In [1]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix

data = pd.read_csv("iris.csv")
data.head()
```

Out[1]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [2]:
```python
X = data.iloc[:, :-1]  # Features
y = data['Species']    # Target variable
```

In [3]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [4]:
```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [5]:
```python
lda = LinearDiscriminantAnalysis()
X_train_lda = lda.fit_transform(X_train_scaled, y_train)
X_test_lda = lda.transform(X_test_scaled)
```

In [6]:
```python
classifier = LogisticRegression()
classifier.fit(X_train_lda, y_train)
y_pred = classifier.predict(X_test_lda)
```

In [7]:
```python
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [1]:
```python
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import r2_score, accuracy_score
import warnings
warnings.filterwarnings("ignore")


# Load the diabetes dataset
data = pd.read_csv("diabetes.csv")
data.describe()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diat
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [2]:
```python
data.skew()
```

Out[2]:
```
Pregnancies                 0.901674
Glucose                     0.173754
BloodPressure              -1.843608
SkinThickness               0.109372
Insulin                     2.272251
BMI                        -0.428982
DiabetesPedigreeFunction    1.919911
Age                         1.129597
Outcome                     0.635017
dtype: float64
```

```
In [3]: data.kurt()
```

```
Out[3]: Pregnancies                 0.159220
        Glucose                     0.640780
        BloodPressure               5.180157
        SkinThickness              -0.520072
        Insulin                     7.214260
        BMI                         3.290443
        DiabetesPedigreeFunction    5.594954
        Age                         0.643159
        Outcome                    -1.600930
        dtype: float64
```

```
In [4]: data.mode().iloc[0]
```

```
Out[4]: Pregnancies                  1.000
        Glucose                     99.000
        BloodPressure               70.000
        SkinThickness                0.000
        Insulin                      0.000
        BMI                         32.000
        DiabetesPedigreeFunction     0.254
        Age                         22.000
        Outcome                      0.000
        Name: 0, dtype: float64
```

```
In [5]: X = data.drop('Outcome', axis=1)
        y = data['Outcome']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

```
In [6]: linear_reg = LinearRegression()
        linear_reg.fit(X_train, y_train)
        y_pred_linear = linear_reg.predict(X_test)
        r2_linear = r2_score(y_test, y_pred_linear)
        print(f"Linear Regression R-squared: {r2_linear}")

        # Bivariate analysis - Logistic regression
        logistic_reg = LogisticRegression()
        logistic_reg.fit(X_train, y_train)
        y_pred_logistic = logistic_reg.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred_logistic)
        print(f"Logistic Regression Accuracy: {accuracy}")
```

```
Linear Regression R-squared: 0.25500281176741757
Logistic Regression Accuracy: 0.7467532467532467
```

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_sco

data = pd.read_csv("Social_Network_Ads.csv")
data.head()
```

Out[1]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [2]:
```
X = data.iloc[:, [2, 3]]  # Features (Age and EstimatedSalary columns)
y = data['Purchased']     # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [3]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

k = 5  # Number of neighbors
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train_scaled, y_train)

y_pred = knn_classifier.predict(X_test_scaled)
confusion_matrix(y_test, y_pred)
```

