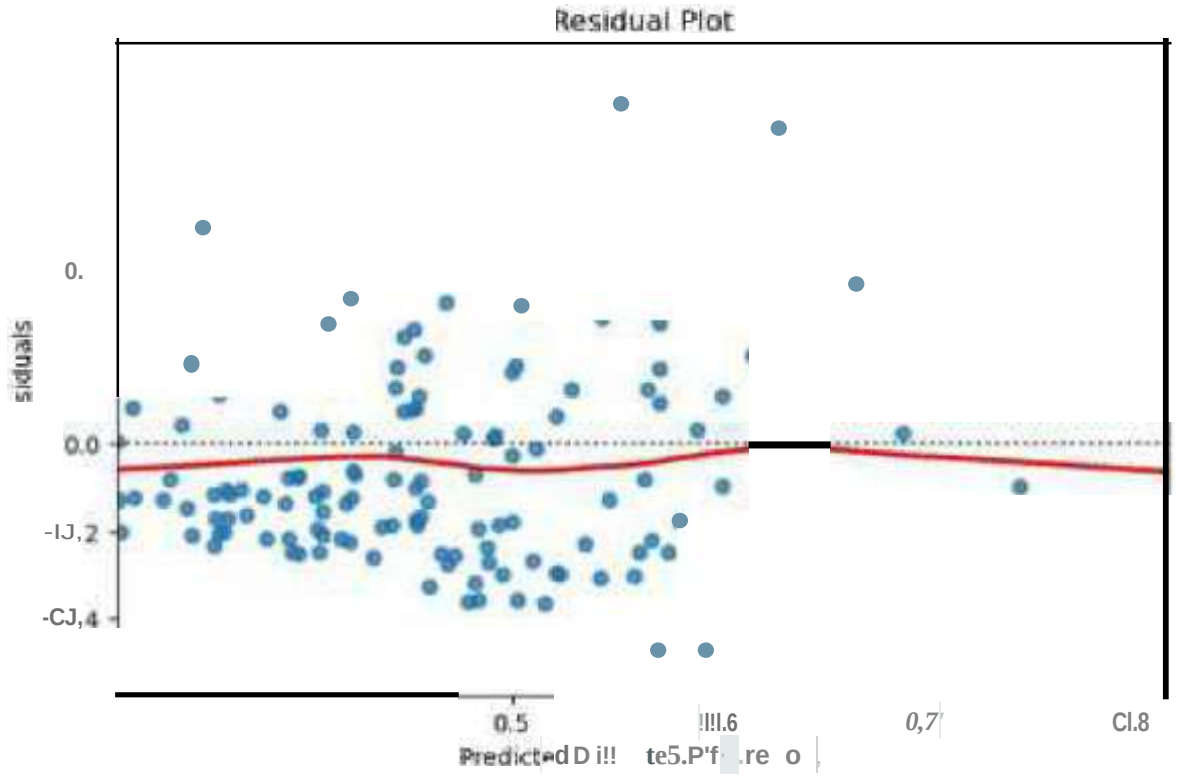


In [34]:
```
#59
# Consider a dataset Diabetes.csv with various medical attributes and a target
# Write a Python code to fit a Ridge regression model using all available feat
# Use the complete dataset to train the model. Plot the coefficients of the fe
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score

# Load dataset
file_path = "Diabetes.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Define independent variables (features) and dependent variable (target)
X = df.drop(columns=['DiabetesPedigreeFunction']) # Assuming 'diabetes_progrE
y = df['DiabetesPedigreeFunction']

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Ridge Regression model
ridge= Ridge(alpha=1.0)  # Regularization strength
ridge.fit(X_train,  y_train)

# Predictions
y_pred = ridge.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")

# Plot feature coefficients
plt.figure(figsize=(10, 5))
plt.bar(X.columns, ridge.coef_)
plt.xlabel("Features")
plt.ylabel("Coefficient Values")
plt.title("Ridge Regression Coefficients")
plt.xticks(rotation=45)
plt.show()

# Residual plot
residuals= y_test - y_pred
plt.figure(figsize=(8,  5))
sns.residplot(x=y_pred, y=residuals, lowess=True, line_kws={"color": "red"})
plt.xlabel("Predicted Diabetes Progression")
```

```
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

