Drug Classification using Supervised Machine Learning

Overview

Data has been collected from a set of patients that all suffer from the same illness. Throughout their treatment, each patient responded to one of five different medications. The purpose of this exercise, is to develop a model that could help predict the appropriate drug for a future patient with the same illness based on certain characteristics about their health.

The public repository in Github can be found here: https://github.com/scollareno28/DTSA-5509-Final-Project

The model will be developed using three different supervised machine learning models: a decision tree, logistic regression, and a random forest.

This dataset came from Kaggle: https://www.kaggle.com/datasets/prathamtripathi/drug-classification See the citations section for detailed citation of this dataset.

The license of this dataset is a public domain with over 261 different code contributions and the size of the dataset is 2 kB.

The data will be imported using pandas.

Loading and Exploring Raw Dataset

In [ ]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('drugdata200.csv')
```

Now that the dataset has been imported, it's time to understand what is in this dataset.

In [ ]:
```
data.info
```

Out[ ]:
```
<bound method DataFrame.info of      Age Sex      BP Cholesterol  Na_to_K   Drug
0     23   F    HIGH        HIGH   25.355  DrugY
1     47   M     LOW        HIGH   13.093  drugC
2     47   M     LOW        HIGH   10.114  drugC
3     28   F  NORMAL        HIGH    7.798  drugX
4     61   F     LOW        HIGH   18.043  DrugY
..   ...  ..     ...         ...      ...    ...
195   56   F     LOW        HIGH   11.567  drugC
196   16   M     LOW        HIGH   12.006  drugC
197   52   M  NORMAL        HIGH    9.894  drugX
198   23   M  NORMAL      NORMAL   14.020  drugX
199   40   F     LOW      NORMAL   11.349  drugX

[200 rows x 6 columns]>
```

In [ ]:
```
data.shape
```

Out[ ]:
```
(200, 6)
```

In [ ]:
```
data.dtypes
```

Out[ ]:
```
Age              int64
Sex             object
BP              object
Cholesterol     object
Na_to_K        float64
Drug            object
dtype: object
```

Looking at the columns, this dataset uses a variety of factors including age, sex, blood pressure (BP), Cholesterol, and Na/k ratio (NA_to_K). Na is sodium and K is potassium. In recent years, doctors have been using the Na/K as a ratio to measure a person's diet quality and overall health according to the European Journal of Medical Research: https://eurjmedres.biomedcentral.com/articles/10.1186/s40001-020-00476-5#citeas.

The drug column is the label and ultimately the target.

Data cleaning

First check to see if there are any missing values:

In [ ]:
```
if data.isnull().values.any():
    print("Dataframe contains null values")
else:
    print("Dataframe has no null values")
```

```
Dataframe has no null values
```

Looking at the Na/K column, the number of decimal places will be reduce in order to make the data easier to look at.

In [ ]:
```
#Rounds the decimal places
data["Na_to_K"] = data["Na_to_K"].round(2)
data.info
```

Out[ ]:
```
<bound method DataFrame.info of      Age Sex      BP Cholesterol  Na_to_K   Drug
0     23   F    HIGH        HIGH    25.36  DrugY
1     47   M     LOW        HIGH    13.09  drugC
2     47   M     LOW        HIGH    10.11  drugC
3     28   F  NORMAL        HIGH     7.80  drugX
4     61   F     LOW        HIGH    18.04  DrugY
..   ...  ..     ...         ...      ...    ...
195   56   F     LOW        HIGH    11.57  drugC
196   16   M     LOW        HIGH    12.01  drugC
197   52   M  NORMAL        HIGH     9.89  drugX
198   23   M  NORMAL      NORMAL    14.02  drugX
199   40   F     LOW      NORMAL    11.35  drugX

[200 rows x 6 columns]>
```

Look at how each drug is labeled:

In [ ]:
```
unique_drugs = data["Drug"].unique()
for drug in unique_drugs:
    print(drug)
```

```
DrugY
drugC
drugX
drugA
drugB
```

To reduce redundancy, each drug variable will be renamed.

In [ ]:
```
#Create a mapping dictionary
mapping = {
    'DrugY': 'Y',
    'drugC': 'C',
    'drugX': 'X',
    'drugA': 'A',
    'drugB': 'B'
}
#Now use the dictionary to replace
data["Drug"] = data["Drug"].replace(mapping)
print(data)
```

```
     Age Sex      BP Cholesterol  Na_to_K Drug
0     23   F    HIGH        HIGH    25.36    Y
1     47   M     LOW        HIGH    13.09    C
2     47   M     LOW        HIGH    10.11    C
3     28   F  NORMAL        HIGH     7.80    X
4     61   F     LOW        HIGH    18.04    Y
..   ...  ..     ...         ...      ...  ...
195   56   F     LOW        HIGH    11.57    C
196   16   M     LOW        HIGH    12.01    C
197   52   M  NORMAL        HIGH     9.89    X
198   23   M  NORMAL      NORMAL    14.02    X
199   40   F     LOW      NORMAL    11.35    X

[200 rows x 6 columns]
```

Let's also take a look at the statistics of the dataset

In [ ]:
```
#Statistical Analysis
statistics = data.describe()
print(statistics)
```

```
              Age     Na_to_K
count  200.000000  200.000000
mean    44.315000   16.084900
std     16.544315    7.224417
min     15.000000    6.270000
25%     31.000000   10.447500
50%     45.000000   13.940000
75%     58.000000   19.382500
max     74.000000   38.250000
```

The numeric values all seem to be reasonable and no outliers are detected. It's important to check if there are any unique values for the other remaining categorical values to ensure a clean dataset.

In [ ]:
```
# Unique Blood Pressure
unique_BP = data["BP"].unique()
for i in unique_BP:
    print(i)

# Unique Cholesterol
unique_cholesterol = data["Cholesterol"].unique()
for i in unique_cholesterol :
    print(i)

# Unique Sex
unique_sex = data["Sex"].unique()
for i in unique_sex :
    print(i)
```

```
HIGH
LOW
NORMAL
HIGH
NORMAL
F
M
```

All of the categorical values have the expected values. There are not any typos or misplaced data labels.

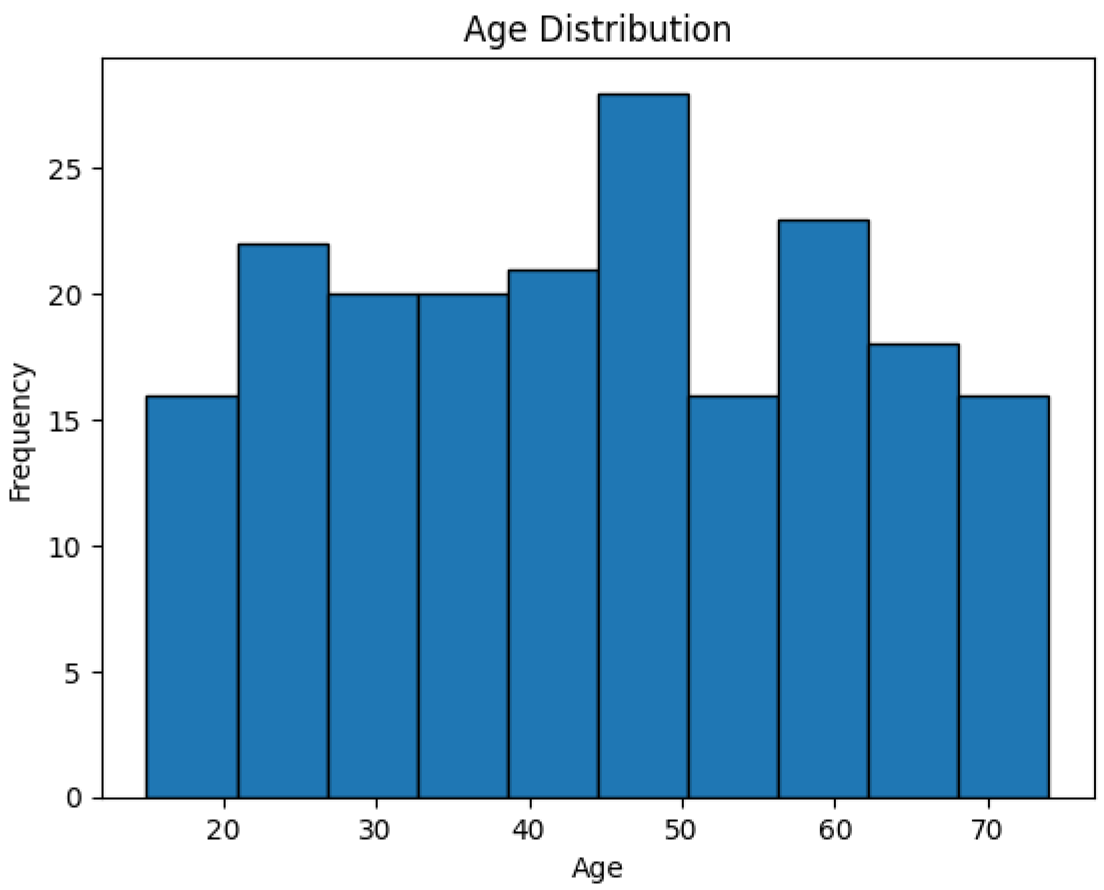
The cleaning that was done for this dataset was done in order to make working with decimals in the "Na_to_K" easier and reduce the redundancy of the "Drug" column.

Exploratory Data Analysis (EDA)

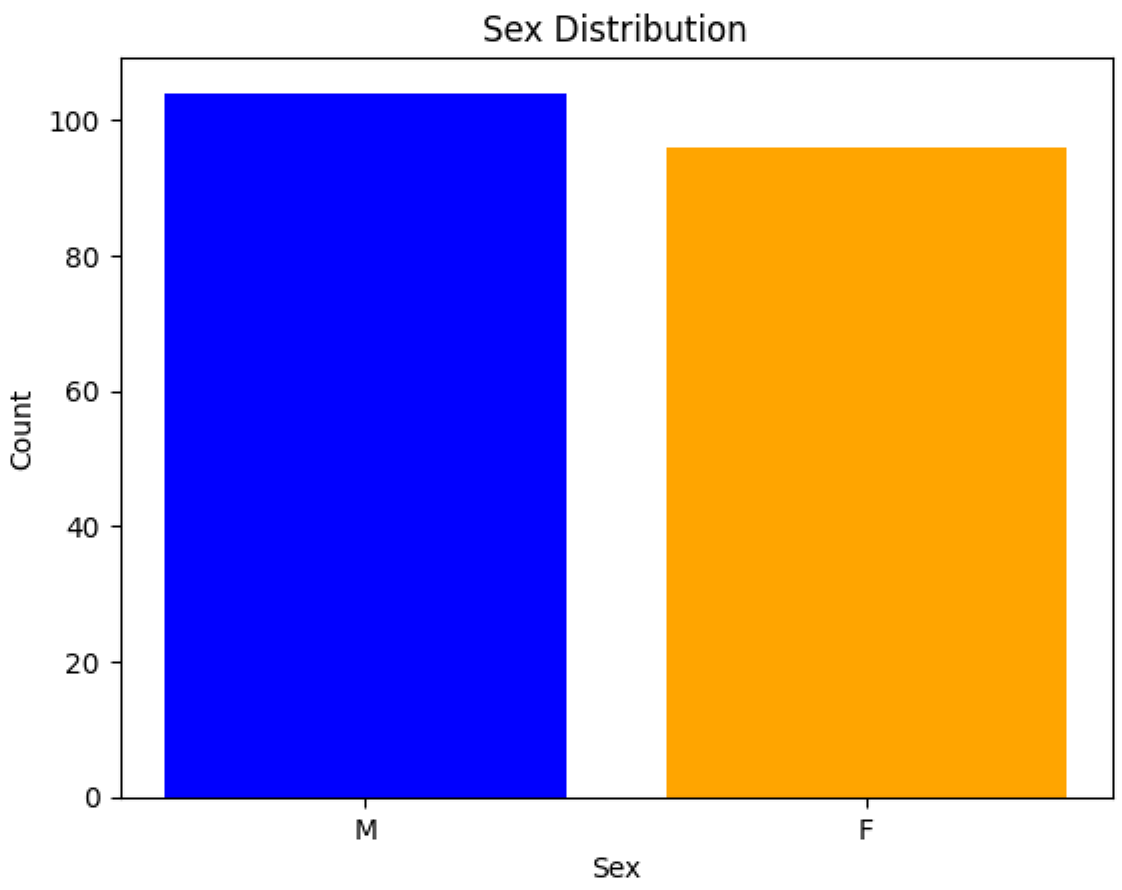
The data looks good enought work with and explore at a higher lever. The purpose of doing this is to detect any outliers that might mess with the data or be a mistake.

Additionally, deeper analysis could result in uncovering relationships between variables or variables and the label.

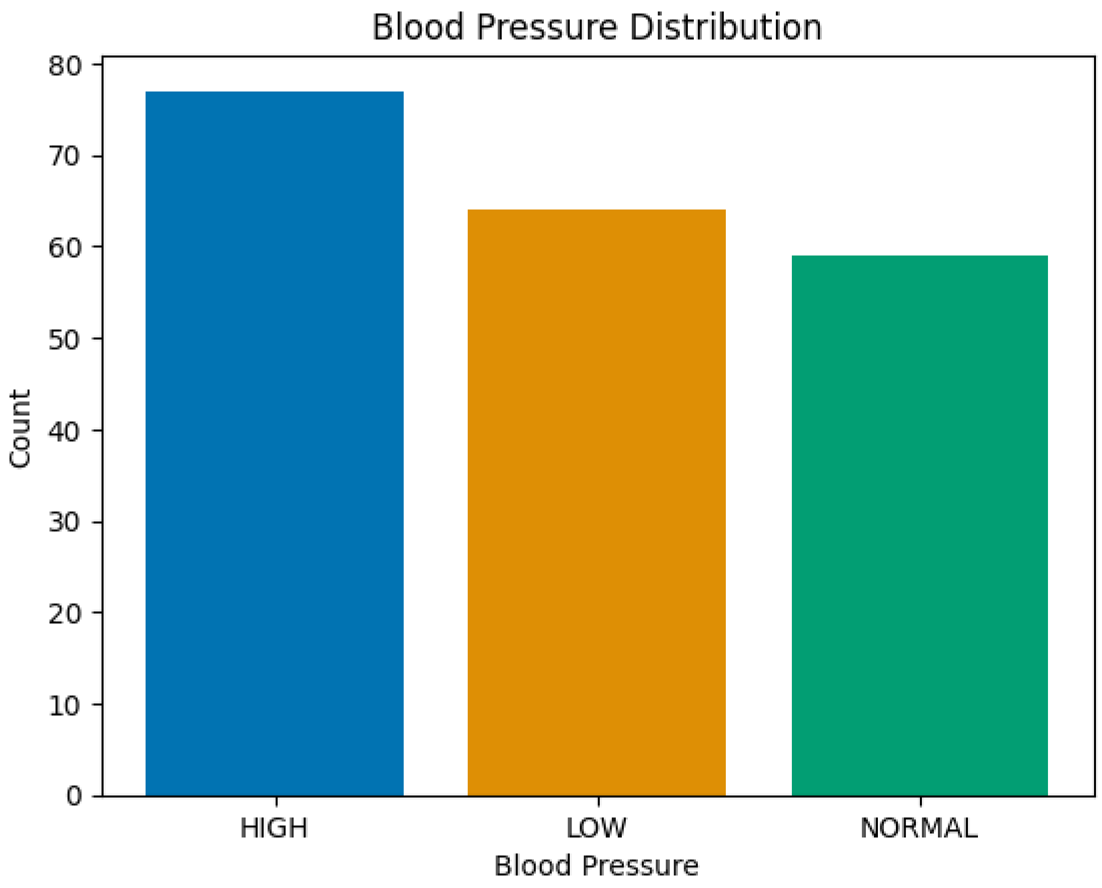
In [ ]:
```
import matplotlib.pyplot as plt
# Age distribution
plt.hist(data["Age"], bins = 10, edgecolor = 'black')
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



