Assignment 3

Drug Clasification

Importing Libraries

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px
import warnings
warnings.simplefilter("ignore")
```

Loading up the data

In [6]:
```
data = pd.read_csv("drug200.csv")
data.head()
```

Out[6]:

	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

- **Age**: Age of the patient
- **Sex**: Gender of the patients
- **BP**: Blood Pressure of the patient
- **Cholesterol**: Cholesterol of the patient

- **Na_to_K**: Sodium to Potassium ratio in patient's blood
- **Drug**: Drug type give to patients

In [7]:
```python
# Looking for missing values in the dataset
data.isna().sum()
```

Out[7]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

In [8]:
```python
data.shape
```

Out[8]:
```
(200, 6)
```

In [9]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

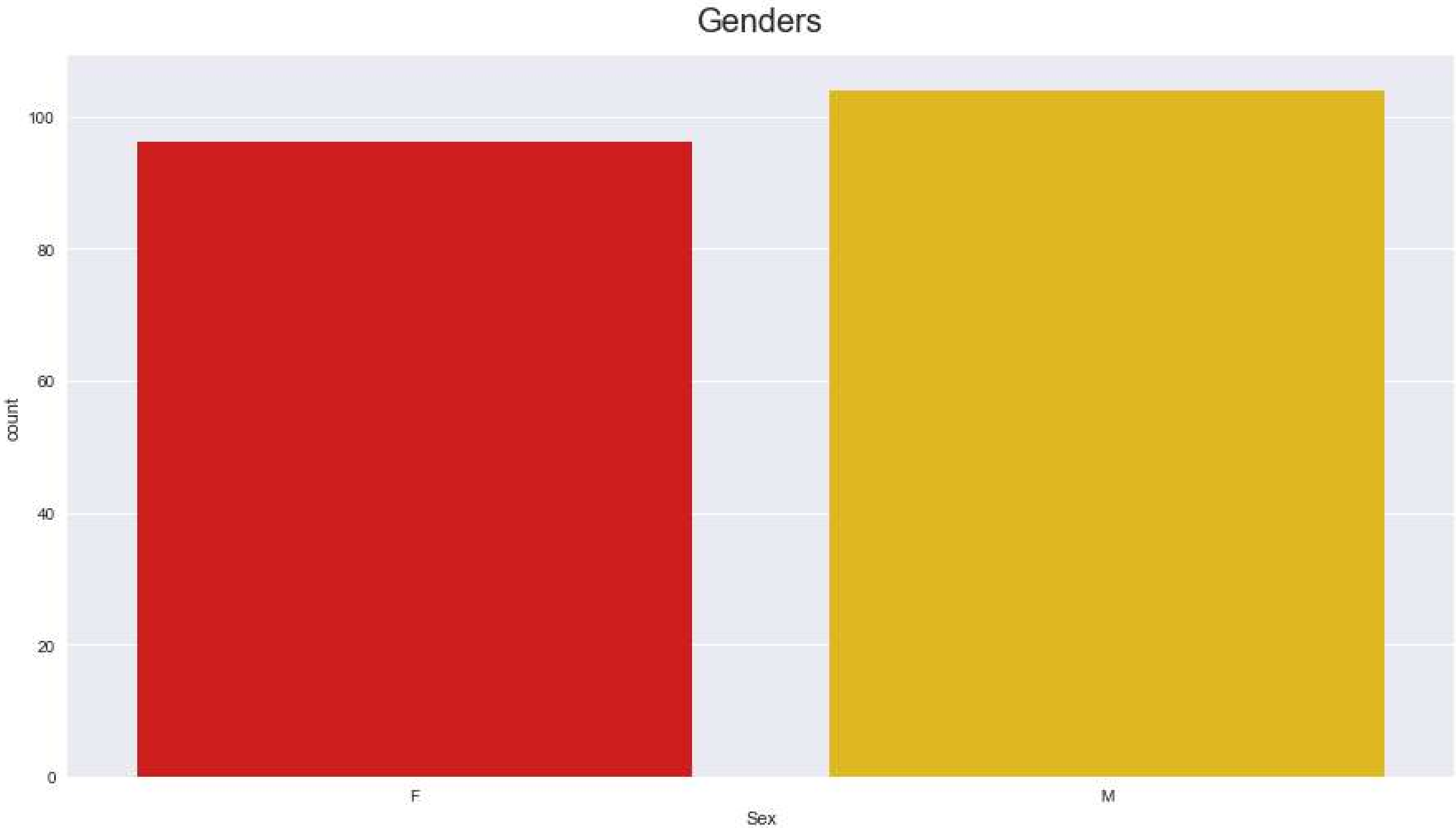
In [10]:
```python
data.dtypes
```

Out[10]:
```
