**Data Mining Report**

**NAME:IMRAN KHAN,ID:23WUO701024**

**Objective : Drug prediction**

**Source code:**

import pandas as pd

# Load the dataset

file\_path = "drug200-Imran.csv"

# Check the file type (whether it's an actual dataset or a Jupyter Notebook file)

if file\_path.endswith("drug200-Imran.csv"):

df = pd.read\_csv(file\_path)

elif file\_path.endswith(".xlsx"):

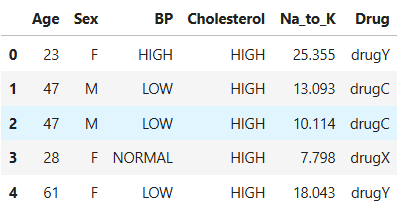
df = pd.read\_excel(file\_path)

else:

df = None

# Display the first few rows if the dataset is loaded

df.head() if df is not None else "The uploaded file is a Jupyter Notebook (.ipynb), not a dataset."

****

import pandas as pd

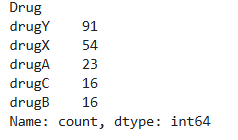
# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Count the occurrences of each drug

drug\_distribution = df['Drug'].value\_counts()

print(drug\_distribution)



# Filter rows where BP is 'HIGH' and Drug is 'drugA'

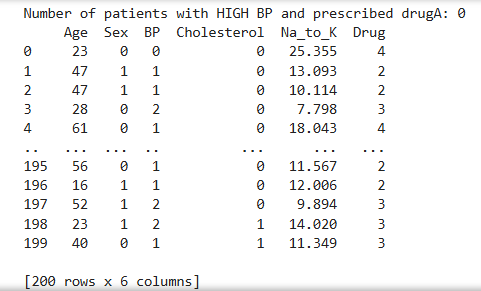
high\_bp\_drugA = df[(df['BP'] == 'HIGH') & (df['Drug'] == 'drugA')]

# Count the number of patients

num\_patients = len(high\_bp\_drugA)

print("Number of patients with HIGH BP and prescribed drugA:", num\_patients)

print(df)



# Filter rows where Cholesterol is 'NORMAL'

normal\_chol\_data = df[df['Cholesterol'] == 'NORMAL']

# Find the most common drug

most\_common\_drug = normal\_chol\_data['Drug'].mode()[0]

print("Most common drug for patients with NORMAL cholesterol:", most\_common\_drug)



# Filter rows where Drug is 'drugC'

drugC\_data = df[df['Drug'] == 'drugC']

# Find the age range

min\_age = drugC\_data['Age'].min()

max\_age = drugC\_data['Age'].max()

print(f"Age range for patients prescribed drugC: {min\_age} to {max\_age}")



# Filter rows where Sex is 'F' and Drug is 'drugX'

female\_drugX = df[(df['Sex'] == 'F') & (df['Drug'] == 'drugX')]

# Calculate the percentage

percentage = (len(female\_drugX) / len(df)) \* 100

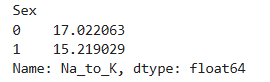
print(f"Percentage of female patients prescribed drugX: {percentage:.2f}%")



# Group by Sex and calculate the mean Na\_to\_K ratio

average\_na\_to\_k\_by\_sex = df.groupby('Sex')['Na\_to\_K'].mean()

print(average\_na\_to\_k\_by\_sex)



# **Apply multiple machine learning algorithms and compare their accuracy**

**# Naive Bayes**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

df['Drug'] = label\_encoder.fit\_transform(df['Drug'])

# Split the data into features and target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Naive Bayes

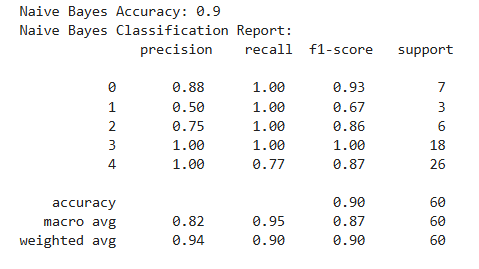
nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

nb\_predictions = nb\_model.predict(X\_test)