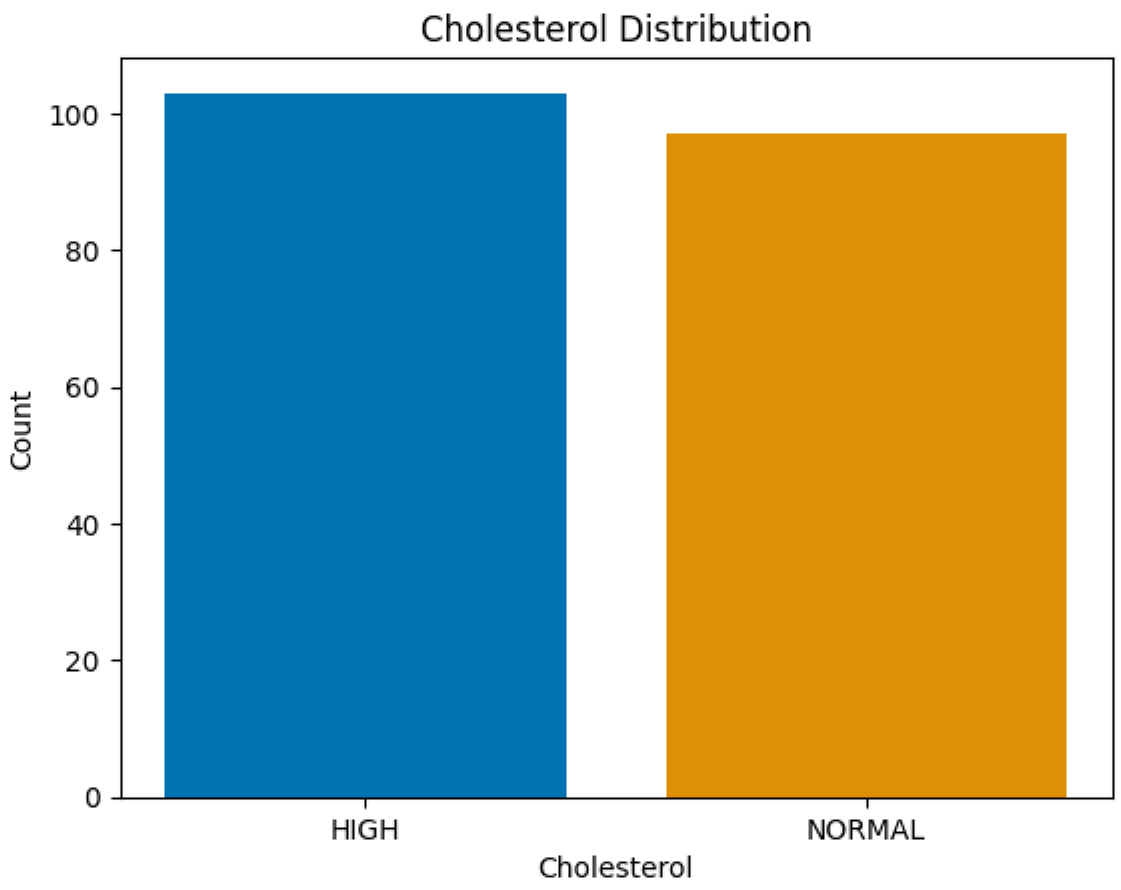
```
In [ ]: # Sex distribution
        sex_counts = data["Sex"].value_counts()
        colors = ['blue', 'orange']
        plt.bar(sex_counts.index, sex_counts.values, color = colors)
        plt.title("Sex Distribution")
        plt.xlabel("Sex")
        plt.ylabel("Count")
        plt.show()
```



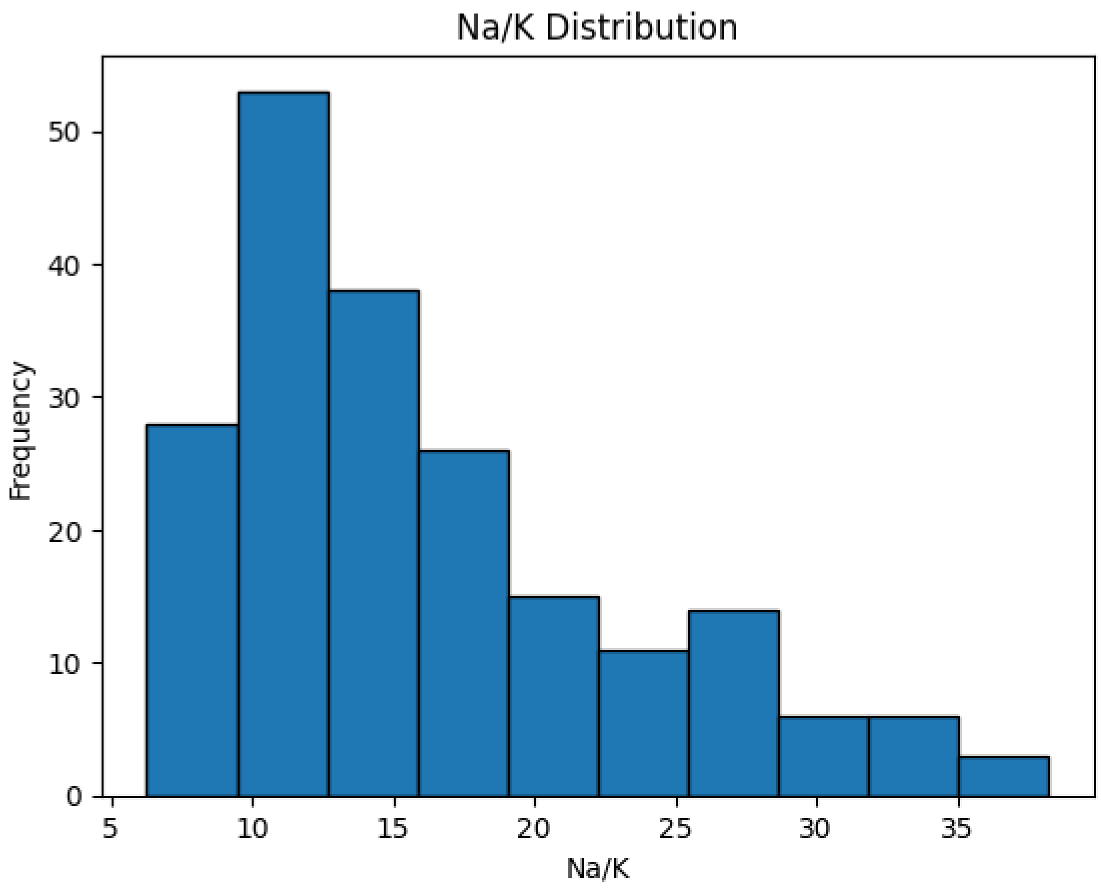
In [ ]:
```
import seaborn as sns
# Blood pressure distribution
bp_counts = data["BP"].value_counts()
colors = sns.color_palette("colorblind")
plt.bar(bp_counts.index, bp_counts.values, color = colors)
plt.title("Blood Pressure Distribution")
plt.xlabel("Blood Pressure")
plt.ylabel("Count")
plt.show()
```



In [ ]:
```
# Cholesterol distribution
Cholesterol_counts = data["Cholesterol"].value_counts()
colors = sns.color_palette("colorblind")
plt.bar(Cholesterol_counts.index, Cholesterol_counts.values, color = colors)
plt.title("Cholesterol Distribution")
plt.xlabel("Cholesterol")
plt.ylabel("Count")
plt.show()
```