Age              int64
Sex             object
BP              object
Cholesterol     object
Na_to_K        float64
Drug            object
dtype: object
```

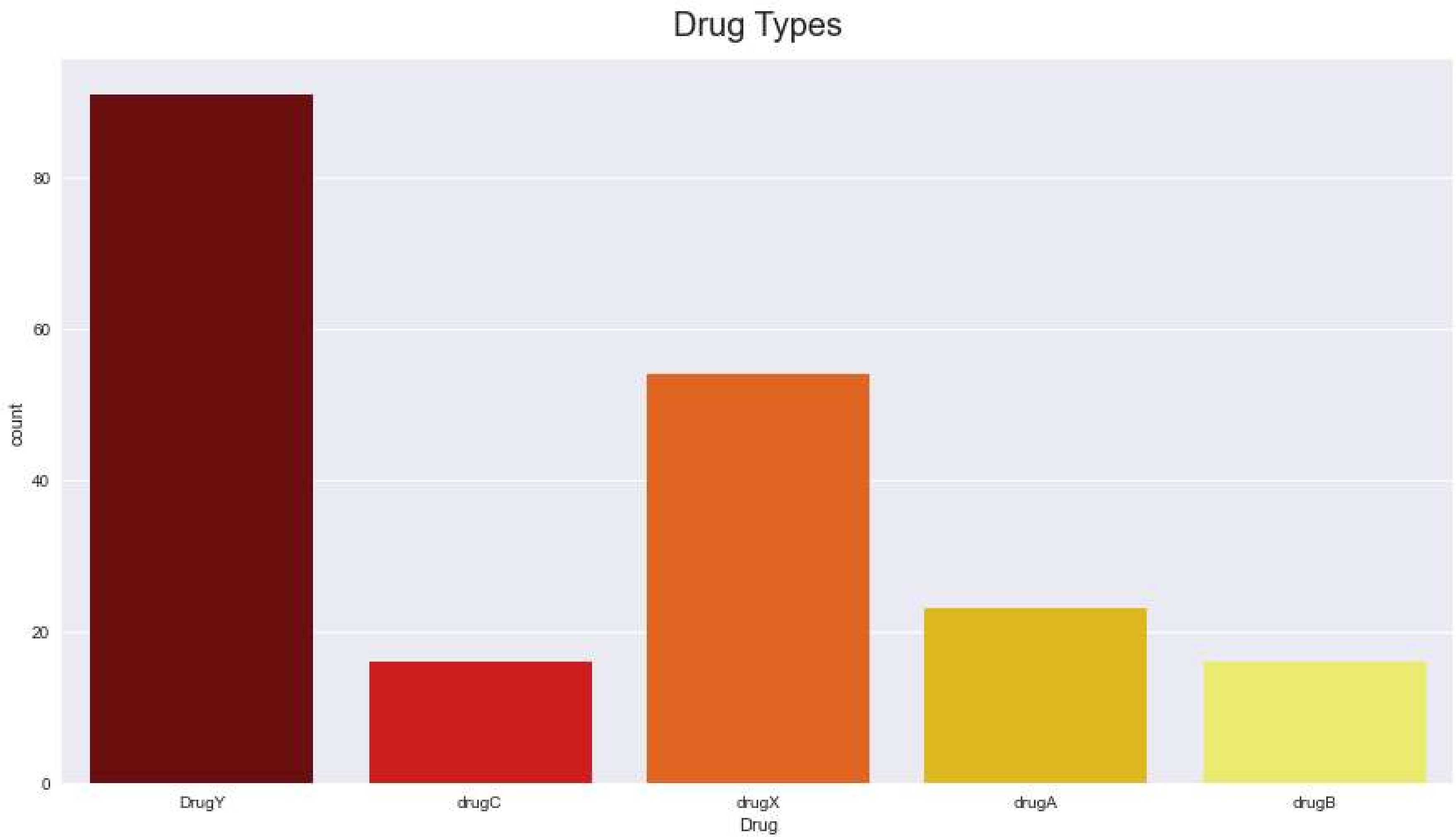
In [11]:
```python
data['Drug'].value_counts()
```

Out[11]:
```
DrugY    91
drugX    54
drugA    23
drugB    16
drugC    16
Name: Drug, dtype: int64
```

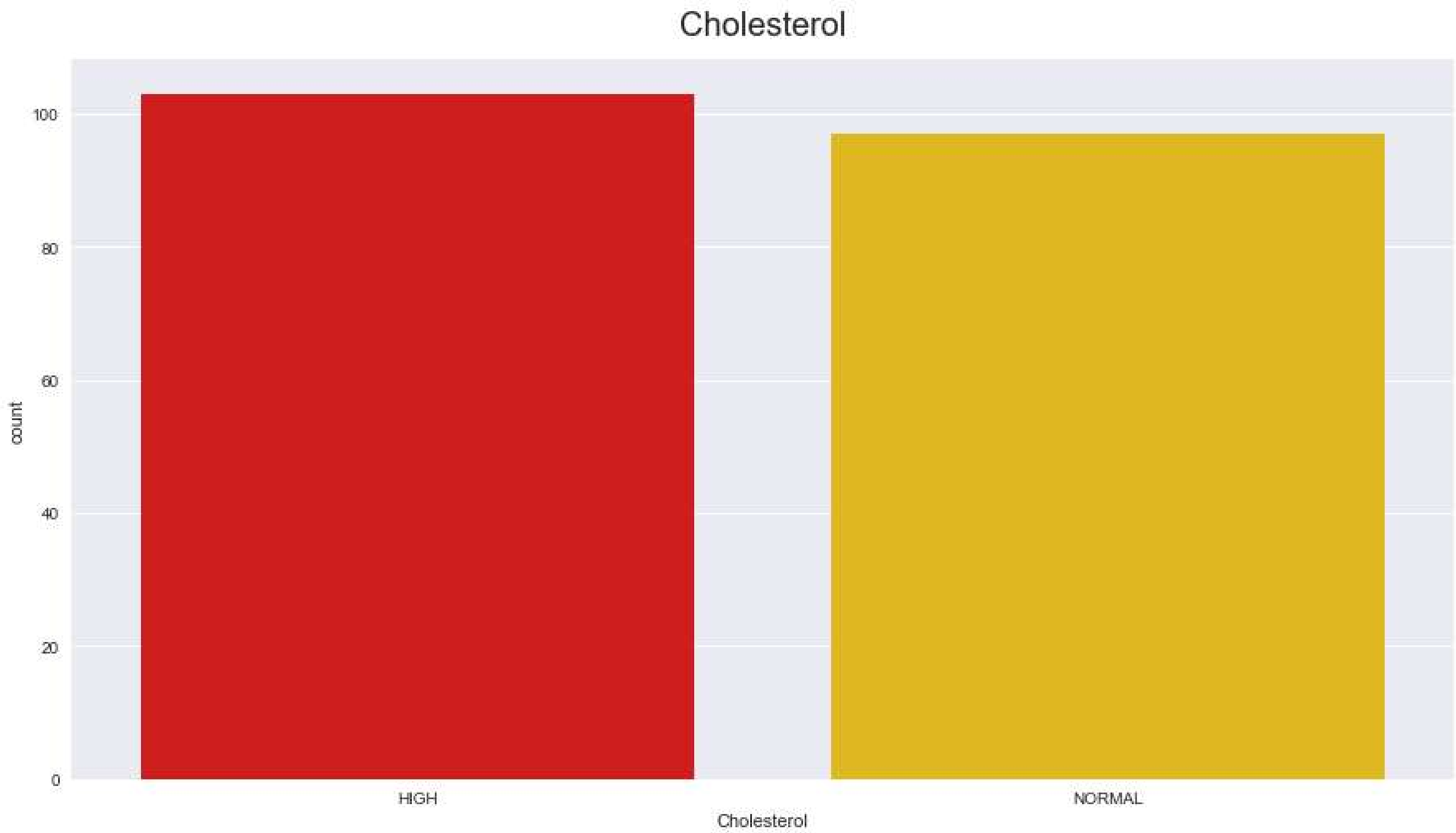
In [12]:
```
plt.style.use("seaborn")
plt.figure(figsize=(15,8))