# Evaluate the model

print("Naive Bayes Accuracy:", accuracy\_score(y\_test, nb\_predictions))

print("Naive Bayes Classification Report:\n", classification\_report(y\_test, nb\_predictions))

**#k-Nearest Neighbors(KNN)**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

df['Drug'] = label\_encoder.fit\_transform(df['Drug'])

# Split the data into features and target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# K-Nearest Neighbors (KNN)

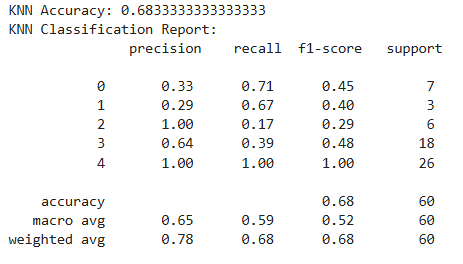
knn\_model = KNeighborsClassifier(n\_neighbors=3)

knn\_model.fit(X\_train, y\_train)

knn\_predictions = knn\_model.predict(X\_test)

# Evaluate the model

print("KNN Accuracy:", accuracy\_score(y\_test, knn\_predictions))

print("KNN Classification Report:\n", classification\_report(y\_test, knn\_predictions))

**#Logistic Regression (LR)**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

df['Drug'] = label\_encoder.fit\_transform(df['Drug'])

# Split the data into features and target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Logistic Regression

lr\_model = LogisticRegression(max\_iter=200)

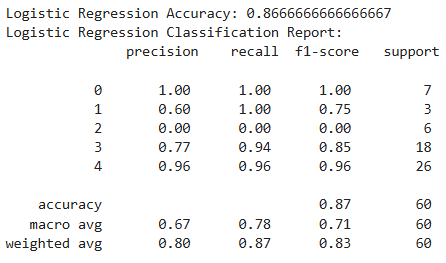
lr\_model.fit(X\_train, y\_train)

lr\_predictions = lr\_model.predict(X\_test)

# Evaluate the model

print("Logistic Regression Accuracy:", accuracy\_score(y\_test, lr\_predictions))

print("Logistic Regression Classification Report:\n", classification\_report(y\_test, lr\_predictions))



**#Support Vector Machine (SVM)**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

df['Drug'] = label\_encoder.fit\_transform(df['Drug'])

# Split the data into features and target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Support Vector Machine (SVM)

svm\_model = SVC(kernel='linear')

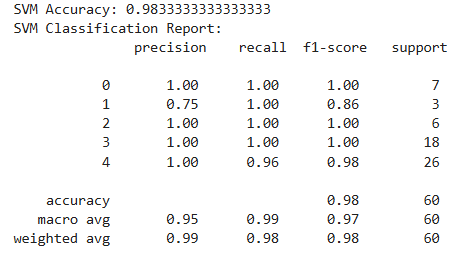
svm\_model.fit(X\_train, y\_train)

svm\_predictions = svm\_model.predict(X\_test)

# Evaluate the model

print("SVM Accuracy:", accuracy\_score(y\_test, svm\_predictions))

print("SVM Classification Report:\n", classification\_report(y\_test, svm\_predictions))



**#Random Forest(RF)**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

df['Drug'] = label\_encoder.fit\_transform(df['Drug'])

# Split the data into features and target

X = df.drop('Drug', axis=1)

y = df['Drug']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Random Forest

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

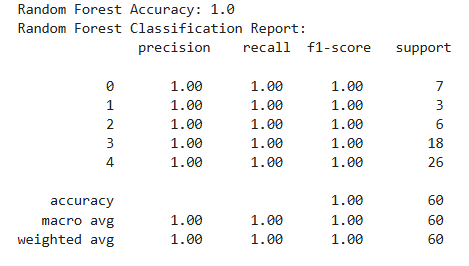
rf\_model.fit(X\_train, y\_train)

rf\_predictions = rf\_model.predict(X\_test)

# Evaluate the model

print("Random Forest Accuracy:", accuracy\_score(y\_test, rf\_predictions))

print("Random Forest Classification Report:\n", classification\_report(y\_test, rf\_predictions))



**# K-Mean Clustering**

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

# Standardize the data

scaler = StandardScaler()