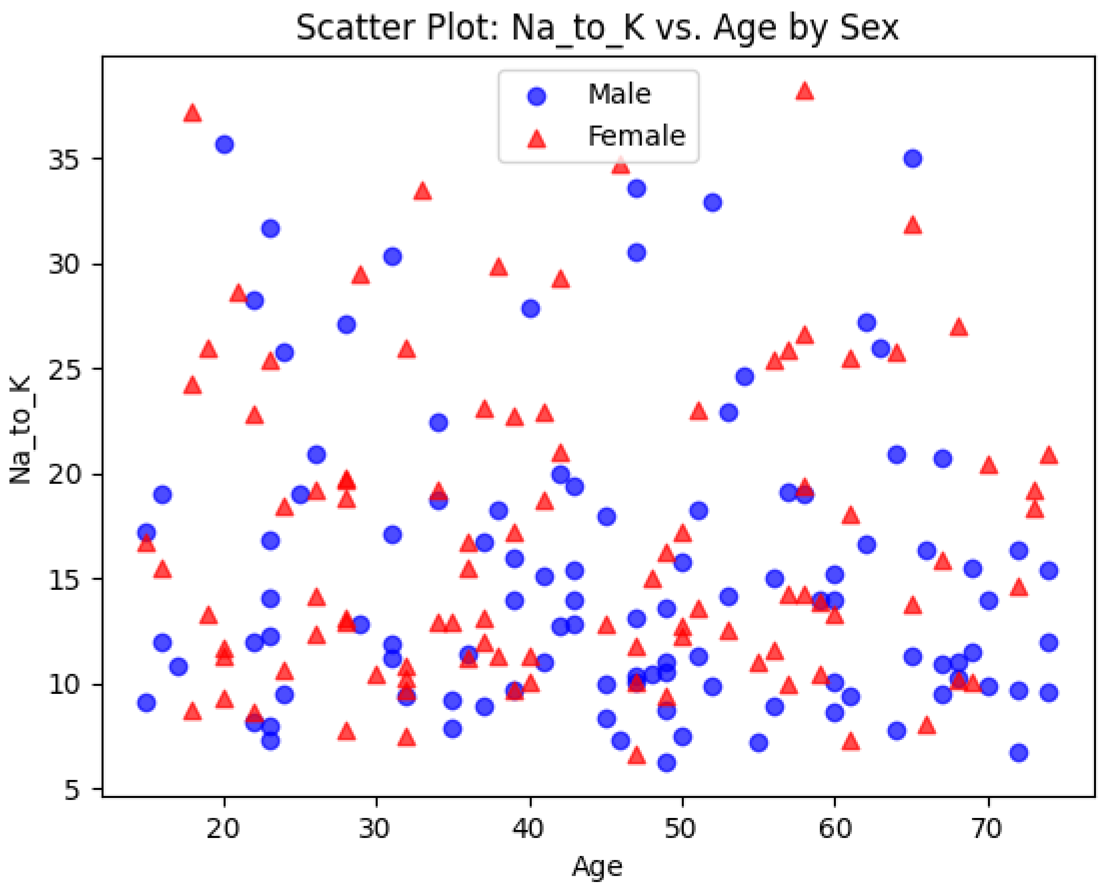


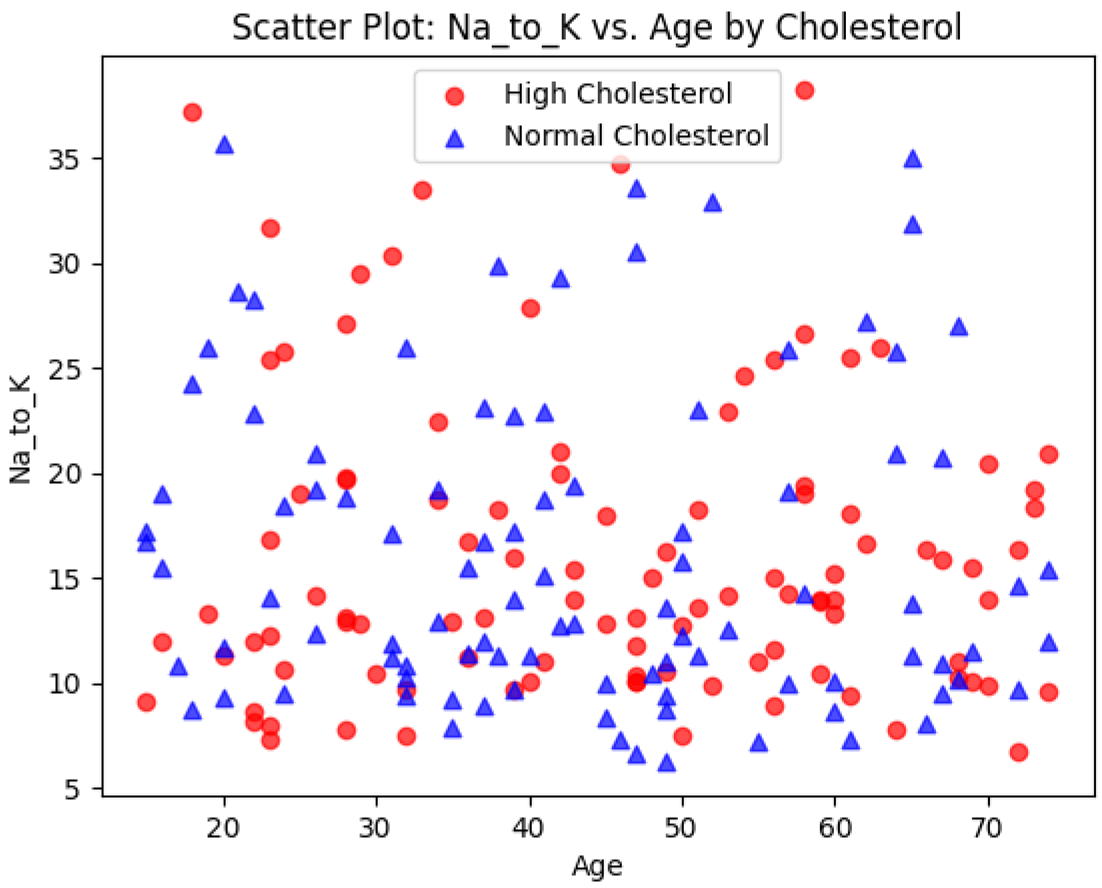
In [ ]:
```
# Na/K distribution
plt.hist(data["Na_to_K"], bins = 10, edgecolor = 'black')
plt.title("Na/K Distribution")
plt.xlabel("Na/K")
plt.ylabel("Frequency")
plt.show()
```

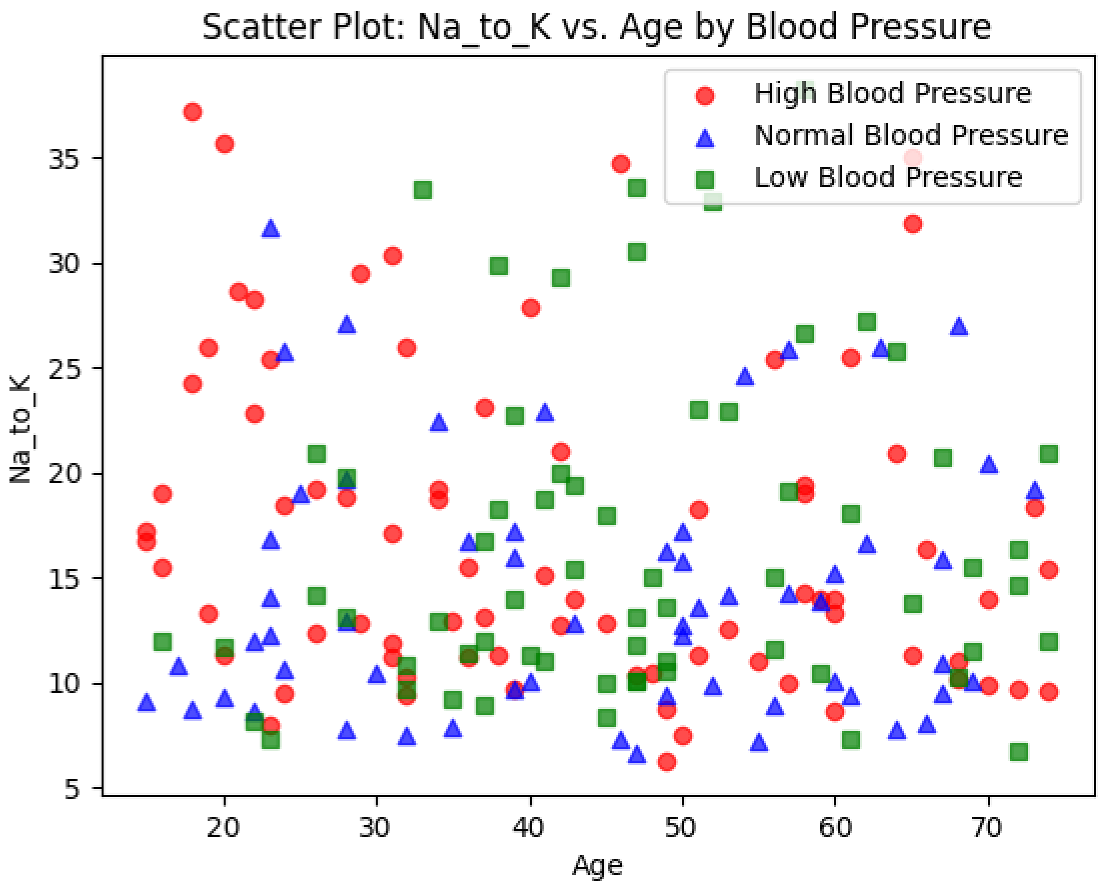


Age, cholesterol, blood pressure, and sex all seem to have a good distribution but Na/K appears a little lopsided. Let's look at how this may affect things by comparing the N/K ratio across age, cholesterol, blood pressure, and sex.

In [ ]:
```
#Create plot comparing Na/K vs age by sex
# Create the scatter plot
plt.scatter(data[data['Sex'] == 'M']['Age'], data[data['Sex'] == 'M']['Na_to_K'], c
plt.scatter(data[data['Sex'] == 'F']['Age'], data[data['Sex'] == 'F']['Na_to_K'], c


plt.title("Scatter Plot: Na_to_K vs. Age by Sex")
plt.xlabel("Age")
plt.ylabel("Na_to_K")

plt.legend()

plt.show()


#Create plot comparing Na/K vs age by Cholesterol
# Create the scatter plot
plt.scatter(data[data['Cholesterol'] == 'HIGH']['Age'], data[data['Cholesterol'] ==
plt.scatter(data[data['Cholesterol'] == 'NORMAL']['Age'], data[data['Cholesterol']


plt.title("Scatter Plot: Na_to_K vs. Age by Cholesterol")
plt.xlabel("Age")
plt.ylabel("Na_to_K")

plt.legend()

plt.show()


#Create plot comparing Na/K vs age by Blood Pressure
# Create the scatter plot
plt.scatter(data[data['BP'] == 'HIGH']['Age'], data[data['BP'] == 'HIGH']['Na_to_K'
plt.scatter(data[data['BP'] == 'NORMAL']['Age'], data[data['BP'] == 'NORMAL']['Na_t
plt.scatter(data[data['BP'] == 'LOW']['Age'], data[data['BP'] == 'LOW']['Na_to_K'],

plt.title("Scatter Plot: Na_to_K vs. Age by Blood Pressure")
plt.xlabel("Age")
plt.ylabel("Na_to_K")

plt.legend()

plt.show()
```



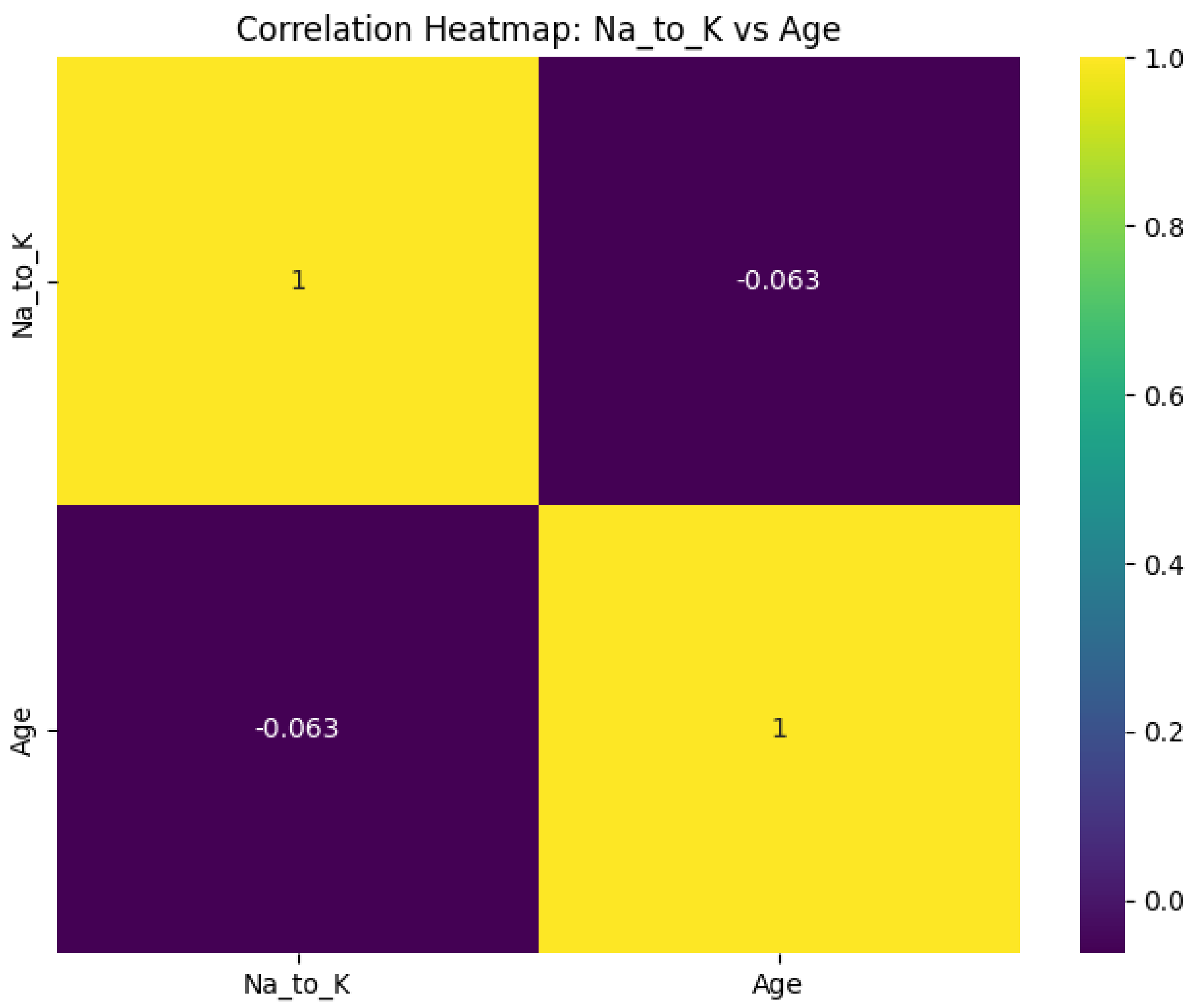

Scatter Plot: Na_to_K vs. Age by Cholesterol
High Cholesterol
Normal Cholesterol
Na_to_K
Age
5
10
15
20
25
30
35
20
30
40
50
60
70