plt.title("Genders", fontsize=20, y=1.02)
sns.countplot(x = data.Sex, palette="hot")
plt.show()
```



In [13]:
```
plt.figure(figsize=(15,8))
plt.title("Drug Types", fontsize=20, y=1.02)
sns.countplot(x = data.Drug, palette="hot")
plt.show()
```

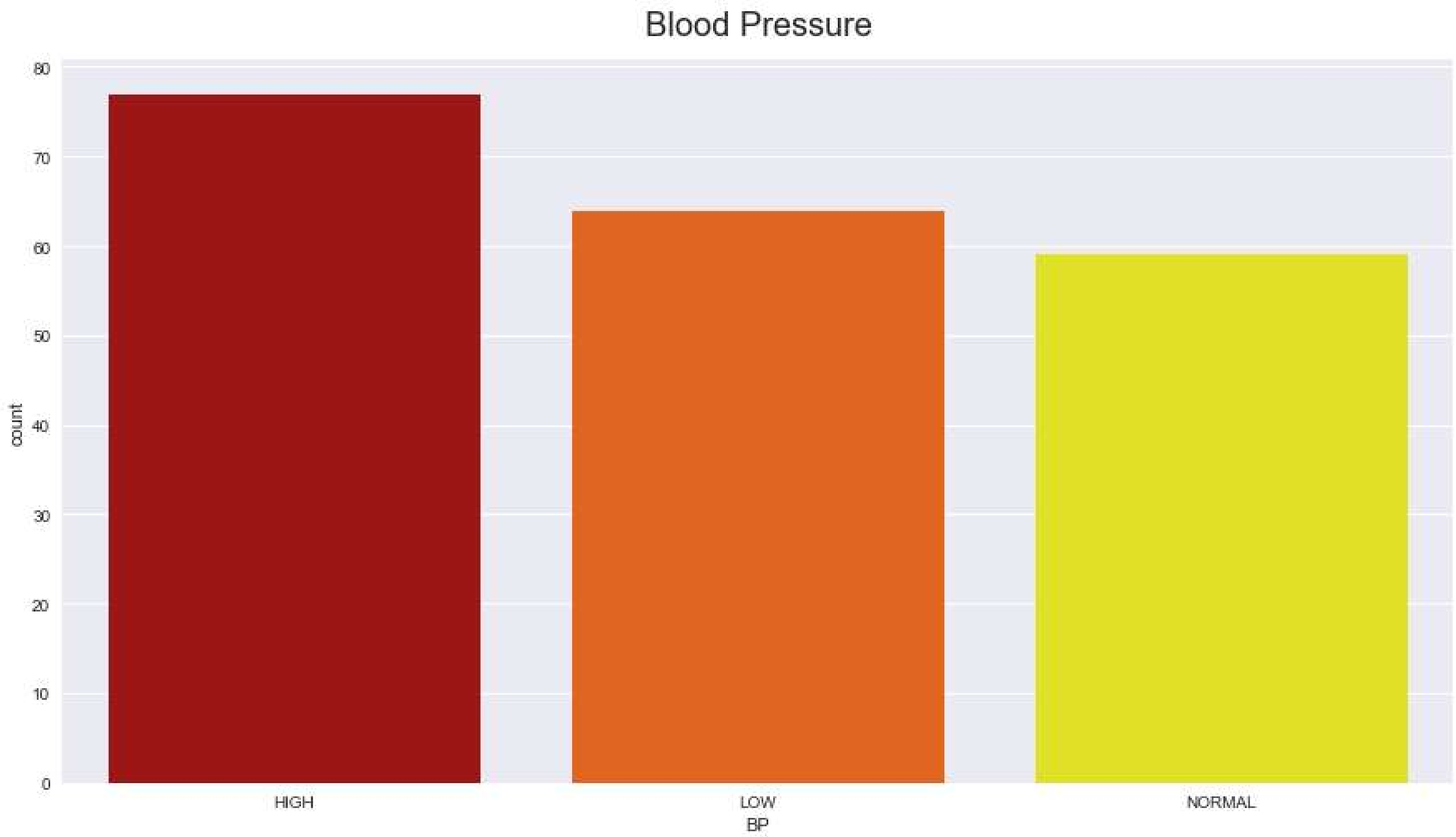


In [14]:
```
plt.figure(figsize=(15,8))
plt.title("Cholesterol", fontsize=20, y=1.02)
sns.countplot(x = data.Cholesterol, palette="hot")
plt.show()
```



In [15]:

```
plt.figure(figsize=(15,8))