```
Mean Squared Error (MSE): 0.08
Mean Absolute Error (MAE): 0.23
R² Score: 0.12
```



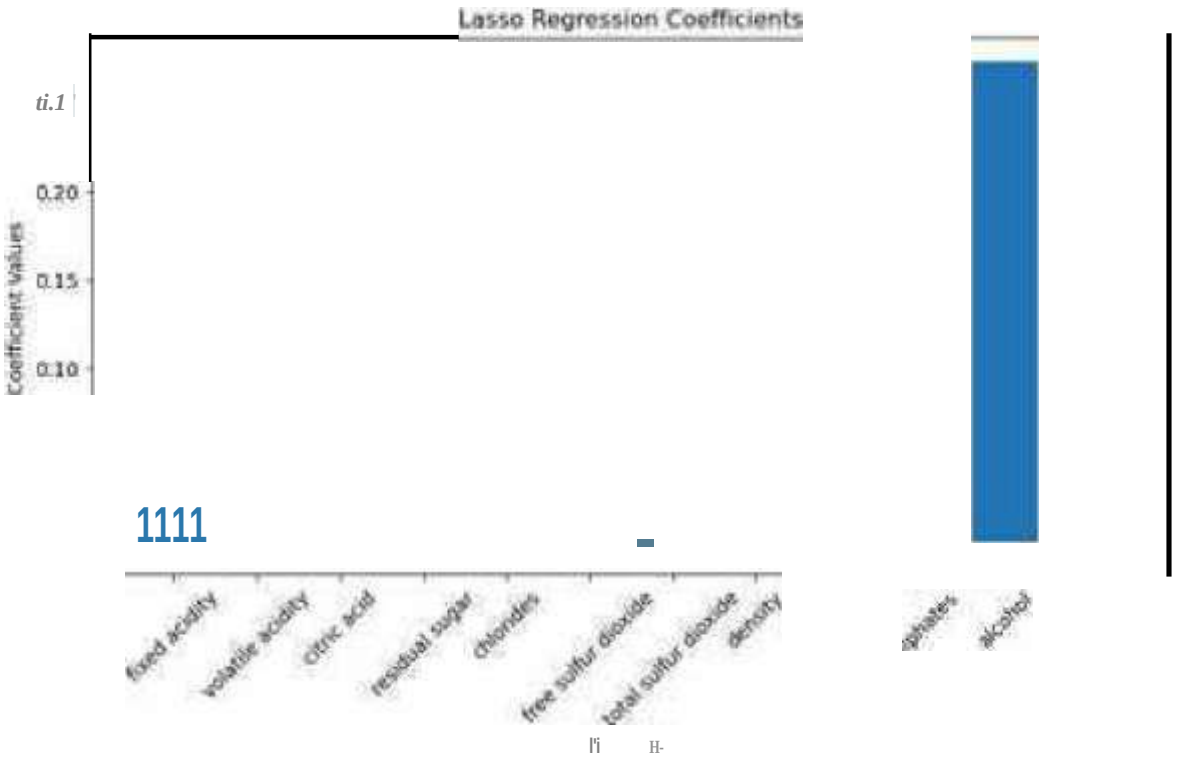


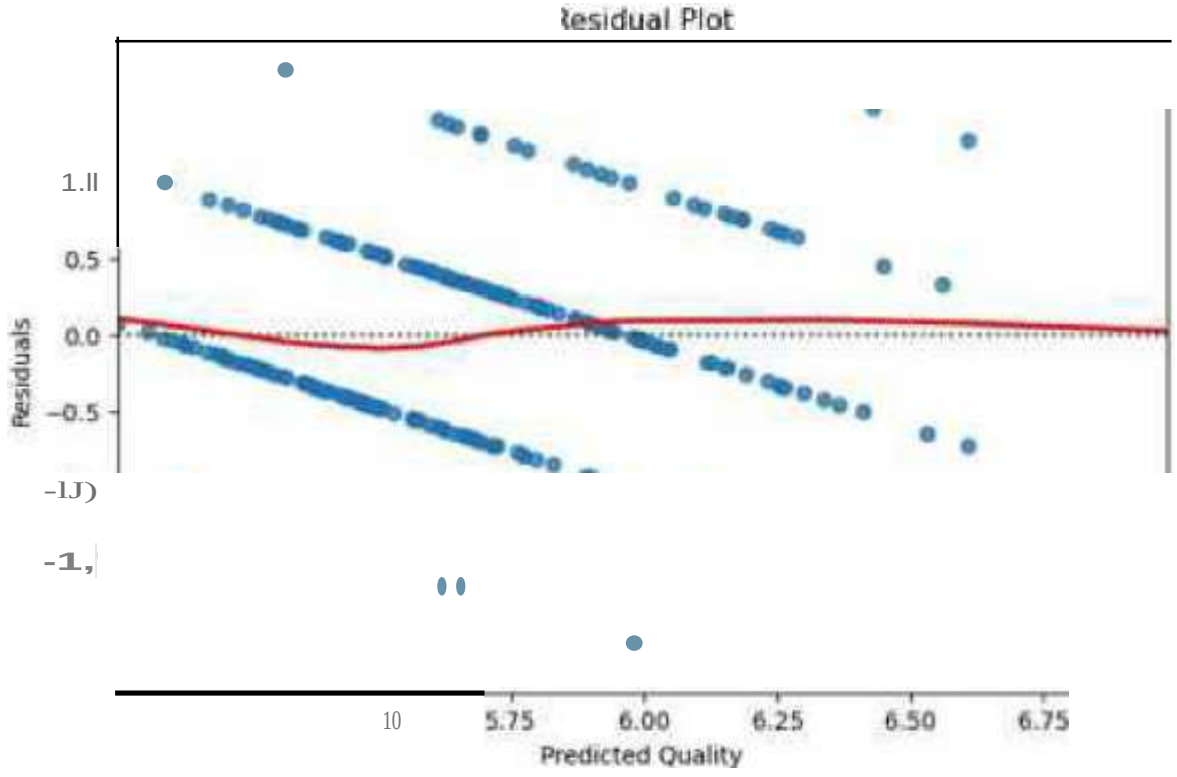
In [35]:

```
#60
# Consider a dataset WineQuality.csv with various chemical properties of wine
# Write a Python code to fit a Lasso regression model using all available feat
# Use the complete dataset to train the model. Plot the coefficients of the fe
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score

# Load dataset
file_path = "WineQuality.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Define independent variables (features) and dependent variable (target)
X = df.drop(columns=['quality']) # Assuming 'quality' is the target column
y = df['quality']

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Lasso Regression model
lasso= Lasso(alpha=0.1) # Adjust alpha for regularization strength
lasso.fit(X_train, y_train)

# Predictions
y_pred = lasso.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")

# Plot feature coefficients
plt.figure(figsize=(10, 5))
plt.bar(X.columns, lasso.coef_)
plt.xlabel("Features")
plt.ylabel("Coefficient Values")
plt.title("Lasso Regression Coefficients")
plt.xticks(rotation=45)
plt.show()

# Residual plot
residuals= y_test - y_pred
plt.figure(figsize=(8, 5))
sns.residplot(x=y_pred, y=residuals, lowess=True, line_kws={"color": "red"})
plt.xlabel("Predicted Quality")
```

```
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

```
Mean Squared Error (MSE): 0.41
Mean Absolute Error (MAE): 0.51
R² Score: 0.27
```

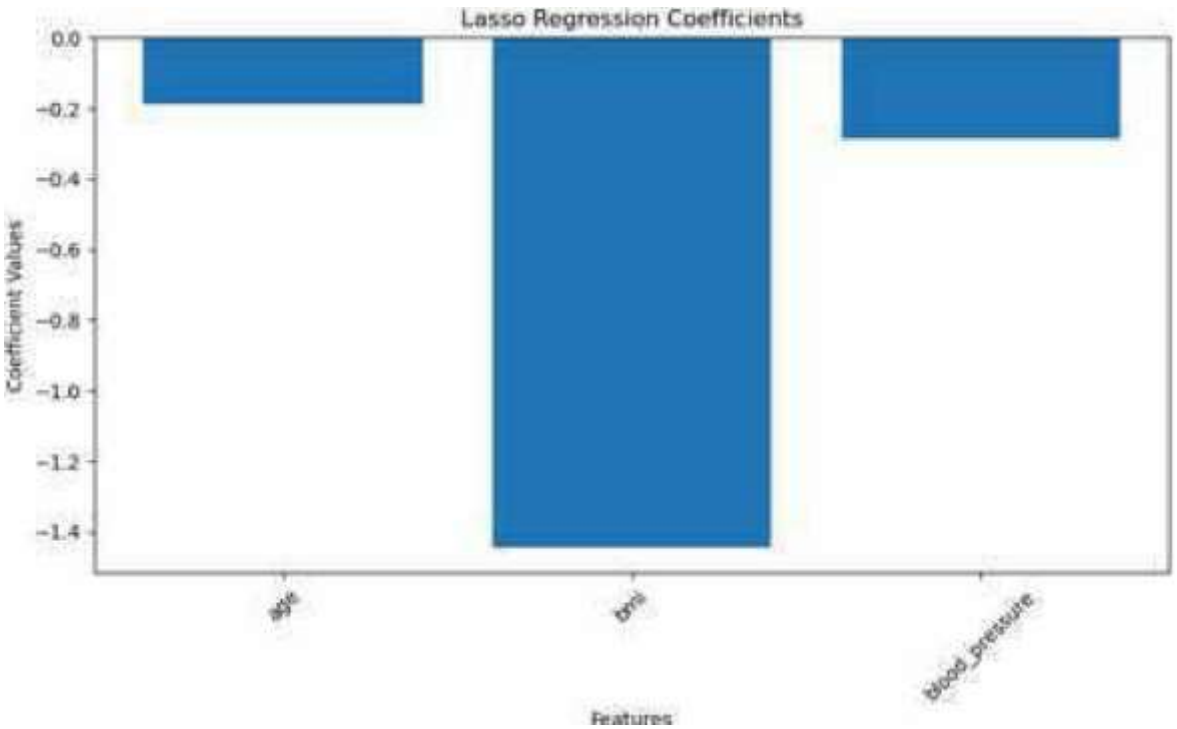


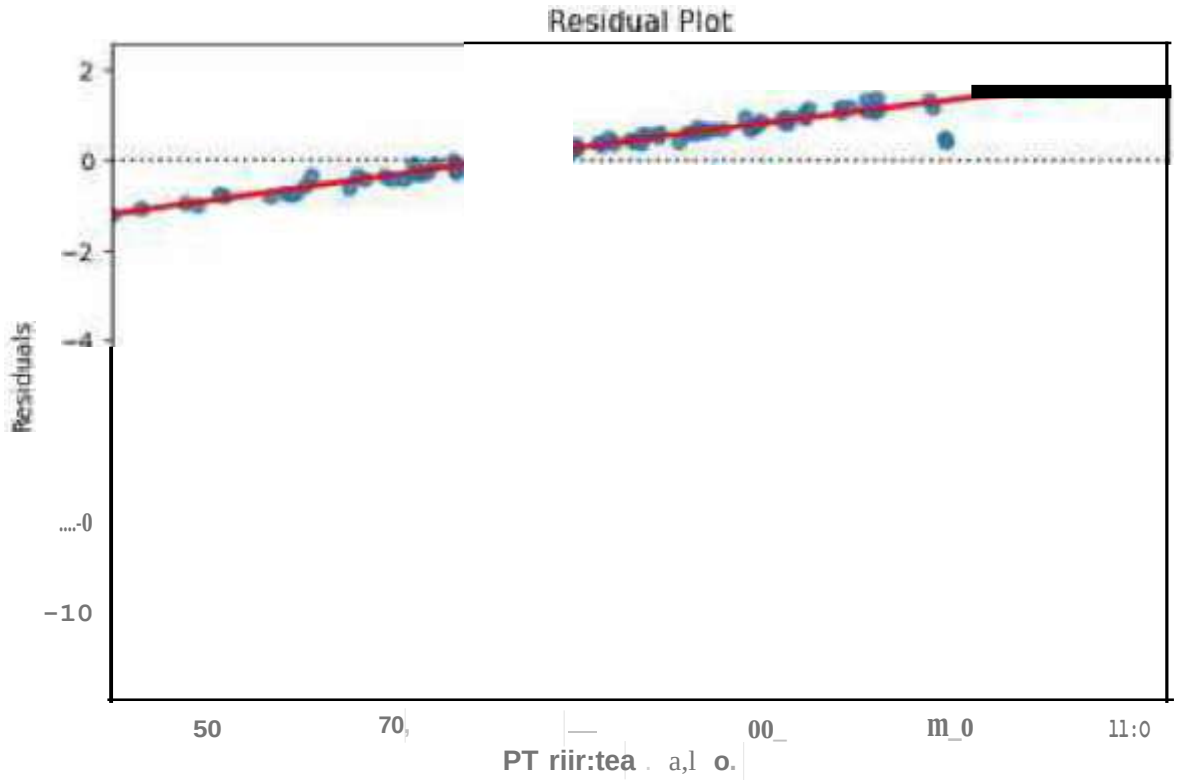


In [36]:
```
#61
# Consider a dataset HealthData.csv with features such as age  bmi  blood_pres
# Write a Python code to fit a Lasso regression model using all available feat
# Use the complete dataset to train the model. Plot the coefficients of the fe
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2 score

# Load dataset
file_path = "HealthData.csv"
df = pd.read_csv(file_path)

# Handle missing values
df = df.dropna()

# Define independent variables (features) and dependent variable (target)
X = df.drop(columns=['health_score'])  # Assuming 'health score' is the target
y = df['health_score']

# Split dataset into training and testing sets (80% train  20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Train Lasso Regression model
lasso= Lasso(alpha=0.1)  # Adjust alpha for regularization strength