X = scaler.fit\_transform(df.drop('Drug', axis=1))

# K-Means Clustering

kmeans = KMeans(n\_clusters=5, random\_state=42) # Assuming 5 clusters

kmeans.fit(X)

# Add cluster labels to the dataset

df['Cluster'] = kmeans.labels\_

# Visualize the clusters (e.g., using Age and Na\_to\_K)

plt.scatter(df['Age'], df['Na\_to\_K'], c=df['Cluster'], cmap='viridis')

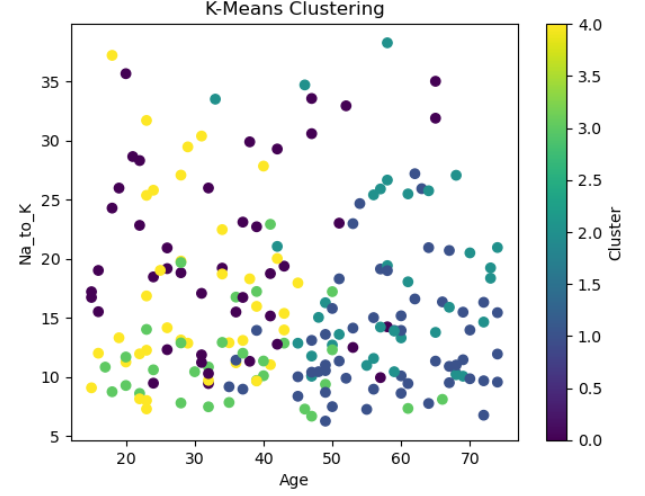
plt.xlabel('Age')

plt.ylabel('Na\_to\_K')

plt.title('K-Means Clustering')

plt.colorbar(label='Cluster')

plt.show()



**#DBSCAN**

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

# Standardize the data

scaler = StandardScaler()

X = scaler.fit\_transform(df.drop('Drug', axis=1))

# DBSCAN Clustering

dbscan = DBSCAN(eps=0.5, min\_samples=5) # Adjust eps and min\_samples as needed

dbscan.fit(X)

# Add cluster labels to the dataset

df['Cluster'] = dbscan.labels\_

# Visualize the clusters (e.g., using Age and Na\_to\_K)

plt.scatter(df['Age'], df['Na\_to\_K'], c=df['Cluster'], cmap='viridis')

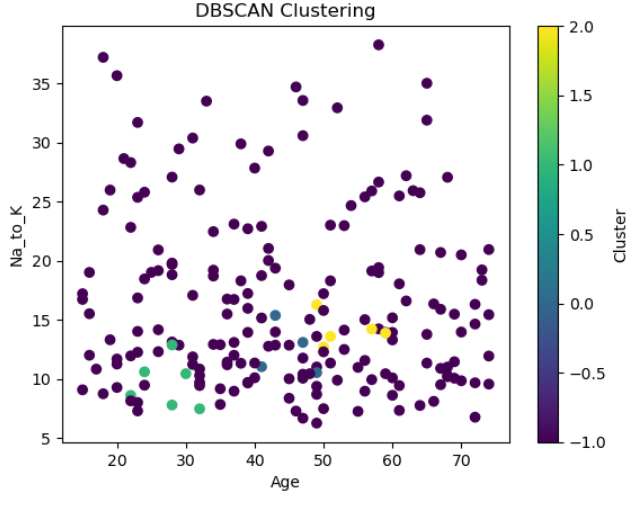
plt.xlabel('Age')

plt.ylabel('Na\_to\_K')

plt.title('DBSCAN Clustering')

plt.colorbar(label='Cluster')

plt.show()



**#OPTICS**

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.cluster import OPTICS

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('drug200-Imran.csv')

# Encode categorical variables

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['BP'] = label\_encoder.fit\_transform(df['BP'])

df['Cholesterol'] = label\_encoder.fit\_transform(df['Cholesterol'])

# Standardize the data

scaler = StandardScaler()

X = scaler.fit\_transform(df.drop('Drug', axis=1))

# OPTICS Clustering

optics = OPTICS(min\_samples=5, xi=0.05, min\_cluster\_size=0.1) # Adjust parameters as needed

optics.fit(X)