plt.title("Blood Pressure", fontsize=20, y=1.02)
sns.countplot(x = data.BP, palette="hot")
plt.show()
```



In [16]:
```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(15,8))
sns.histplot(data["Age"], kde=True, bins=25, color="firebrick")
plt.title("Age of the patients", fontsize=20, y=1.02)
ax.set_xlabel("Age",fontsize=15);
```

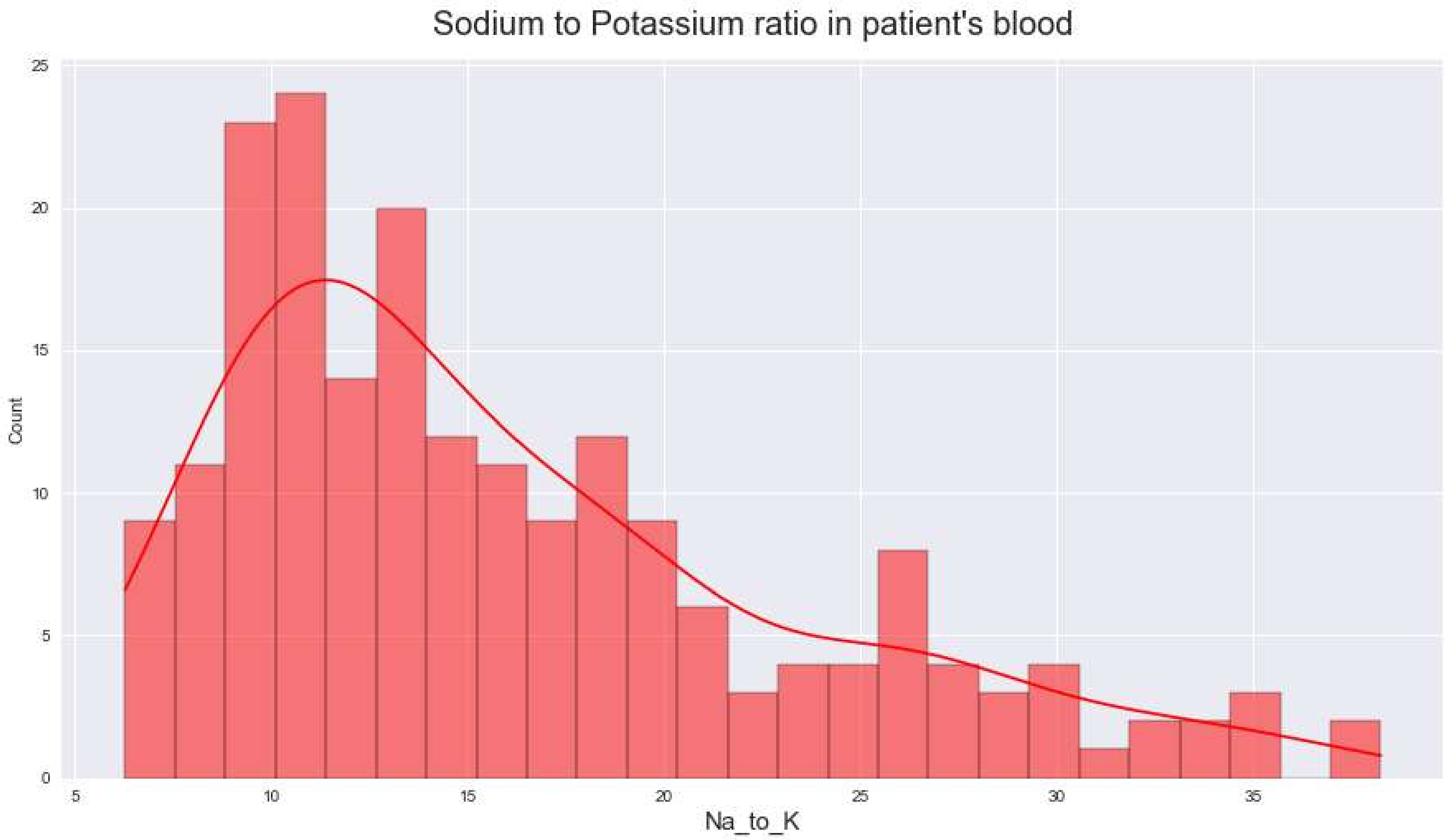
In [17]:
```
data.head(1)
```

Out[17]:

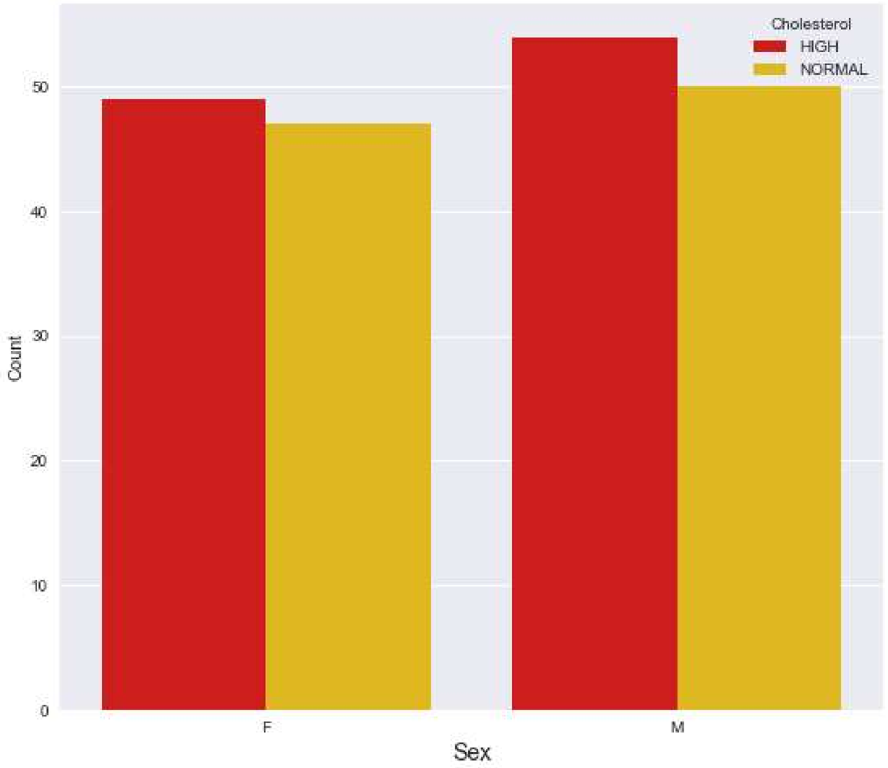