lasso.fit(X_train,  y_train)

# Predictions
y_pred = lasso.predict(X_test)

# Evaluate model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R² Score: {r2:.2f}")

# Plot feature coefficients
plt.figure(figsize=(10, 5))
plt.bar(X.columns, lasso.coef_)
plt.xlabel("Features")
plt.ylabel("Coefficient Values")
plt.title("Lasso Regression Coefficients")
plt.xticks(rotation=45)
plt.show()

# Residual plot
residuals= y_test - y_pred
plt.figure(figsize=(8,  5))
sns.residplot(x=y_pred, y=residuals, lowess=True, line_kws={"color": "red"})
plt.xlabel("Predicted Health Score")
```

```
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

```
Mean Squared Error (MSE): 2.02
Mean Absolute Error (MAE): 0.63
R² Score: 0.99
```



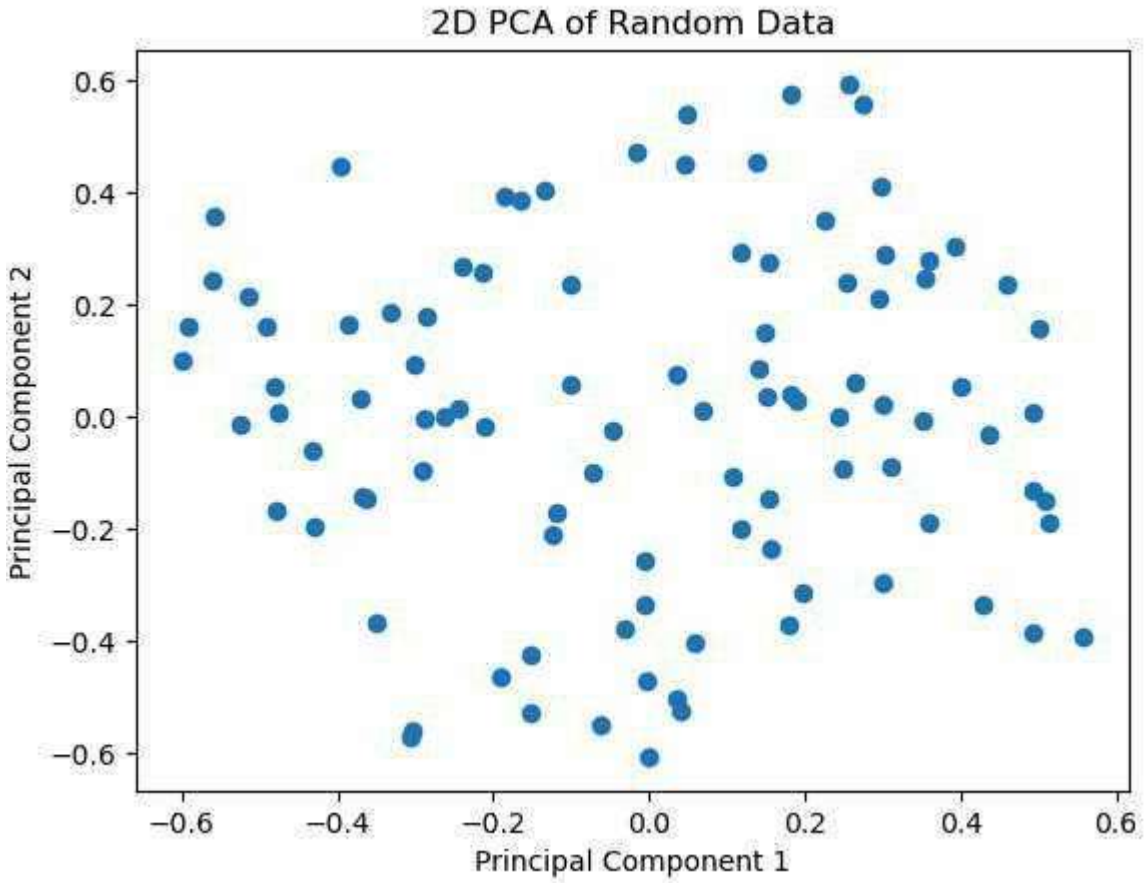


In [37]:
```
# 62
# Write a python snippet to demonstrate use of PCA on 100 samples with 3 featu
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Generate random data with 100 samples and 3 features
np.random.seed(42)
X = np.random.rand(100, 3)