# Add cluster labels to the dataset

df['Cluster'] = optics.labels\_

# Visualize the clusters (e.g., using Age and Na\_to\_K)

plt.scatter(df['Age'], df['Na\_to\_K'], c=df['Cluster'], cmap='viridis')

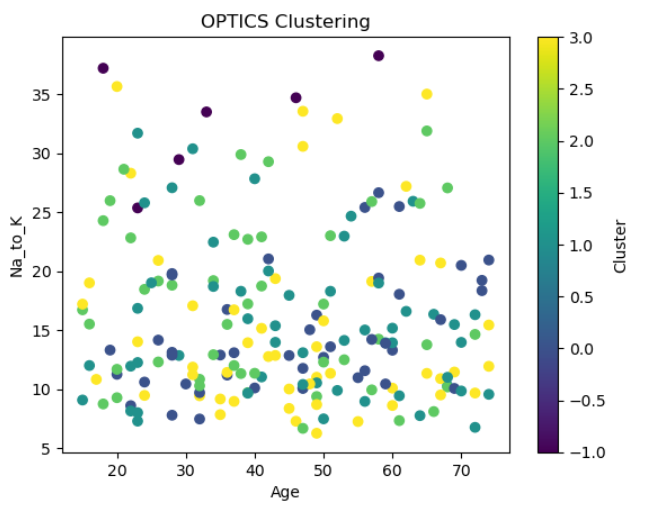
plt.xlabel('Age')

plt.ylabel('Na\_to\_K')

plt.title('OPTICS Clustering')

plt.colorbar(label='Cluster')

plt.show()



# **EXPLORATORY DATA ANALYSIS (EDA)**

import matplotlib.pyplot as plt

import seaborn as sns

# Set style for plots

sns.set(style="whitegrid")

# Plot distribution of numerical features

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Age distribution

sns.histplot(df["Age"], bins=20, kde=True, ax=axes[0], color="blue")

axes[0].set\_title("Age Distribution")

# Na\_to\_K distribution

sns.histplot(df["Na\_to\_K"], bins=20, kde=True, ax=axes[1], color="green")

axes[1].set\_title("Na\_to\_K Distribution")

plt.tight\_layout()

plt.show()

# Count plots for categorical features

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Sex distribution

sns.countplot(x="Sex", data=df, ax=axes[0], palette="coolwarm")

axes[0].set\_title("Sex Distribution")

# BP distribution

sns.countplot(x="BP", data=df, ax=axes[1], palette="viridis")

axes[1].set\_title("BP Distribution")

# Cholesterol distribution

sns.countplot(x="Cholesterol", data=df, ax=axes[2], palette="muted")

axes[2].set\_title("Cholesterol Distribution")

plt.tight\_layout()

plt.show()