Scatter Plot: Na_to_K vs. Age by Blood Pressure
High Blood Pressure
Normal Blood Pressure
Low Blood Pressure
Na_to_K
Age
5
10
15
20
25
30
35
20
30
40
50
60
70

Based on the scatter plots above, the biggest takeaway is that there doesn't seem to be any relation among the variables. Further analysis will be needed to confirm if this is the case.

Since sex, cholesterol, and blood pressure are all categorical data some scatter plots were made since these values aren't useful for making a correlation heatmap or corrleation matrix. This will create further confirmation about the relationship between variables.

In [ ]:
```python
# Select the "Na_to_K" and "Age" columns
columns = ["Na_to_K", "Age"]
subset_data = data[columns]
# Compute the correlation matrix
correlation_matrix = subset_data.corr()

# Create the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="viridis")

plt.title("Correlation Heatmap: Na_to_K vs Age")
plt.show()
```



As expected based on the scatterplots, there is a very weak negative correlation between the two variables.

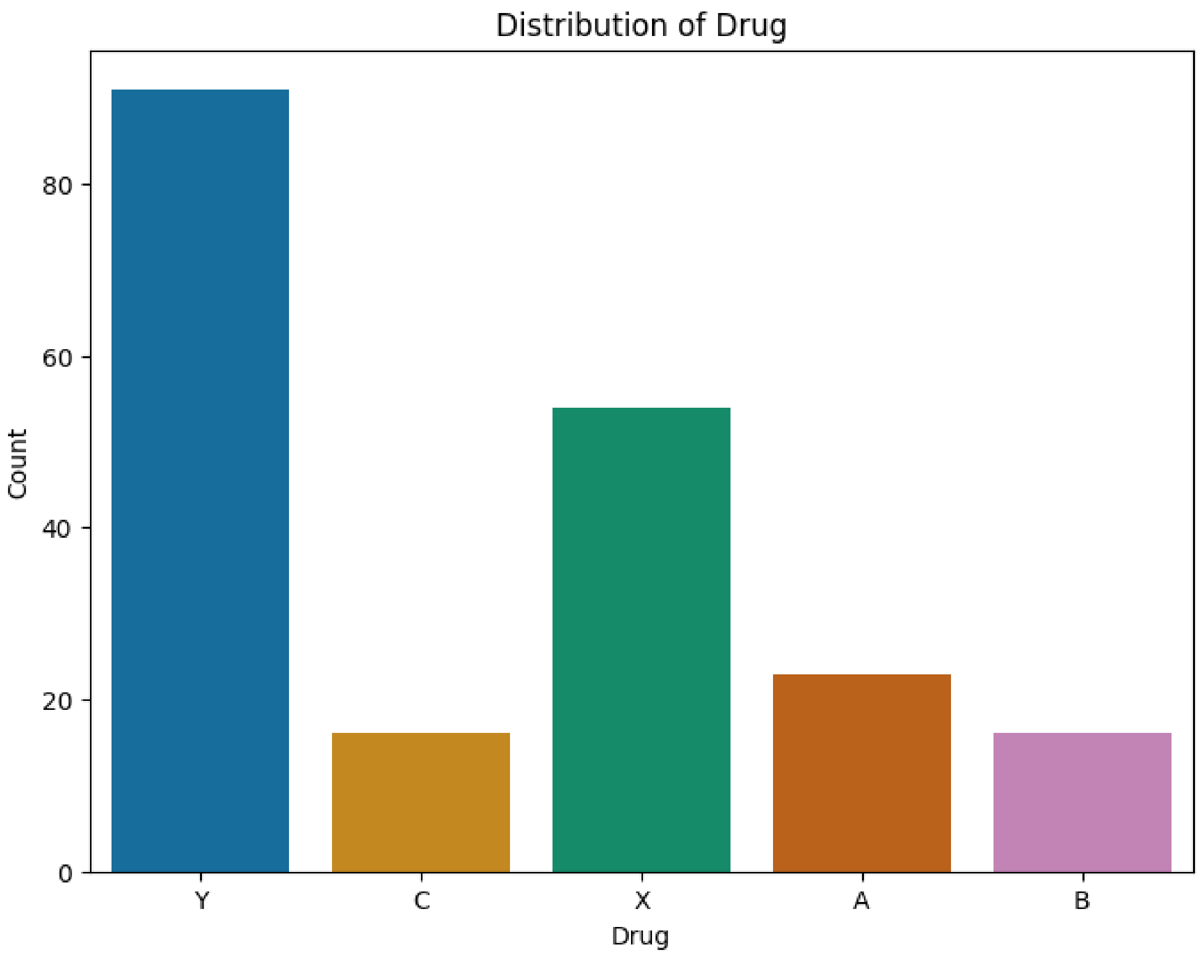
Overall it does not look like the variables will influence one another very much.

Exploring the variables and their relation to the drug type

Continuing the exploration, it is important to take a look at how each drug type may relate to each variable.

First, let's take a look at the distribution of the drug type.

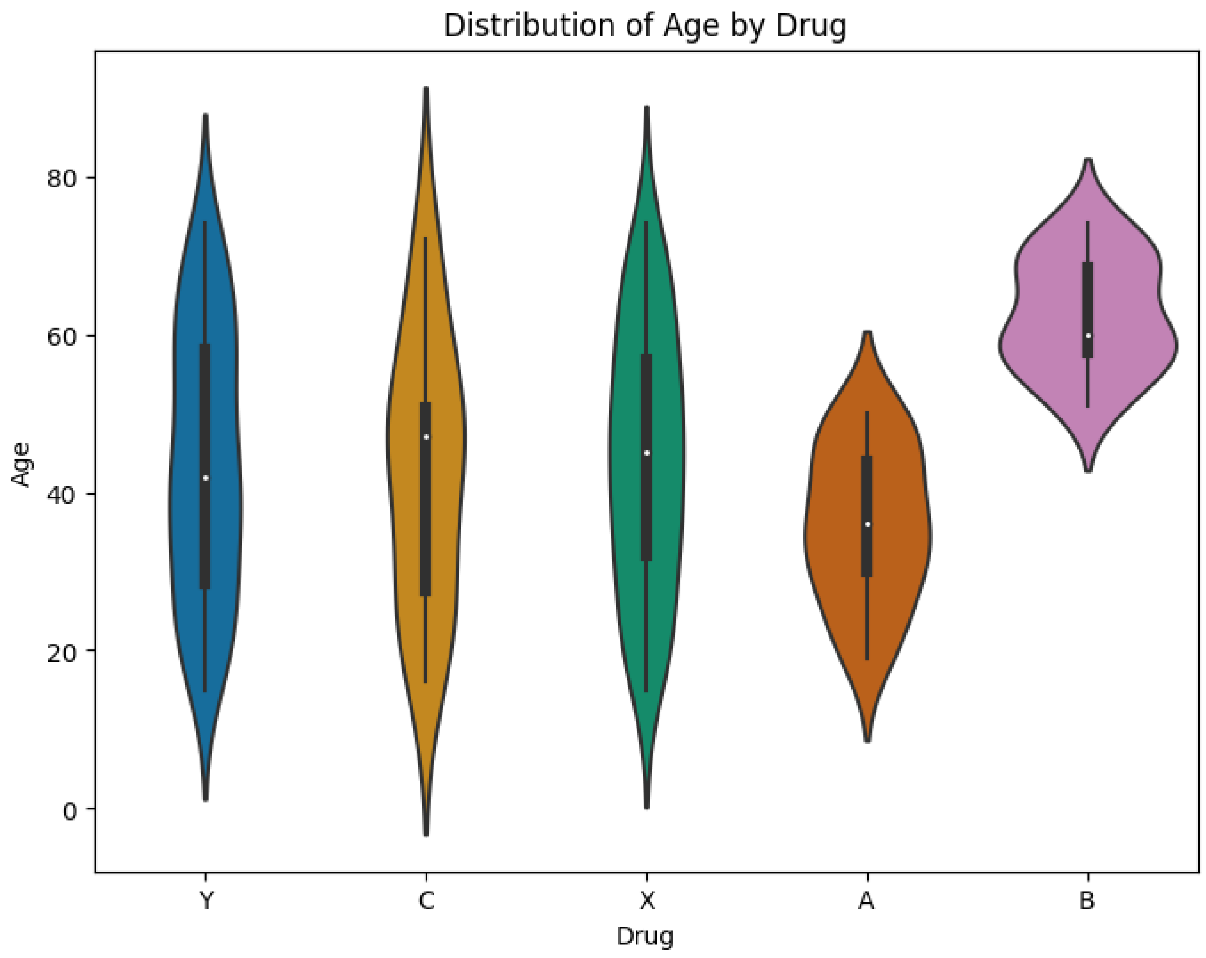
In [ ]:
```
# Count the occurrences of each unique value in the "Drug" column
drug_counts = data["Drug"].value_counts()

# Create the bar plot
plt.figure(figsize=(8, 6))
sns.countplot(x="Drug", data=data, palette="colorblind")


plt.title("Distribution of Drug")
plt.xlabel("Drug")
plt.ylabel("Count")


plt.show()
```