# Standardize the data
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pea= PCA(n_components=2)  # Reduce to 2 principal components
X_pca = pca.fit_transform(X_scaled)

# Print explained variance ratio
print("Explained variance  ratio:", pca.explained_variance_ratio_)

# Scatter plot of PCA components
plt.scatter(X_pca[:, 0], X_pca[:, 1], color='blue', alpha=0.7)
plt.xlabel("Principal Component  1")
plt.ylabel("Principal Component 2")
plt.title("PCA Transformation (2D Projection)")
pH.grid()
plt.show()
```

```
Explained  variance  ratio: [0.37032399  0.32900197]
```

In [38]:
```
# 63
# Write a python snippet to demonstrate use of PCA on dataset of your choice.
# Also print explained variance of all principle components.
#Write a python snippet to demonstrate use of PCA on 100 samples with 3 featur
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Generate random data: 100 samples 3 features
np.random.seed(42)
X = np.random.rand(100, 3)

# Perform PCA
pea= PCA(n_components=2)  # Reduce to 2 components for visualization
X_pca = pca.fit_transform(X)

# Print explained variance ratio to see how much variance is captured by each
print("Explained Variance Ratio:", pca.explained_variance_ratio_)

# Visualize the 20 data after PCA
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.title("2D PCA of Random Data")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

```
Explained Variance Ratio: [0.37743432 0.33646087]
```



In [39]:
```
# 64
# Create synthetic Dataset using make_blobs with 3 features and 3 centers init
# object with n_components=2 to reduce the dataset to 2 dimensions.
# use matplotlib to plot the original 30 dataset and the PCA-reduced 20 datase
#Create synthetic Dataset using make_blobs with 3 features and 3 centers. init
#use matplotlib to plot the original 30 dataset and the PCA-reduced 20 dataset
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA

# Create a synthetic dataset with 3 features and 3 centers
X, y = make_blobs(n_samples=300, centers=3, n_features=3, random_state=42)

# Initialize PCA and reduce to 2 components
pea= PCA(n_components=2)
X_pca = pca.fit_transform(X)

#Setup the plot with two subplots (side by side)
fig=  plt.figure(figsize=(14, 6))

# 30 plot of the original dataset
axl = fig.add_subplot(1, 2, 1, projection='3d')
axl.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap='viridis', marker='o')
axl.set_title("Original 3D Dataset")
axl.set_xlabel("Feature  1")
axl.set_ylabel("Feature  2")
axl.set_zlabel("Feature  3")

# 20 plot of the PCA-reduced dataset
ax2 = fig.add_subplot(1, 2, 2)
scatter= ax2.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', marker='c
ax2.set_title("PCA Reduced 2D Dataset")
ax2.set_xlabel("Principal Component  1")
ax2.set_ylabel("Principal Component  2")

# Add colorbar to the 20 plot
fig.colorbar(scatter, ax=ax2, label='Class')

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

In [40]:
```
#65
# Use Load_iris from sklearn.datasets to Load the Iris dataset. initialize the
# with n_components=2 to reduce the dataset to 2. use matplotlib to plot the P
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris= load_iris()
X = iris.data
y = iris.target
feature  names= iris.feature  names