	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY

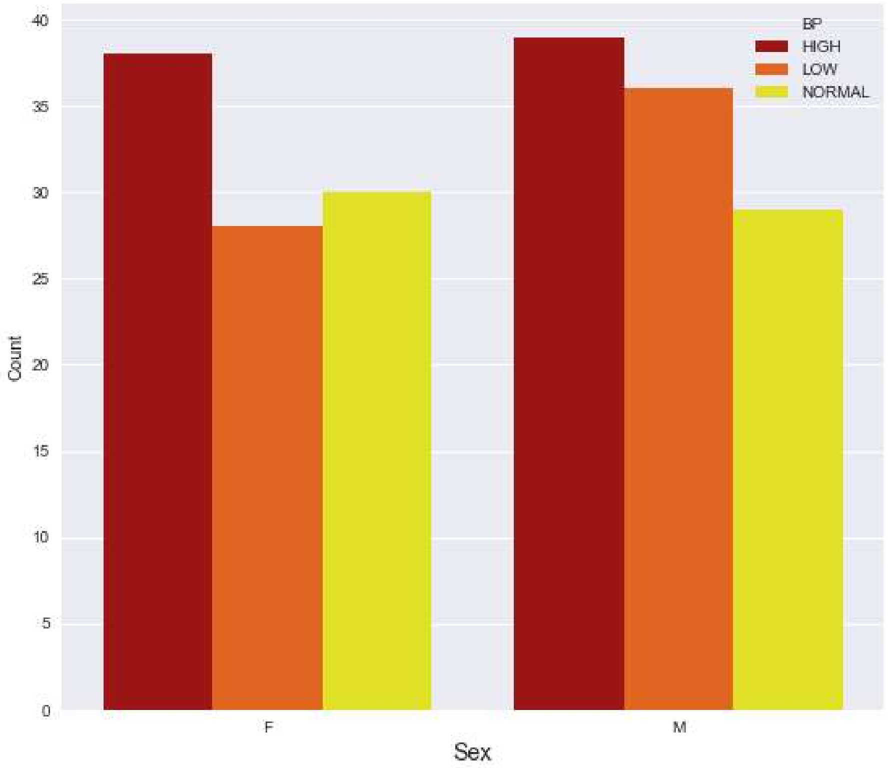
In [18]:
```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(15,8))
sns.histplot(data["Na_to_K"], color="red", kde=True, bins=25)
```

```
plt.title("Sodium to Potassium ratio in patient's blood", fontsize=20, y=1.02)
ax.set_xlabel("Na_to_K",fontsize=15);
```



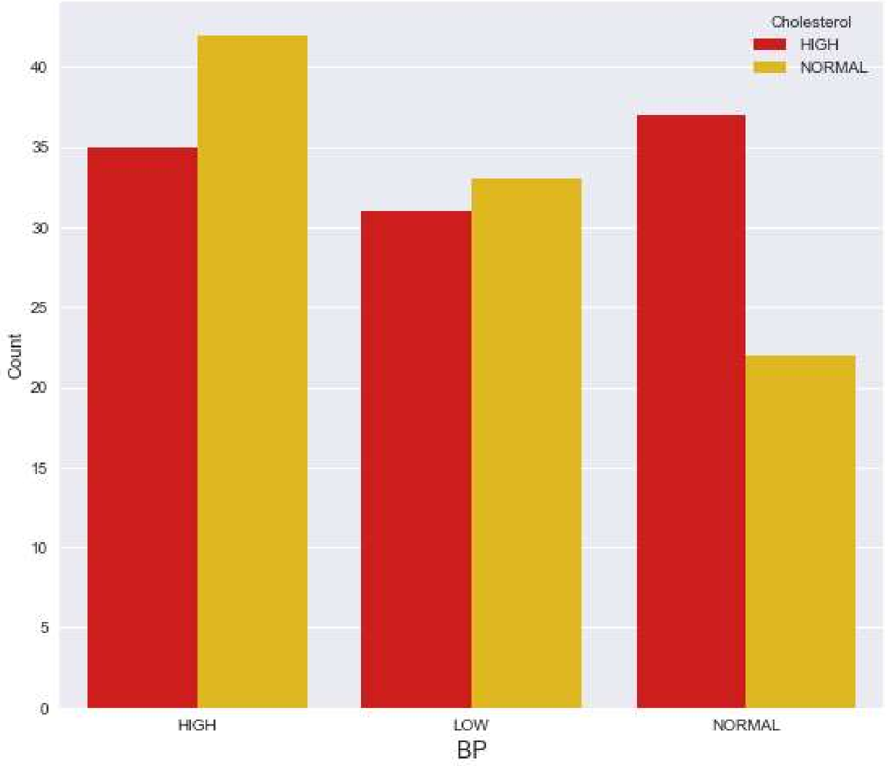
In [19]:
```
plt.style.use("seaborn")