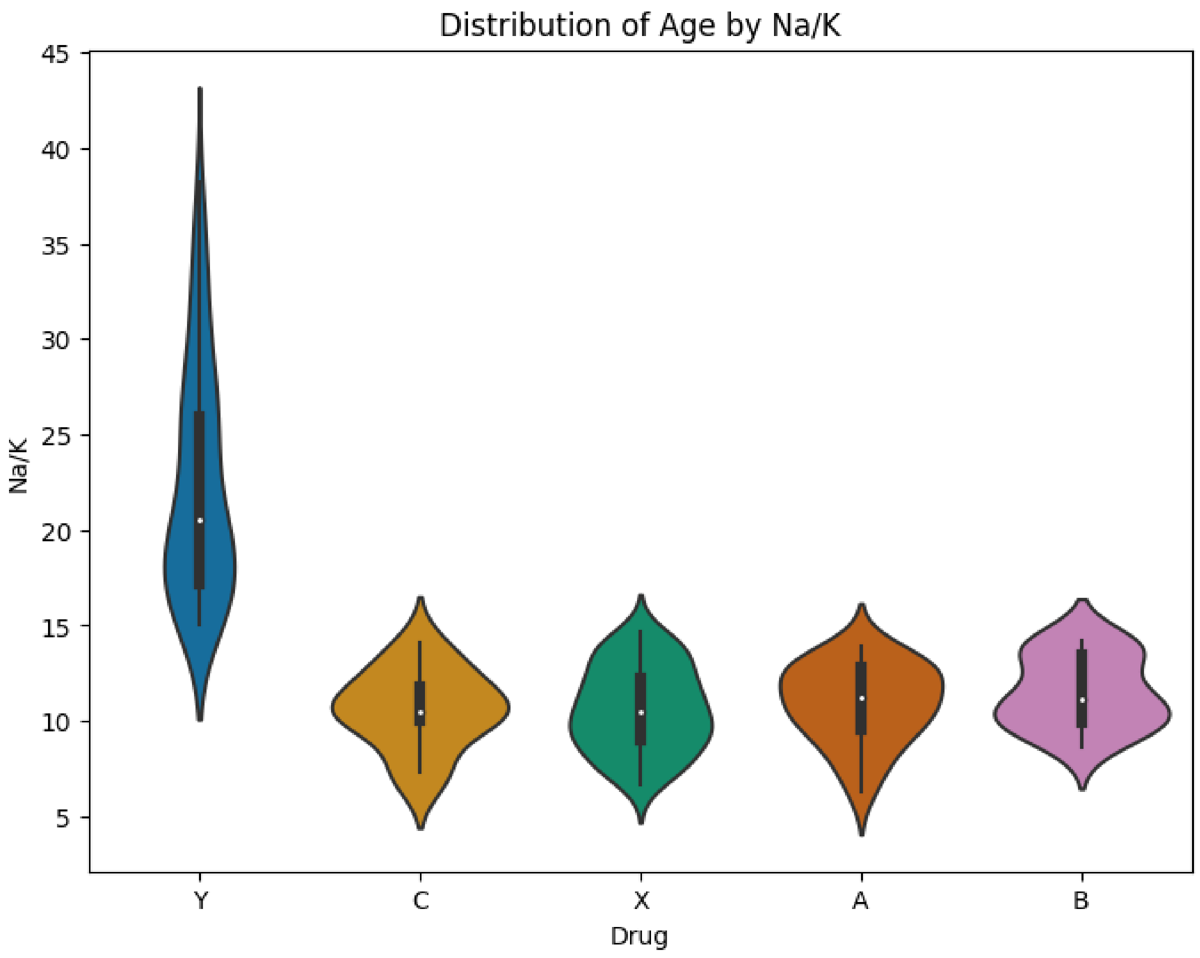


There is a noticeable amount of Drug Y compared to the other drugs in this dataset. This will become important before any classification begins, but for now more analysis is required.

Violin plots will be used to examine the distribution of drugs across age and Na/K ratio becuase they are numeric.

In [ ]:
```
# Violin Plots
# Drug and age
plt.figure(figsize=(8, 6))
sns.violinplot(x="Drug", y="Age", data=data, palette="colorblind")


plt.title("Distribution of Age by Drug")
plt.xlabel("Drug")
plt.ylabel("Age")


plt.show()

# Drug and Na/K
plt.figure(figsize=(8, 6))
sns.violinplot(x="Drug", y="Na_to_K", data=data, palette="colorblind")


plt.title("Distribution of Age by Na/K")
plt.xlabel("Drug")
plt.ylabel("Na/K")


plt.show()
```

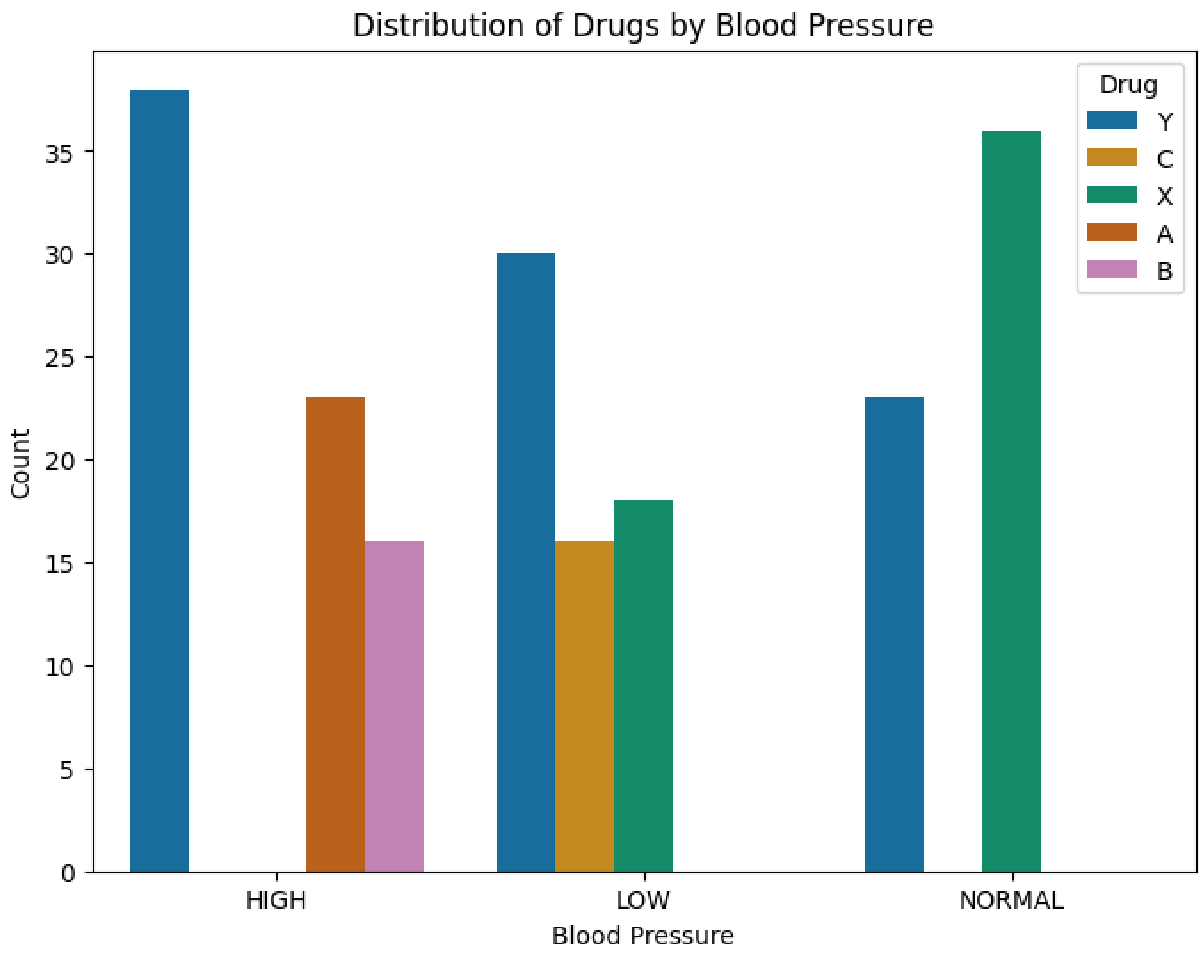

Distribution of Age by Drug
Age
80
60
40
20
0
Y
C
X
A
B
Drug

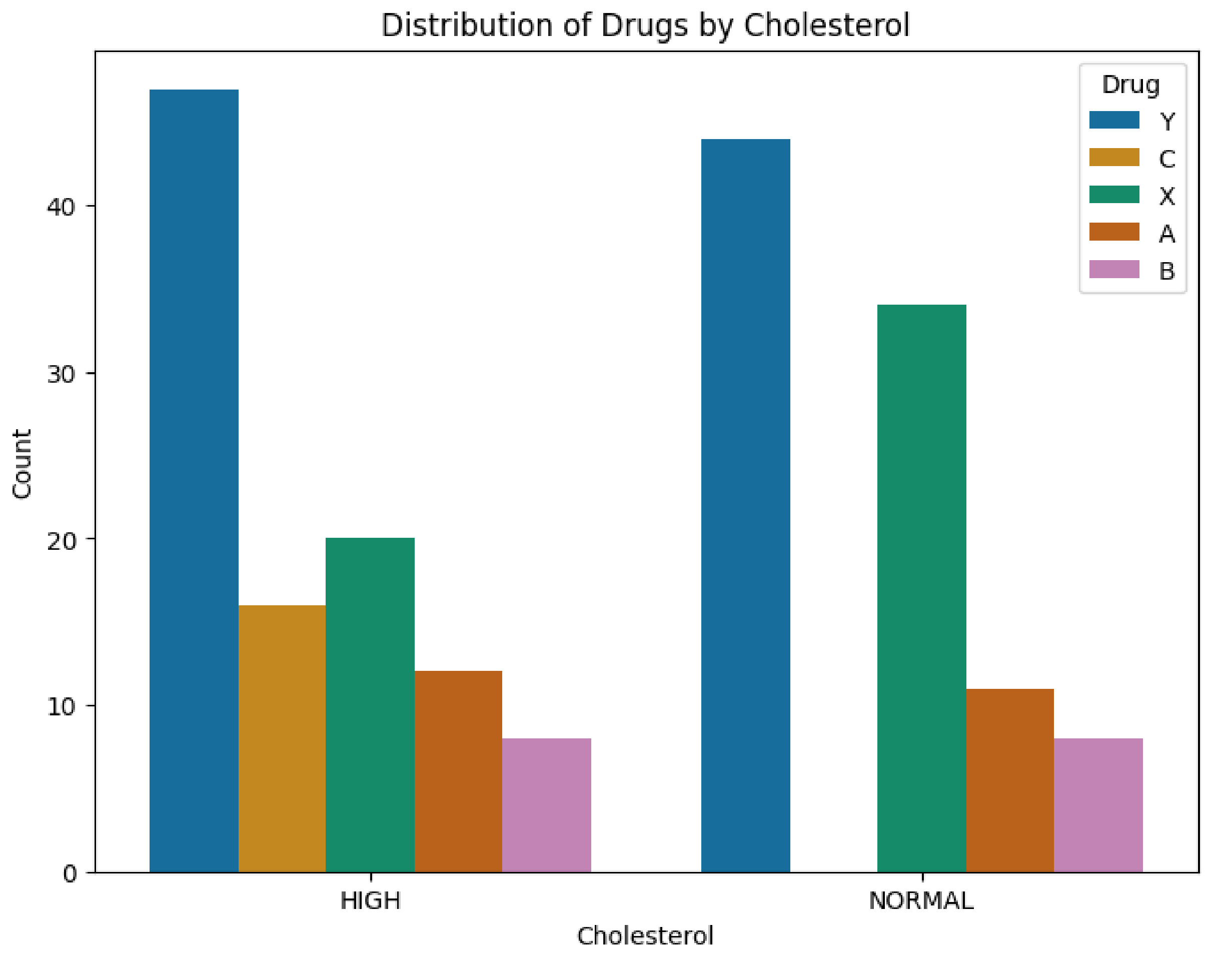


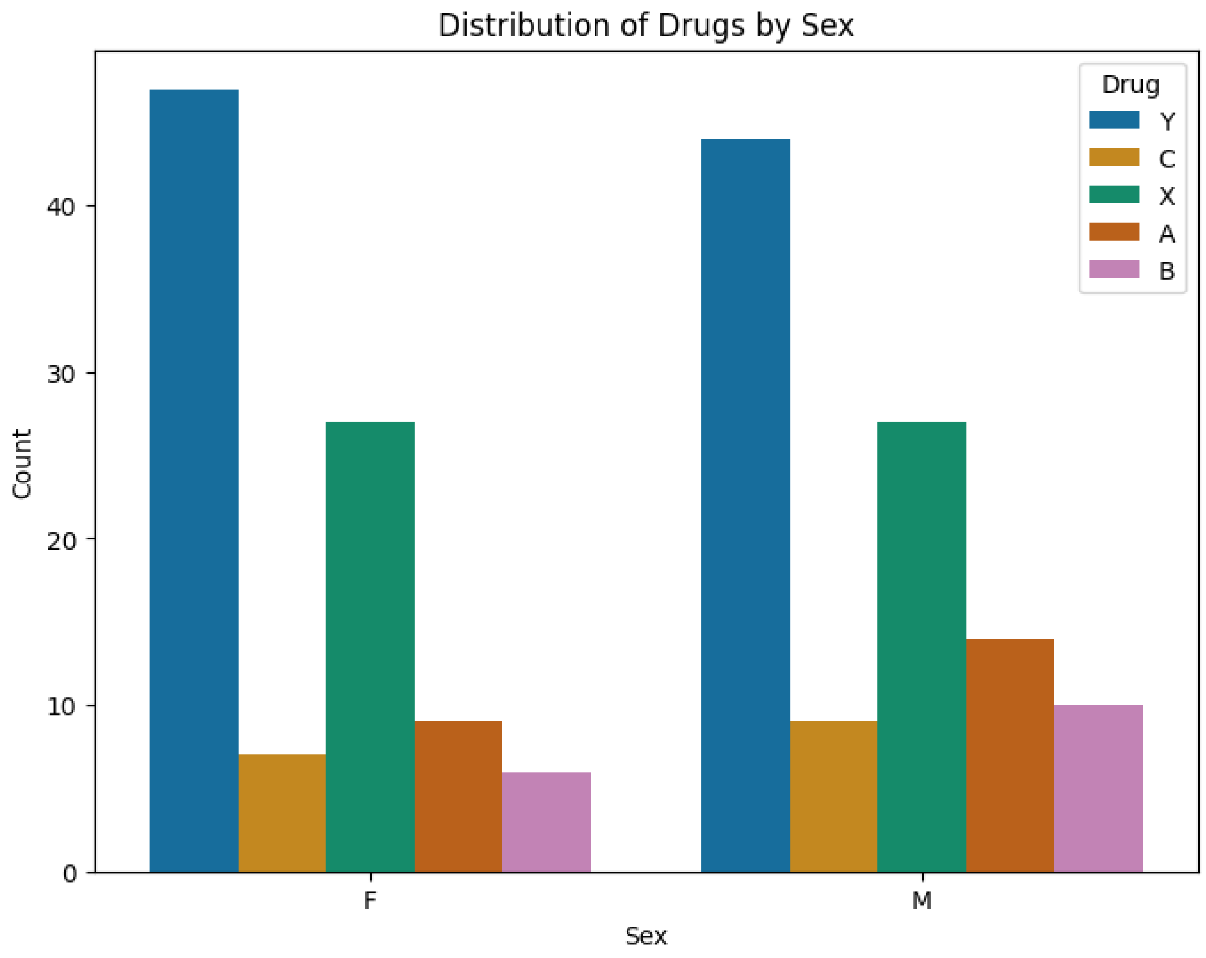
Now bar plots can be used comparing the drugs vs Blood Pressure, Cholesterol, and sex.