# Standardize the data (important for PCA)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensions to 2
pea= PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
pca_df = pd.DataFrame(data=X_pca, columns=['PCl', 'PC2'])
pca_df['species'] = y
print(pca_df)
# Create a DataFrame for the reduced data
pca_df = pd.DataFrame(data=X_pca, columns=['PCl', 'PC2'])
pca_df['species'] = y

# Plot the results
plt.figure(figsize=(8, 6))
for species in np.unique(y):
    plt.scatter(pca_df[pca_df['species']  ==  species]['PCl'],
                pca_df[pca_df['species']  ==  species]['PC2'],
                label=iris.target_names[species])

plt.title('PCA of Iris Dataset (2D)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```

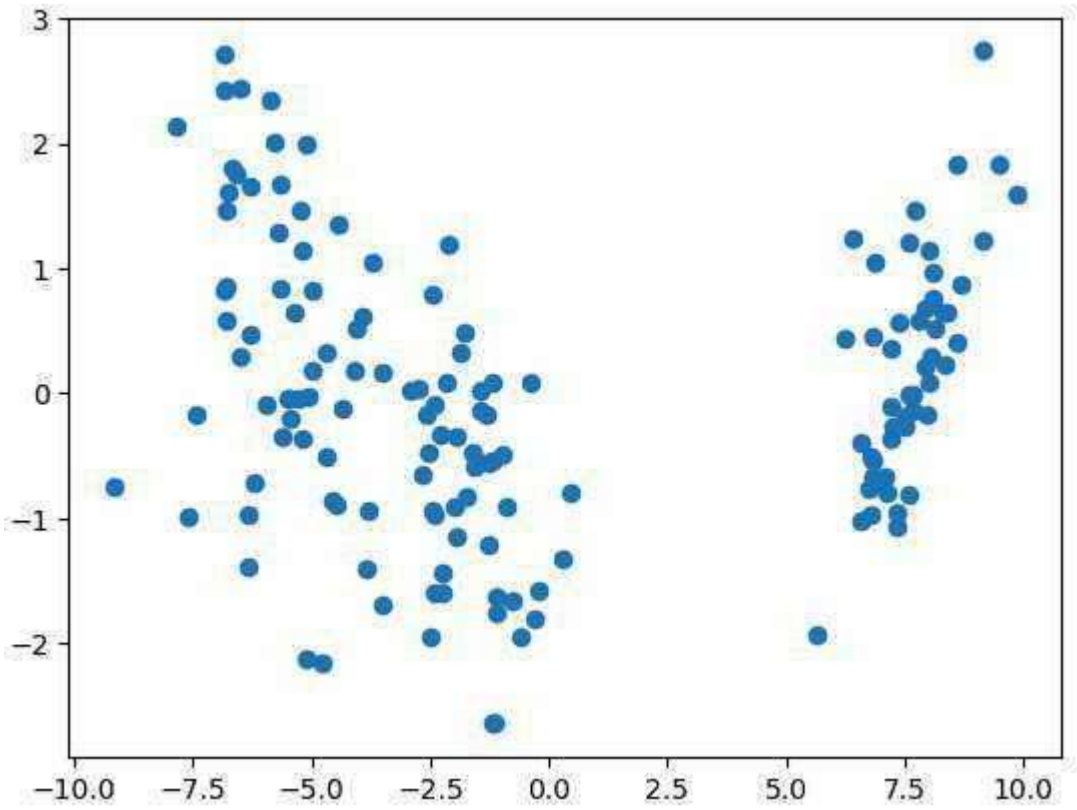
In [41]:
```
#66
# Use Load_wine from sklearn.datasets to Load the wine dataset.
# initialize the LOA object with n_components=2 to reduce the dataset to 2.
# use matplotlib to plot the LOA-reduced dataset in 20 represented by a differ
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler

# Load wine dataset
wine= load_wine()
X = wine.data  # Features
y = wine.target  # Target Labels
target_names = wine.target_names

# Standardize the data
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LOA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda  = lda.fit_transform(X_scaled, y)

# Plot the LOA-reduced dataset
plt.figure(figsize=(8,  6))
for i, label in enumerate(np.unique(y)):
    plt.scatter(X_lda[y == label, 0], X_lda[y -- label, 1], label=target_names

plt.xlabel("LDA Component  1")
plt.ylabel("LDA Component 2")
plt.title("LDA Projection of Wine Dataset")
plt.legend()
plt.grid()
plt.show()
```