fig, ax =plt.subplots(1,2, figsize=(20,8))

sns.barplot(x = "Sex", y = "Count", hue = "Cholesterol", data = data.groupby(["Sex", "Cholesterol"]).size().reset_index(name = "Co
ax[0].set_xlabel("Sex",fontsize=14);

sns.barplot(x = "Sex", y = "Count", hue = "BP", data = data.groupby(["Sex", "BP"]).size().reset_index(name = "Count"), palette="ho
ax[1].set_xlabel("Sex",fontsize=14);
```

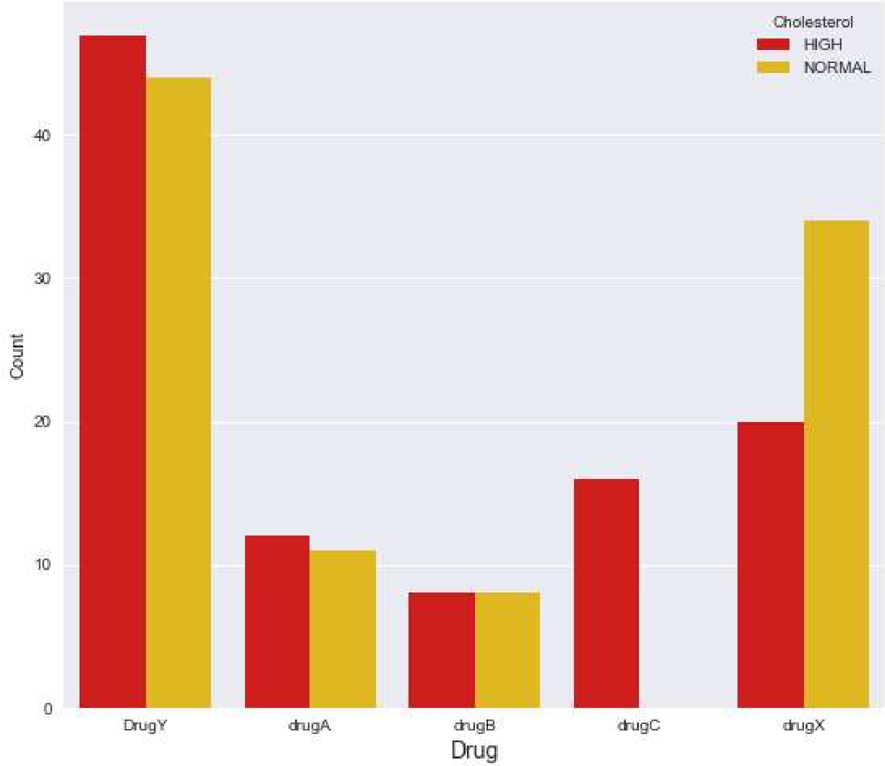




In [20]:

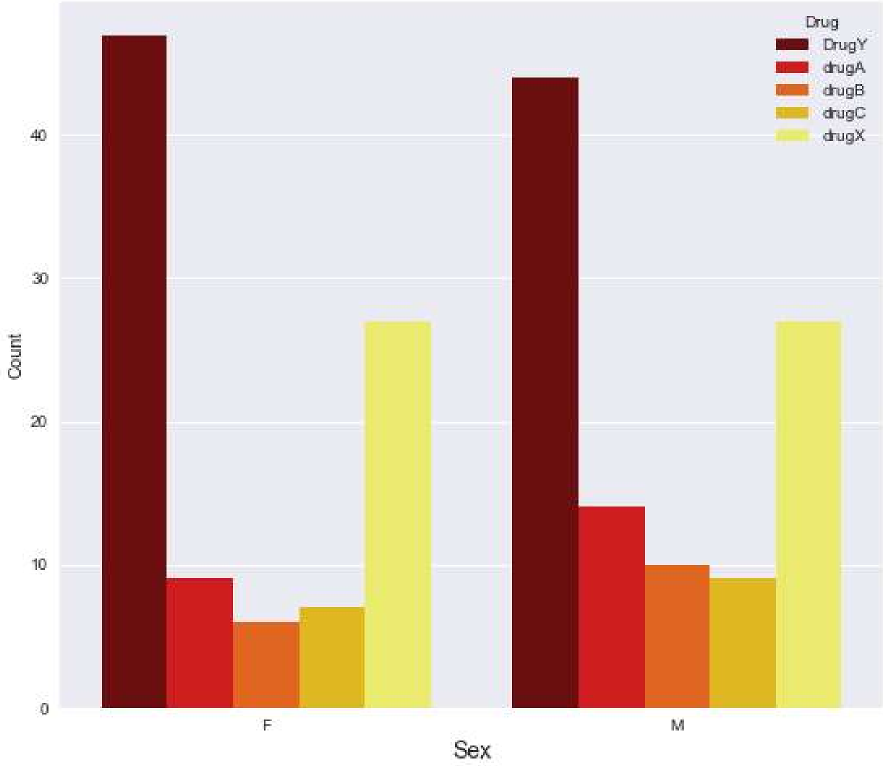
```