Out[3]:
```
array([[48,  4],
       [ 3, 25]], dtype=int64)
```

In [4]:
```
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print("Accuracy:", accuracy*100,"%")
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 91.25 %
Error Rate: 0.08750000000000002
Precision: 0.8620689655172413
Recall: 0.8928571428571429
```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = pd.read_csv("iris.csv")
data.head()
```

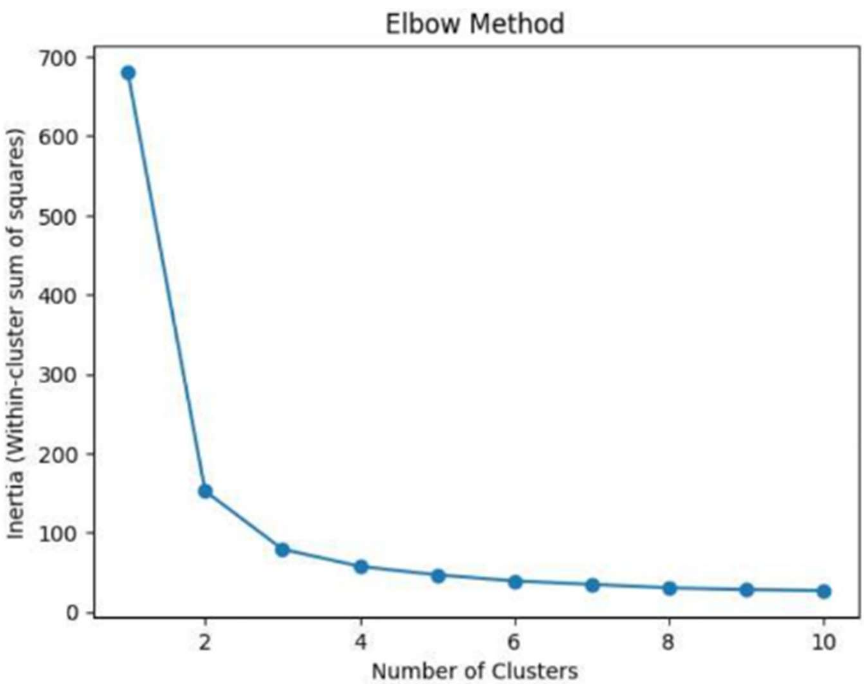
Out[1]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [2]:
```
X = data.iloc[:, [1, 2, 3, 4]]
```

In [3]:
```
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, max_iter=300, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)
```

In [4]:
```
# Plot the Elbow Method graph
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia (Within-cluster sum of squares)')
plt.title('Elbow Method')
plt.show()
```



In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import category_encoders as ce
from sklearn.metrics import accuracy_score, confusion_matrix

data = pd.read_csv("car_evaluation.csv")
data.head()
```

Out[1]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [2]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
data.columns = col_names

data.head()
```

Out[2]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [3]:
```
X =data.drop(['class'],axis=1)
y = data['class']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
X_train.shape,X_test.shape
```

Out[3]: ((1208, 6), (519, 6))

In [4]:
```
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safet
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
```

In [5]:
```
rfc=RandomForestClassifier(random_state=0)
rfc.fit(X_train,y_train)
```

Out[5]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=0)
```

In [6]:
```
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy,"\n")

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

```
Accuracy: 0.928709055876686

Confusion Matrix:
 [[107   2   8   1]
 [  8   6   2   1]
 [  7   0 354   0]
 [  7   1   0  15]]
```

In [23]:
```python
import numpy as np

maze = np.array([
    [0, 0, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 2]  # 2 is the goal
])

learning_rate = 0.1
discount_factor = 0.9
epsilon = 0.1
num_episodes = 1000

num_states, num_actions = maze.size, 4
Q = np.zeros((num_states, num_actions))

for _ in range(num_episodes):
    state = 0  # Starting position

    while True:
        action = np.random.choice(num_actions) if np.random.uniform(0, 1) < epsilon else
        new_state = state + [0,1,2,3][action]  # Up, Down, Left, Right
        reward = [-1, 1, 0][maze.flat[new_state]]
        if reward: break
        state = new_state

current_state = 0
while current_state != 16:  # Goal state
    action = np.argmax(Q[current_state, :])
    current_state = current_state + (action + 1)
    print("Agent moved to state:", current_state)
```

```
Agent moved to state: 1
Agent moved to state: 2
Agent moved to state: 3
Agent moved to state: 4
Agent moved to state: 5
Agent moved to state: 6
Agent moved to state: 7
Agent moved to state: 8
Agent moved to state: 9
Agent moved to state: 10
Agent moved to state: 11
Agent moved to state: 12
Agent moved to state: 13
Agent moved to state: 14
Agent moved to state: 15
Agent moved to state: 16
```