In [42]:
```
#67
#Use breast cancer from sklearn.datasets to Load the cancer dataset.
# initialize the LOA object with one component to reduce the dataset to 10. us
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the breast cancer dataset
data= load_breast_cancer()
X, y = data.data, data.target

# Standardize the data
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LOA with 1 component
lda = LinearDiscriminantAnalysis(n_components=1)
X_lda = lda.fit_transform(X_scaled,y)

# Plot the LOA-reduced dataset
plt.figure(figsize=(8,  5))
plt.scatter(X_lda,  np.zeros_like(X_lda),  c=y, cmap='coolwarm', alpha=0.7, edgE

#The x-coordinates  come from X_Lda (Likely  a 10 projection of your data  such
#They-coordinates are all  e  (effectively  placing the points along a horizonta
#The color of each point corresponds to its class Label (y)  with the colors c
#The transparency  of the points is set to 70% (alpha=@.7)  and the points have

plt.xlabel('LDA Component  1')
plt.title('LDA Reduction of Breast Cancer Dataset')
plt.yticks([])
plt.colorbar(label='Class Label')
plt.show()
```

In [43]:
```
# 68
# Use Load_digits() from sklearn.datasets to Load the hand written numbers dat
# initialize the LOA object with two component to reduce the dataset. Use matp
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler

# Load the digits dataset
digits= load_digits()
X = digits.data  # Features
y = digits.target  # Target Labels

# Standardize the data
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LOA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)

# Plot the LOA-reduced dataset
plt.figure(figsize=(10, 6))
scatter= plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='jet', alpha=0.7)
plt.colorbar(scatter, label="Digit Class")
plt.xlabel("LDA Component 1")
plt.ylabel("LDA Component 2")
plt.title("LDA Projection of Handwritten Digits Dataset")
plt.grid()
plt.show()
```

In [45]:
```
# 69
# Use Load_iris from sklearn.datasets to Load the Iris dataset.
# initialize the LOA object with n_components=2 to reduce the dataset to 2.
# use matplotlib to plot the LOA-reduced dataset in 20.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris= load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names

# Standardize the data (important for PCA)
scaler= StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce dimensions to 2
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled,y)
# print(X_Lda)
plt.scatter(X_lda[:,0],X_lda[:,1])
```

Out[45]: <matplotlib.collections.PathCollection at 0x1443ad15be0>



In [47]:
```
# 70
# Implement an SVC model to classify iris flowers into three species (Setosa,
# based on their sepal and petal dimensions. The dataset contains 150 samples
# four features: sepal Length, sepal width, petal Length, and petal width.
# Load the Dataset from sklearn.Split the dataset into training and testing se
# Train the SVC Model using the training data. Visualize the Results.
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion rr

#  Load the Iris dataset
iris= datasets.load_iris()
X = iris.data  # Features: sepal Length, sepal width, petal Length, petal widt
y = iris.target  # Target: species (Setosa, Versicolor, Virginica)

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Standardize the features
scaler= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVC model
svc = SVC(kernel='rbf', C=1.0, gamma='scale')
svc.fit(X_train, y_train)

# Make predictions
y_pred = svc.predict(X_test)

# Evaluate model
accuracy= accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
print("\nClassification  Report:\n", classification_report(y_test,  y_pred))

# Compute confusion matrix
cm= confusion_matrix(y_test,  y_pred)

# Plot confusion matrix using seaborn heatmap
plt.figure(figsize=(6,  5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_narr
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix Heatmap")
plt.show()
```