In [ ]:
```
#Create plot for Blood Pressure
plt.figure(figsize=(8, 6))
sns.countplot(x="BP", hue="Drug", data=data, palette="colorblind")

# Set plot title and axis labels
plt.title("Distribution of Drugs by Blood Pressure")
plt.xlabel("Blood Pressure")
plt.ylabel("Count")

# Add a legend
plt.legend(title="Drug", loc="upper right")

# Display the plot
plt.show()







#Create plot for Cholesterol
plt.figure(figsize=(8, 6))
sns.countplot(x="Cholesterol", hue="Drug", data=data, palette="colorblind")
```

```python
# Set plot title and axis labels
plt.title("Distribution of Drugs by Cholesterol")
plt.xlabel("Cholesterol")
plt.ylabel("Count")

# Add a legend
plt.legend(title="Drug", loc="upper right")

# Display the plot
plt.show()







#Create plot for Sex
plt.figure(figsize=(8, 6))
sns.countplot(x="Sex", hue="Drug", data=data, palette="colorblind")

# Set plot title and axis labels
plt.title("Distribution of Drugs by Sex")
plt.xlabel("Sex")
plt.ylabel("Count")

# Add a legend
plt.legend(title="Drug", loc="upper right")

# Display the plot
plt.show()
```


Distribution of Drugs by Blood Pressure
Drug
Y
C
X
A
B
Count
0
5
10
15
20
25
30
35
HIGH
LOW
NORMAL
Blood Pressure


Distribution of Drugs by Cholesterol
Count
0
10
20
30
40
HIGH
NORMAL
Cholesterol
Drug
Y
C
X
A
B



Here are the takeaway for comparing the drugs vs variables:

-Age: Drugs Y, X, and C appear to be prescribed across all ages. However, drug A does not appear to be for young or old people while drug B appears to be exclusively for older people.

-Na/K: Drug Y appears to be prescribed to people with higher Na/K ratios. Really anything with a ratio greater than 15. The other drugs, appear to all be for low ratios.

-Blood Pressure: Drug Y appears across all 3 levels of BP, drug C is only found with the low BP population, drugs A and B are only found with the high BP popluation, and drug X is found with the low and normal BP populations.

-Cholesterol: Drug C is only found in the high cholesterol population while the other 4 drugs are evenly dispersed across high and low cholestorol populations.

-Sex: All drugs are evenly distriubted across both sexes.

Models

Dataset Preparation for Models

After going through the EDA and looking at the results of the correlation matrix as well as all the other plots, there is some potential for colinearity. This is indidcated in the plots done in the previous section as well as the takeaways. While collinearity may not be inherently bad, it could result in overfitting. As a result, feature engineering will be done later to address this concern.

First step is to split the dataset into training and testing data. The data will be split into 80% training and 20% testing. The sklearn library will be important for this.

In [ ]:
```
# Import important libraries
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

# Remove the features and set the target variable
# X will remove the features, y will be the target variable
X = data.drop("Drug", axis = 1)
y = data["Drug"]

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

# To understand the training and testing a little better it's best to see the shape
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(160, 5) (160,)
(40, 5) (40,)
```

The shapes appear to be what is expected.

Now, some feature engineering needs to be done on the training and testing datasets to improve the overall performance of the machine learning models. This is because much of the data is categorical values.

In [ ]:
```
# Import from the sklearn library
from sklearn.preprocessing import LabelEncoder

# Apply label encoding to the categorical columns
label_encoder = LabelEncoder()
categorical_columns = ["BP", "Cholesterol", "Sex"]

for column in categorical_columns:
    X[column] = label_encoder.fit_transform(X[column])

# Split the dataset again

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

X_train.head()
```

Out[ ]:

	Age	Sex	BP	Cholesterol	Na_to_K
79	32	0	1	1	10.84
197	52	1	2	0	9.89
38	39	0	2	1	9.71
24	33	0	1	0	33.49
122	34	1	2	0	22.46

In [ ]:
```
X_test.head()
```

Out[ ]:

	Age	Sex	BP	Cholesterol	Na_to_K
95	36	1	1	1	11.42
15	16	0	0	1	15.52
30	18	0	2	1	8.75
158	59	0	1	0	10.44
128	47	1	1	1	33.54

The last thing needed to be done for the models, is to prevent overfitting.

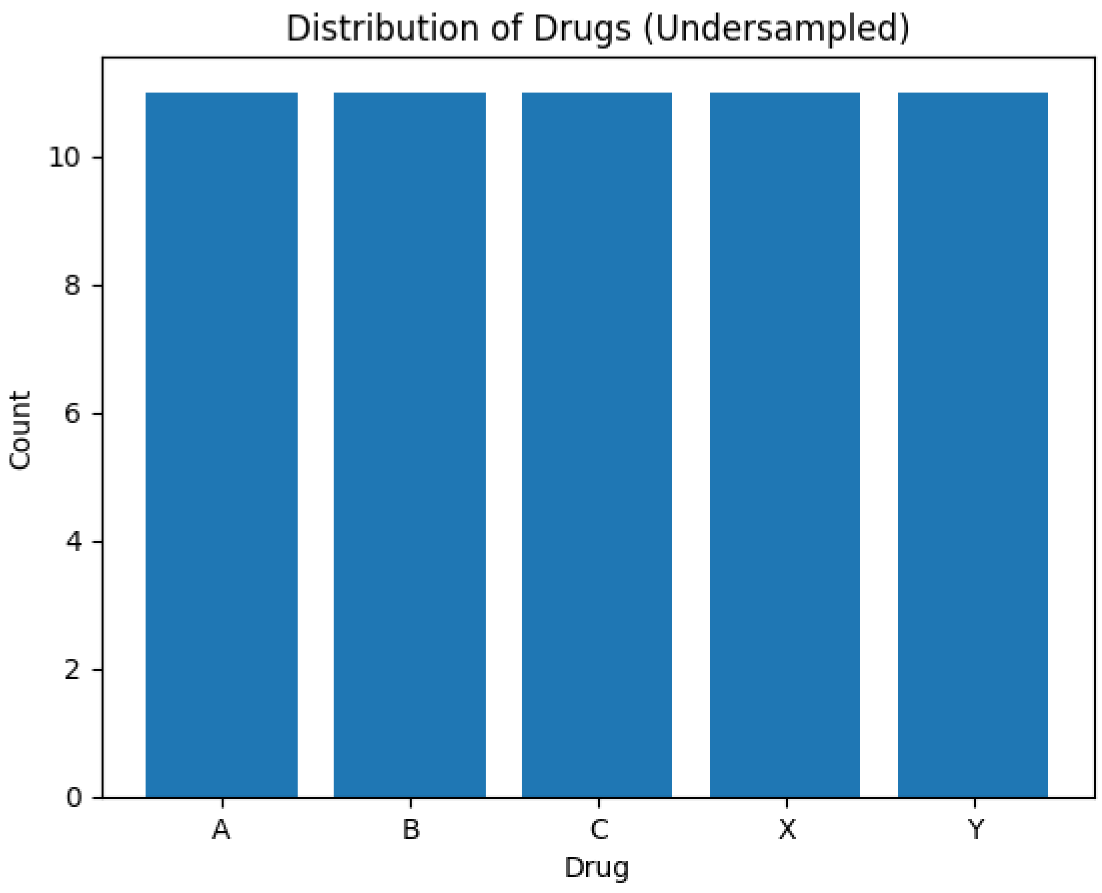
As stated previously, "Y" drug is far more prevelant than the other drugs. Since there's just one drug that is higher than the others, undersampling will be the technique of choice.

In [ ]:
```
# Import the necessary libraries
!pip install imbalanced-learn
from imblearn.under_sampling import RandomUnderSampler

# Use an instance of the under-sampler
undersampler = RandomUnderSampler(random_state = 42)
X_train, y_train = undersampler.fit_resample(X_train, y_train)
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/python/3.10.8/lib/pyth
on3.10/site-packages (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /home/codespace/.local/lib/python3.1
0/site-packages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: scipy>=1.3.2 in /home/codespace/.local/lib/python3.1
0/site-packages (from imbalanced-learn) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /home/codespace/.local/lib/pyt
hon3.10/site-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /home/codespace/.local/lib/python3.1
0/site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/codespace/.local/lib/py
thon3.10/site-packages (from imbalanced-learn) (3.1.0)
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Calculate the count of each drug category in the undersampled data
```

```
drug_counts = y_train.value_counts()

# Create a bar plot to visualize the drug distribution
plt.bar(drug_counts.index, drug_counts.values)
plt.xlabel('Drug')
plt.ylabel('Count')
plt.title('Distribution of Drugs (Undersampled)')
plt.show()
```



Model Selection

Three models will be used on this dataset.

Decision tree classifier: this model was chosen because there does not seem to be much of any linear relationship in the data and decision tree classifiers handle randomization very well.

Logistic Regression: this model was chosen because it is reliant on linear relationships in order to do a good job of predicting. This will be a good way to compare and contrast against the other chosen models. Also, while there does not appear to be a strong linear relationship in the data, there is some. Potentially that means this model could be good at predicting things the other models aren't.

Random forest: this model was chosen for the same reason the decision tree classifier was except that random forests are a more complex version of a decision tree classifier. This should mean that the random forest will perform as well or better than the decision tree.