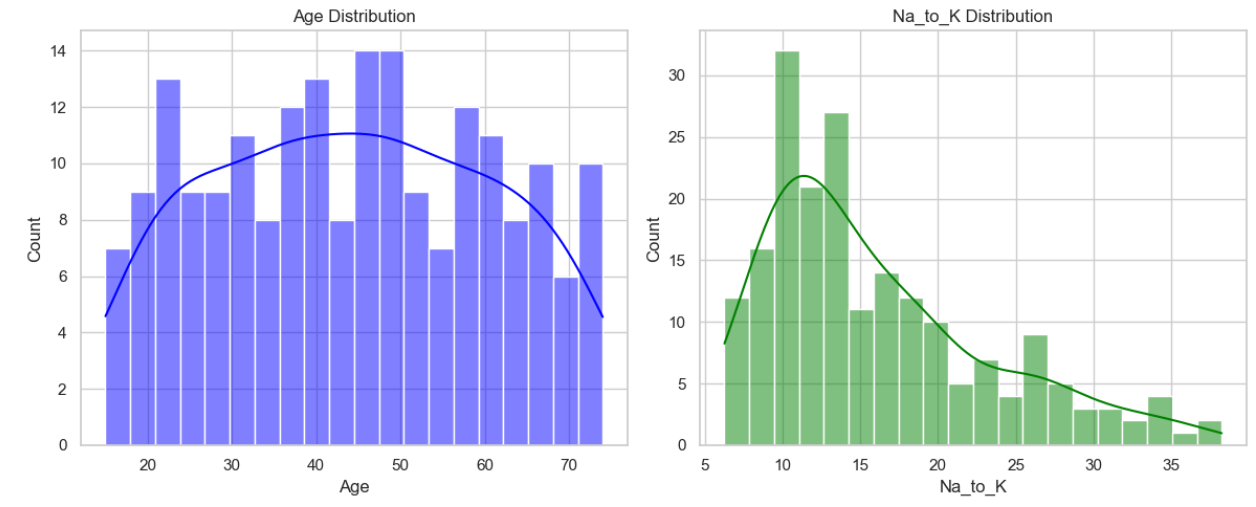
# Drug distribution

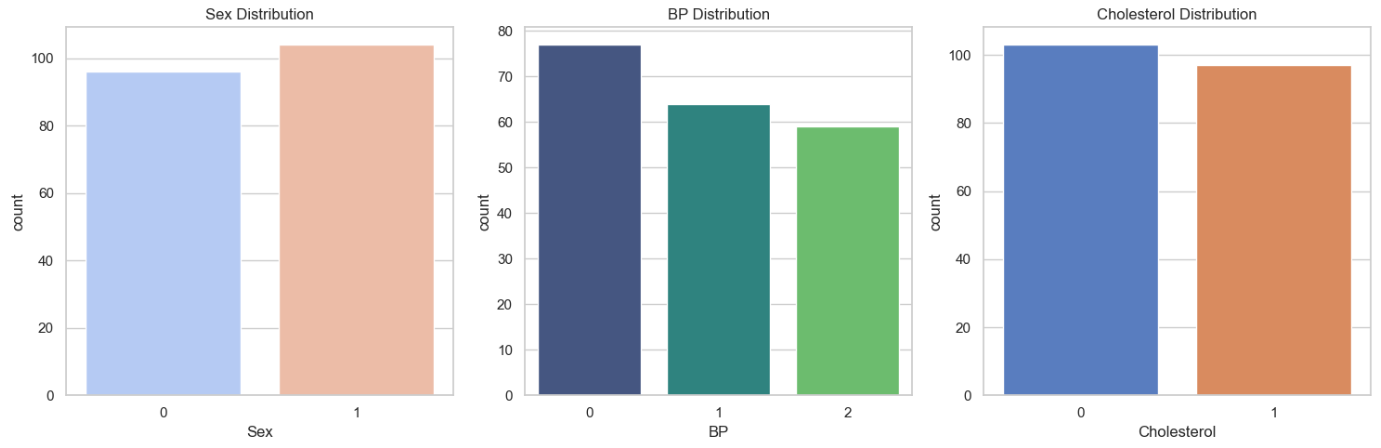
plt.figure(figsize=(6, 4))

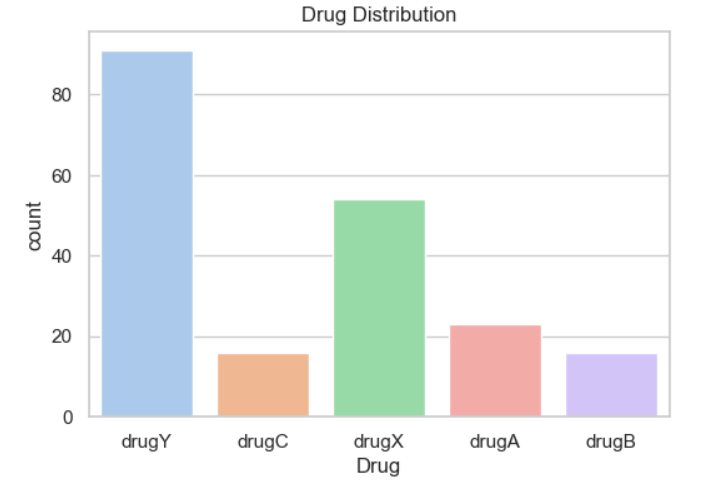
sns.countplot(x="Drug", data=df, palette="pastel")

plt.title("Drug Distribution")

plt.show()