The first model used for this dataset will be a decision tree classifier.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score




# Make the DTC.  Include parameters in order to introduce some randomness.  Otherwi
clf = DecisionTreeClassifier(max_depth=3, min_samples_split=6, min_samples_leaf=8,

# Training data
clf.fit(X_train, y_train)

# Predicitons on testing data
y_pred_dt = clf.predict(X_test)

# Create a classificaiton report
dt_classification_report = classification_report(y_test, y_pred_dt)

# Show the accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(dt_classification_report)
print("Accuracy:", accuracy_dt)

# Get feature importances
feature_importances = clf.feature_importances_

# Get feature names
feature_names = X.columns

# Sort feature importances in descending order
sorted_indices = feature_importances.argsort()[::-1]
sorted_importances = feature_importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]
```

```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         6
           B       1.00      1.00      1.00         3
           C       0.45      1.00      0.62         5
           X       1.00      0.45      0.62        11
           Y       1.00      1.00      1.00        15

    accuracy                           0.85        40
   macro avg       0.89      0.89      0.85        40
weighted avg       0.93      0.85      0.85        40

Accuracy: 0.85
```

The next model will be made using logistical regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logisticial_regression = LogisticRegression(solver = 'sag', max_iter = 200)
```

```
# Train the model
logisticial_regression.fit(X_train, y_train)

# Predict the testing data
y_pred_lr = logisticial_regression.predict(X_test)


# Create a classificaiton report
lr_classification_report = classification_report(y_test, y_pred_lr)

# Summarize the accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(lr_classification_report)
print("Accuracy:", accuracy_lr)
```

```
              precision    recall  f1-score   support

           A       0.67      0.33      0.44         6
           B       0.38      1.00      0.55         3
           C       1.00      0.60      0.75         5
           X       1.00      0.55      0.71        11
           Y       0.70      0.93      0.80        15

    accuracy                           0.70        40
   macro avg       0.75      0.68      0.65        40
weighted avg       0.79      0.70      0.70        40

Accuracy: 0.7
```

```
/home/codespace/.local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn(
```

The next model to be implemented will be Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

#Create instance of Random Forest.  Make sure to include parameters to introduce ra
rfc = RandomForestClassifier(n_estimators = 10, max_depth = 12, min_samples_split =

# Fit model to the training data
rfc.fit(X_train, y_train)

# Make predictions
y_pred_rf = rfc.predict(X_test)

# Create a classificaiton report
rf_classification_report = classification_report(y_test, y_pred_rf)

# Summarize the accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(rf_classification_report)
print("Accuracy:", accuracy_rf)
```

```
              precision    recall  f1-score   support

           A       0.86      1.00      0.92         6
           B       1.00      1.00      1.00         3
           C       0.62      1.00      0.77         5
           X       1.00      0.73      0.84        11
           Y       1.00      0.93      0.97        15

    accuracy                           0.90        40
   macro avg       0.90      0.93      0.90        40
weighted avg       0.93      0.90      0.90        40

Accuracy: 0.9
```

Results and Summary

Here are the basic accuracy results of each model:

```
In [ ]: pip install tabulate
```

```
Requirement already satisfied: tabulate in /usr/local/python/3.10.8/lib/python3.10/s
ite-packages (0.9.0)
Note: you may need to restart the kernel to use updated packages.
```

```
In [ ]: from tabulate import tabulate
        # Create the table names
        models = [
            ["Decision Tree", accuracy_dt],
            ["Logistic Regression", accuracy_lr],
            ["Random Forest", accuracy_rf]
        ]

        # Table headers
        headers = ["Model", "Accuracy"]

        # Display
        table = tabulate(models, headers, tablefmt = "grid")
        print(table)
```

```
+---------------------+------------+
| Model               |   Accuracy |
+=====================+============+
| Decision Tree       |       0.85 |
+---------------------+------------+
| Logistic Regression |       0.7  |
+---------------------+------------+
| Random Forest       |       0.9  |
+---------------------+------------+
```

Here are some additional metrics to measure the performance of each model:

Decision Tree Classification Report

```
In [ ]: print(dt_classification_report)
```

```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         6
           B       1.00      1.00      1.00         3
           C       0.45      1.00      0.62         5
           X       1.00      0.45      0.62        11
           Y       1.00      1.00      1.00        15

    accuracy                           0.85        40
   macro avg       0.89      0.89      0.85        40
weighted avg       0.93      0.85      0.85        40
```

Logistic Regression Classification Report

```
In [ ]: print(lr_classification_report)
```

```
              precision    recall  f1-score   support

           A       0.67      0.33      0.44         6
           B       0.38      1.00      0.55         3
           C       1.00      0.60      0.75         5
           X       1.00      0.55      0.71        11
           Y       0.70      0.93      0.80        15

    accuracy                           0.70        40
   macro avg       0.75      0.68      0.65        40
weighted avg       0.79      0.70      0.70        40
```

Random Forest Classification Report

```
In [ ]: print(rf_classification_report)
```

```
              precision    recall  f1-score   support

           A       0.86      1.00      0.92         6
           B       1.00      1.00      1.00         3
           C       0.62      1.00      0.77         5
           X       1.00      0.73      0.84        11
           Y       1.00      0.93      0.97        15

    accuracy                           0.90        40
   macro avg       0.90      0.93      0.90        40
weighted avg       0.93      0.90      0.90        40
```

Decision Tree

The overall accuracy of the decision tree is 85%, which is commendable. The model is also precise when looking at drugs A, B, X, and Y but performs poorly with drug C. The f1-score and recall peform well with drugs A, B, and Y but fail to perform well with drugs C and X.

Logistic Regression

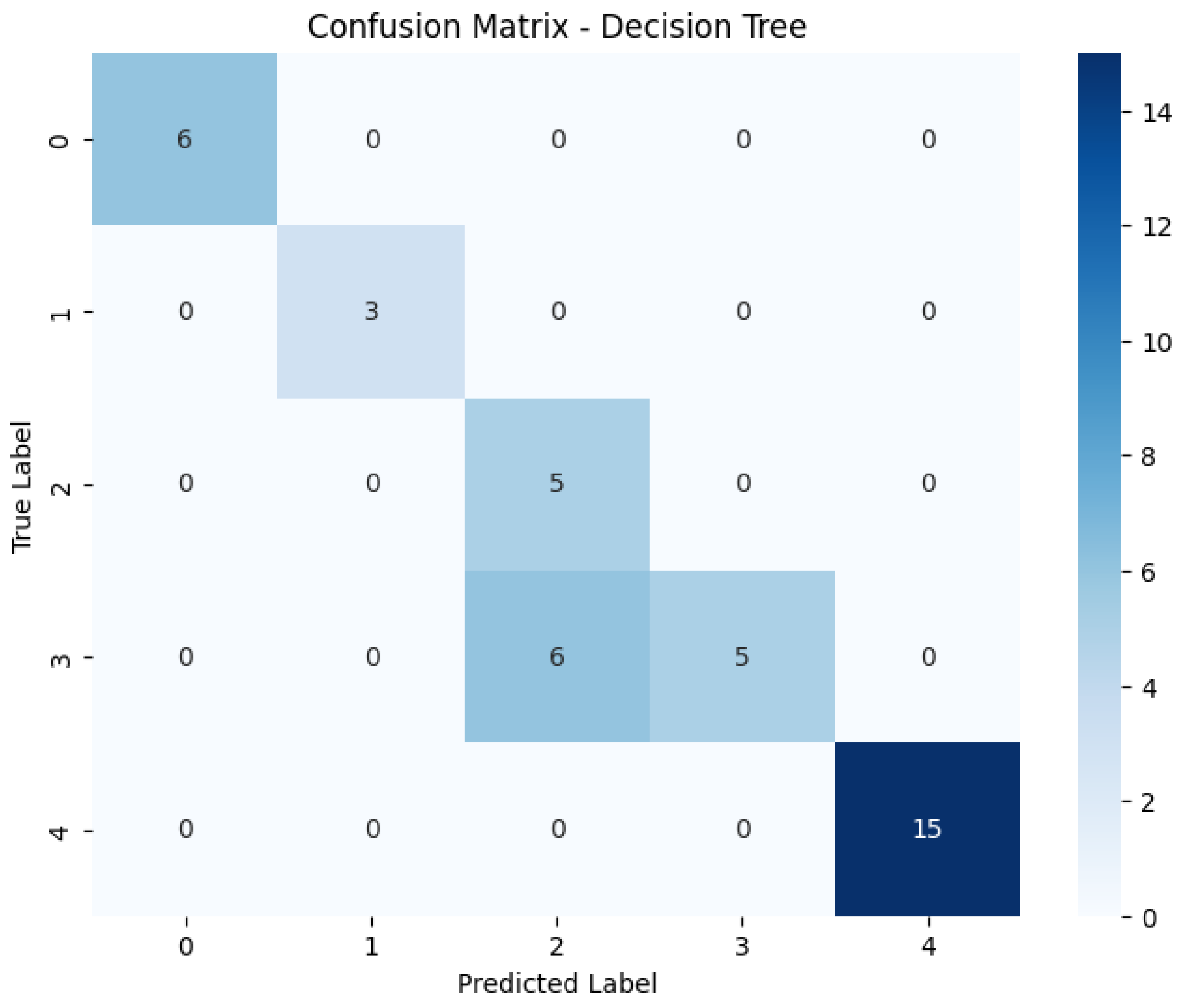
The logistic regression model had an overall accuracy score of 70% which is the lowest of the three models. Generally speaking this model did not perform well in most metrics regardless of the drug type. Interestingly though, it was most precise with drug C unlike the other two models.

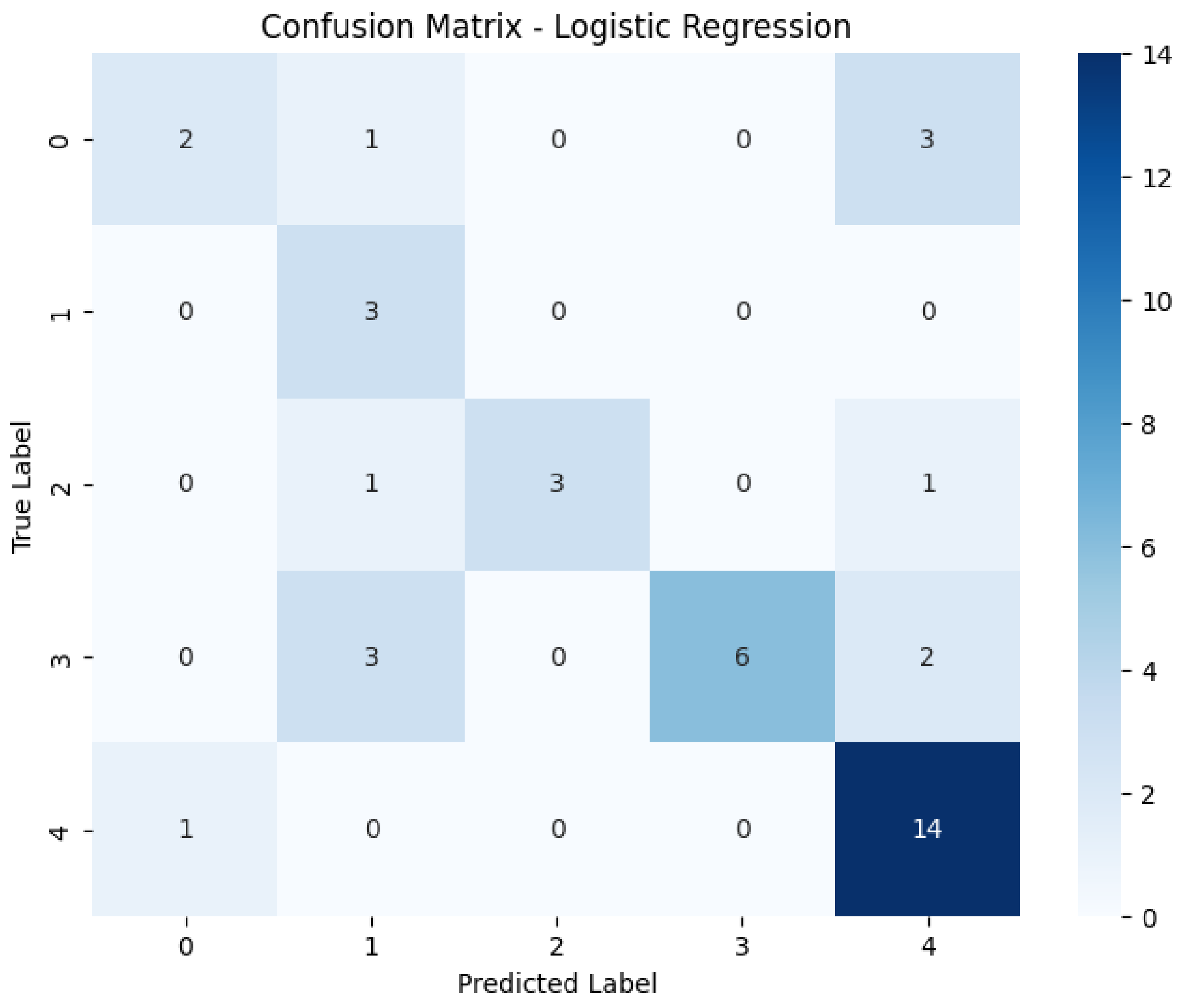
Random Forest

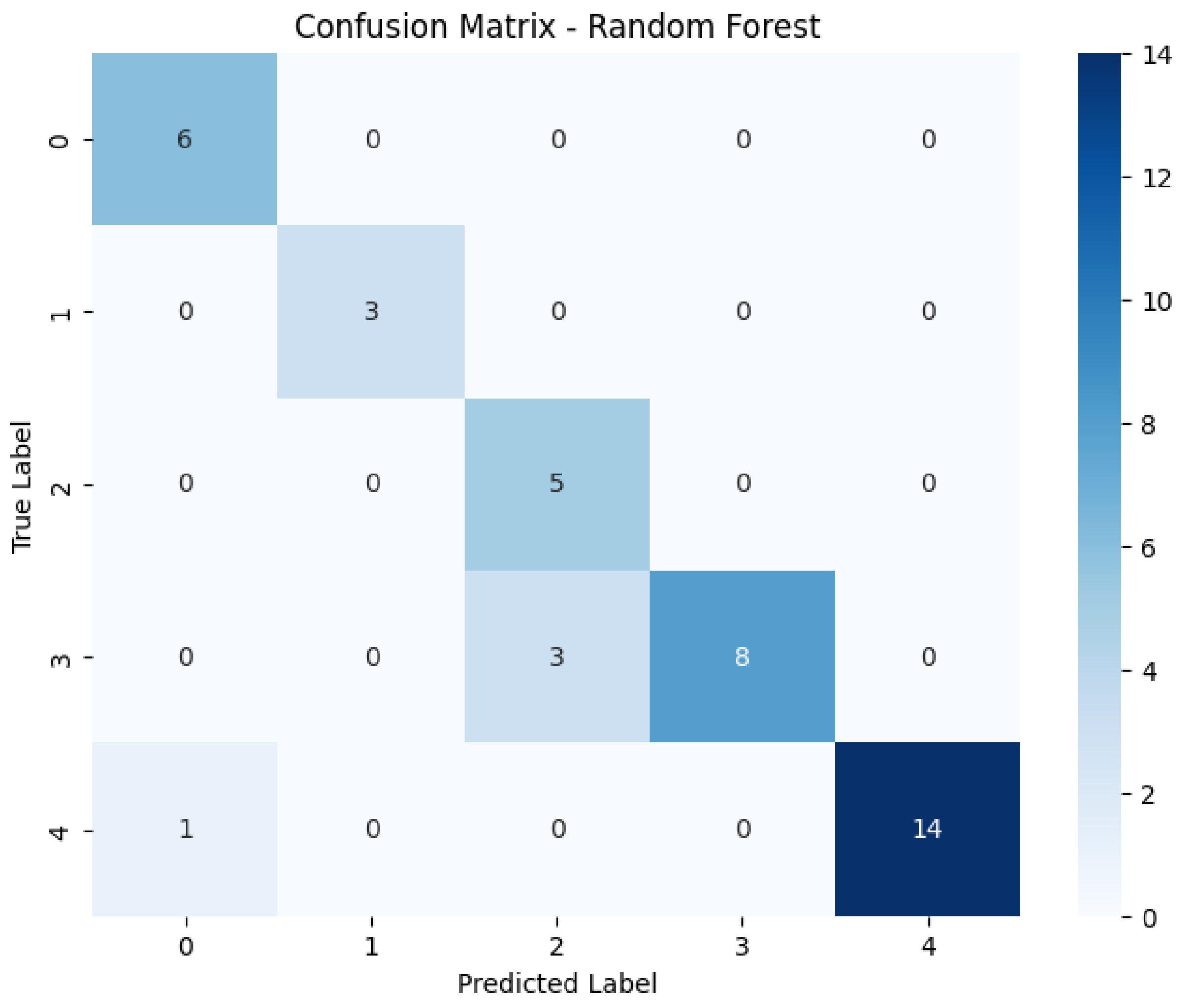
The random forest performed the best of any model with an accuracy of 90%. In fact, this model performed consistently well across all metrics and drugs with the exception of the precision score of drug C.

Confusion Matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix

# Create a list of models and each prediciton they had
models = ["Decision Tree", "Logistic Regression", "Random Forest"]
predictions = [y_pred_dt, y_pred_lr, y_pred_rf]

# Now, go through each model using the zipping method
for model, pred in zip(models, predictions):
    cm = confusion_matrix(y_test, pred)

    # Heatmap
    plt.figure(figsize = (8,6))
    sns.heatmap(cm, annot = True, fmt = 'd', cmap = "Blues")
    plt.title(f"Confusion Matrix - {model}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```


Confusion Matrix - Decision Tree
True Label
Predicted Label
0
1
2
3
4
6
3
5
6
5
15
0
2
4
6
8
10
12
14


Confusion Matrix - Logistic Regression
True Label
Predicted Label
0
1
2
3
4
2 1 0 0 3
0 3 0 0 0
0 1 3 0 1
0 3 0 6 2
1 0 0 0 14
14
12
10
8
6
4
2
0



The confusion matricies further illustrate that the decision tree and random forest models perform well at predicting each drug label. The confusion matrix for the logistic regression also reinforces the prediction limits of the model. Interestingly, the logistic regression model did do a good job predicting drug C unlike the other two models.

Model Improvement

This section is dedicated to trying to improve model performance

In [ ]:
```
# Model improvement attempt for DTC
# Original model (max_depth=3, min_samples_split=5, min_samples_leaf=2, max_feature

# Make the DTC.  Include parameters in order to introduce some randomness.  Otherwi
# CLF introduces two new parameters with no performance improvement
clf = DecisionTreeClassifier(max_depth=3, min_samples_split=6, min_samples_leaf=8,
clf2 = DecisionTreeClassifier(max_depth=2, min_samples_split=7, min_samples_leaf=4,
# Training data
clf.fit(X_train, y_train)
clf2.fit(X_train, y_train)
# Predicitons on testing data
y_pred_dt = clf.predict(X_test)
y_pred_dt_2 = clf2.predict(X_test)
```

```python
# Create a classificaiton report
dt_classification_report = classification_report(y_test, y_pred_dt)
dt_classification_report_2 = classification_report(y_test, y_pred_dt_2)

# Show the accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(dt_classification_report)
print("Accuracy:", accuracy_dt)
accuracy_dt_2 = accuracy_score(y_test, y_pred_dt_2)
print(dt_classification_report_2)
print("Accuracy:", accuracy_dt_2)
```

```
              precision    recall  f1-score   support

           A       1.00      1.00      1.00         6
           B       1.00      1.00      1.00         3
           C       0.45      1.00      0.62         5
           X       1.00      0.45      0.62        11
           Y       1.00      1.00      1.00        15

    accuracy                           0.85        40
   macro avg       0.89      0.89      0.85        40
weighted avg       0.93      0.85      0.85        40

Accuracy: 0.85
              precision    recall  f1-score   support

           A       0.67      1.00      0.80         6
           B       0.00      0.00      0.00         3
           C       0.31      1.00      0.48         5
           X       0.00      0.00      0.00        11
           Y       1.00      1.00      1.00        15

    accuracy                           0.65        40
   macro avg       0.40      0.60      0.46        40
weighted avg       0.51      0.65      0.55        40

Accuracy: 0.65
```

```
/home/codespace/.local/lib/python3.10/site-packages/sklearn/metrics/_classification.
py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/codespace/.local/lib/python3.10/site-packages/sklearn/metrics/_classification.
py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/codespace/.local/lib/python3.10/site-packages/sklearn/metrics/_classification.
py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Two new parameters were introduced into the model which ultimately did not improve performance. In fact, by tuning some of these parameters it degraded the overall performance.

In [ ]:
```
#Model
logisticial_regression = LogisticRegression(solver = 'sag', max_iter = 200)
logisticial_regression2 = LogisticRegression(solver = 'liblinear', max_iter = 200,

# Train the model
logisticial_regression.fit(X_train, y_train)
logisticial_regression2.fit(X_train, y_train)

# Predict the testing data
y_pred_lr = logisticial_regression.predict(X_test)
y_pred_lr2 = logisticial_regression.predict(X_test)

# Create a classificaiton report
lr_classification_report = classification_report(y_test, y_pred_lr)
lr_classification_report2 = classification_report(y_test, y_pred_lr2)

# Summarize the accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(lr_classification_report)
print("Accuracy:", accuracy_lr)
accuracy_lr2 = accuracy_score(y_test, y_pred_lr2)
print(lr_classification_report2)
print("Accuracy:", accuracy_lr2)
```

```
              precision    recall  f1-score   support

           A       0.67      0.33      0.44         6
           B       0.38      1.00      0.55         3
           C       1.00      0.60      0.75         5
           X       1.00      0.55      0.71        11
           Y       0.70      0.93      0.80        15

    accuracy                           0.70        40
   macro avg       0.75      0.68      0.65        40
weighted avg       0.79      0.70      0.70        40

Accuracy: 0.7
              precision    recall  f1-score   support

           A       0.67      0.33      0.44         6
           B       0.38      1.00      0.55         3
           C       1.00      0.60      0.75         5
           X       1.00      0.55      0.71        11
           Y       0.70      0.93      0.80        15

    accuracy                           0.70        40
   macro avg       0.75      0.68      0.65        40
weighted avg       0.79      0.70      0.70        40

Accuracy: 0.7
```

```
/home/codespace/.local/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn(
```

These changes to the parameters also did not improve performance of the model.

In [ ]:
```
#Create instance of Random Forest.  Make sure to include parameters to introduce ra
rfc = RandomForestClassifier(n_estimators = 10, max_depth = 12, min_samples_split =
rfc2 = RandomForestClassifier(n_estimators = 1000, max_depth = 1, min_samples_split

# Fit model to the training data
rfc.fit(X_train, y_train)
rfc2.fit(X_train, y_train)

# Make predictions
y_pred_rf = rfc.predict(X_test)
y_pred_rf2 = rfc.predict(X_test)

# Create a classificaiton report
rf_classification_report = classification_report(y_test, y_pred_rf)
rf_classification_report2 = classification_report(y_test, y_pred_rf2)

# Summarize the accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(rf_classification_report)
print("Accuracy:", accuracy_rf)
accuracy_rf2 = accuracy_score(y_test, y_pred_rf2)
print(rf_classification_report)
print("Accuracy:", accuracy_rf2)
```

```
              precision    recall  f1-score   support

           A       0.86      1.00      0.92         6
           B       1.00      1.00      1.00         3
           C       0.62      1.00      0.77         5
           X       1.00      0.73      0.84        11
           Y       1.00      0.93      0.97        15

    accuracy                           0.90        40
   macro avg       0.90      0.93      0.90        40
weighted avg       0.93      0.90      0.90        40

Accuracy: 0.9
              precision    recall  f1-score   support

           A       0.86      1.00      0.92         6
           B       1.00      1.00      1.00         3
           C       0.62      1.00      0.77         5
           X       1.00      0.73      0.84        11
           Y       1.00      0.93      0.97        15

    accuracy                           0.90        40
   macro avg       0.90      0.93      0.90        40
weighted avg       0.93      0.90      0.90        40

Accuracy: 0.9
```

Similarly, changing the parameters did not improve the performance of this model.

Discussion and Conclusions

The biggest conclusion in comparing these models is that the random forest performed the best by all metrics and should be the recommneded model if trying to use this in practice. The decision tree also performed well but despite attempts at improving the model there does not seem to be room for improvement. The logistic regression model performed the worst out of all three models and could not be improved.

These machine learning models were all relatively easy to set up and work with. There were not too many issues when it came to adjusting the models in order to perform well.

One of the most noteable things about these models was despite turning the parameters, there was not any improvement in any model. This is probably due to the fact that there was feature engineering done prior to setting the testing and training data.

Based on the EDA done earlier, the most likely conclusion for the logistic regression model performing poorly is that there was not a strong sense of linear relationships in the dataset. Logistic regression models are heavily reliant on having linear relationships in the data in order to help predict. However, it should be noted that this model did the best at identifying drug C.

On the other hand, decision trees and random forests both handle a lack of linear relationships well. Random forests in particular are good at capturing more complex relationships in order to make accurate predictions.

The best way to improve these models overall may be to make the dataset bigger. Larger datasets always bode well for making models more robust and have a better representation of how well they may perform. With larger datasets it is possible more linear relationships will be seen which would help the logistic regression model. Both the decision trees and random forests are already capable of handling complex data so this would just make the models more robust.

Citations

Thanks to Pratham Tripathi who supplied this dataset on Kaggle: Pratham Tripathi. (June 2020). Drug Classification, version 1 https://www.kaggle.com/datasets/prathamtripathi/drug-classification

For information about Na/K relationships in the human body:

Mirmiran, P., Gaeini, Z., Bahadoran, Z., Ghasemi, A., Norouzirad, R., Tohidi, M., & Azizi, F. (2021, January 6). Urinary sodium-to-potassium ratio: A simple and useful indicator of diet quality in population-based studies - European Journal of Medical Research. BioMed Central. https://eurjmedres.biomedcentral.com/articles/10.1186/s40001-020-00476-5#citeas