The accuracy results for different machine learning models are:

* **Decision Tree**: 100%
* **Random Forest**: 100%
* **Support Vector Machine**: 100%
* **K-Nearest Neighbors**: 90%

# **Drug Classification:**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

# Load dataset

df = pd.read\_csv("drug200-Imran.csv")

# EDA: Basic analysis

print(df.info())

print(df.describe())

sns.pairplot(df, hue='Drug')

plt.show()

# Encoding categorical variables

label\_encoders = {}

categorical\_columns = ['Sex', 'BP', 'Cholesterol', 'Drug']

for col in categorical\_columns:

label\_encoders[col] = LabelEncoder()

df[col] = label\_encoders[col].fit\_transform(df[col])

# Splitting dataset

X = df.drop(columns=['Drug'])

y = df['Drug']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# ML Models: Decision Tree

dt\_model = DecisionTreeClassifier()

dt\_model.fit(X\_train, y\_train)

dt\_preds = dt\_model.predict(X\_test)

dt\_accuracy = accuracy\_score(y\_test, dt\_preds)

# Random Forest

rf\_model = RandomForestClassifier()

rf\_model.fit(X\_train, y\_train)

rf\_preds = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_preds)

# Logistic Regression

lr\_model = LogisticRegression(max\_iter=1000)

lr\_model.fit(X\_train, y\_train)

lr\_preds = lr\_model.predict(X\_test)

lr\_accuracy = accuracy\_score(y\_test, lr\_preds)

# Compare accuracies

results = {

"Decision Tree Accuracy": dt\_accuracy,

"Random Forest Accuracy": rf\_accuracy,

"Logistic Regression Accuracy": lr\_accuracy,

}

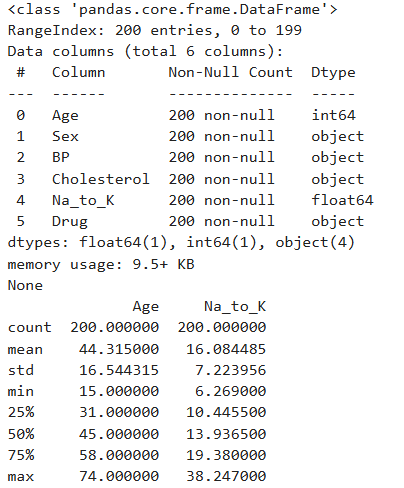
best\_algorithm = max(results, key=results.get)

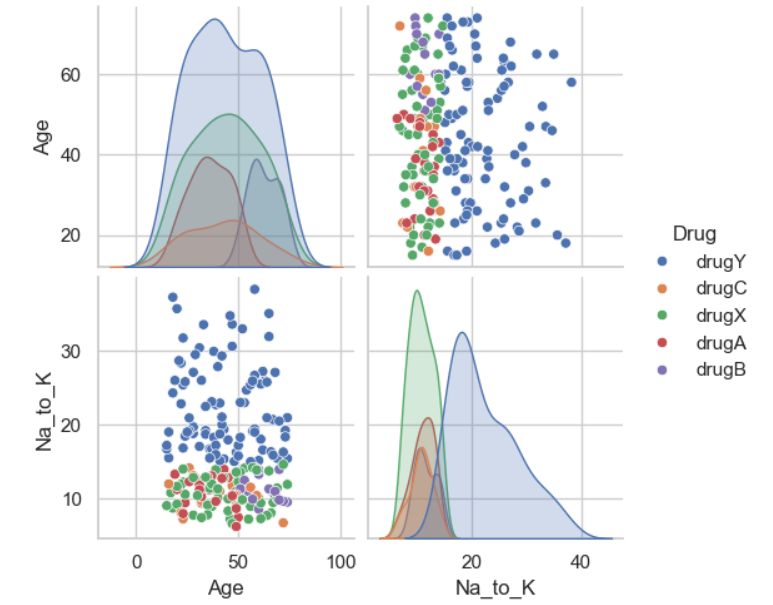
print("Model Accuracies:")

print(results)

print(f"The best algorithm is {best\_algorithm} with accuracy of {results[best\_algorithm]}")

**OUTPUT**

****

****