plt.style.use("seaborn")
fig, ax =plt.subplots(1,2, figsize=(20,8))

sns.barplot(x = "BP", y = "Count", hue = "Cholesterol", data = data.groupby(["BP", "Cholesterol"]).size().reset_index(name = "Coun
ax[0].set_xlabel("BP",fontsize=14);

sns.barplot(x = "Drug", y = "Count", hue = "Cholesterol", data = data.groupby(["Drug", "Cholesterol"]).size().reset_index(name = "
ax[1].set_xlabel("Drug",fontsize=14);
```

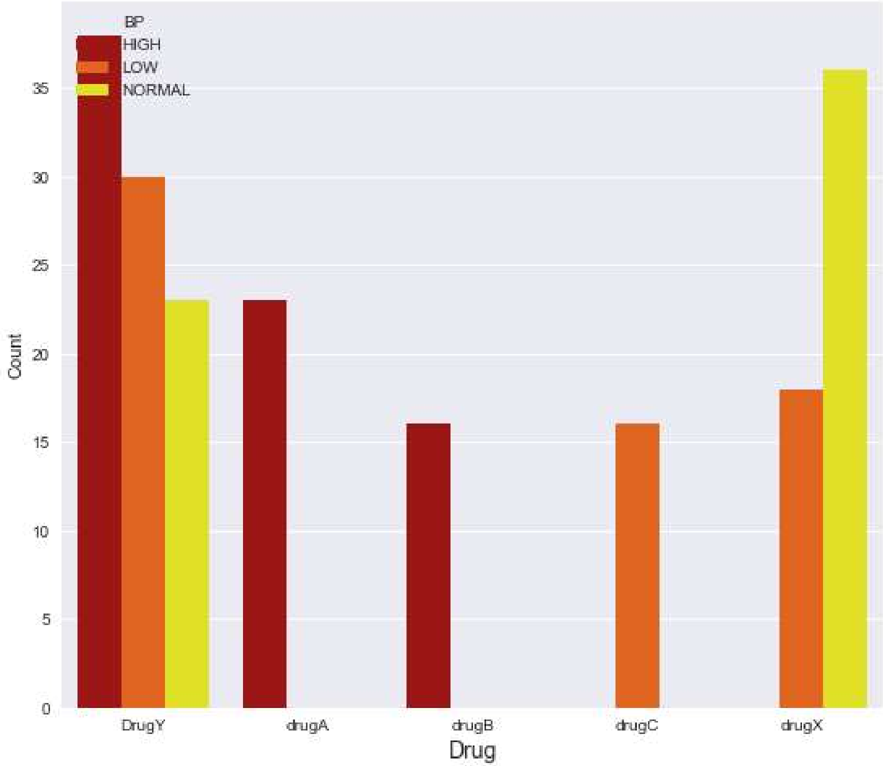




In [21]:

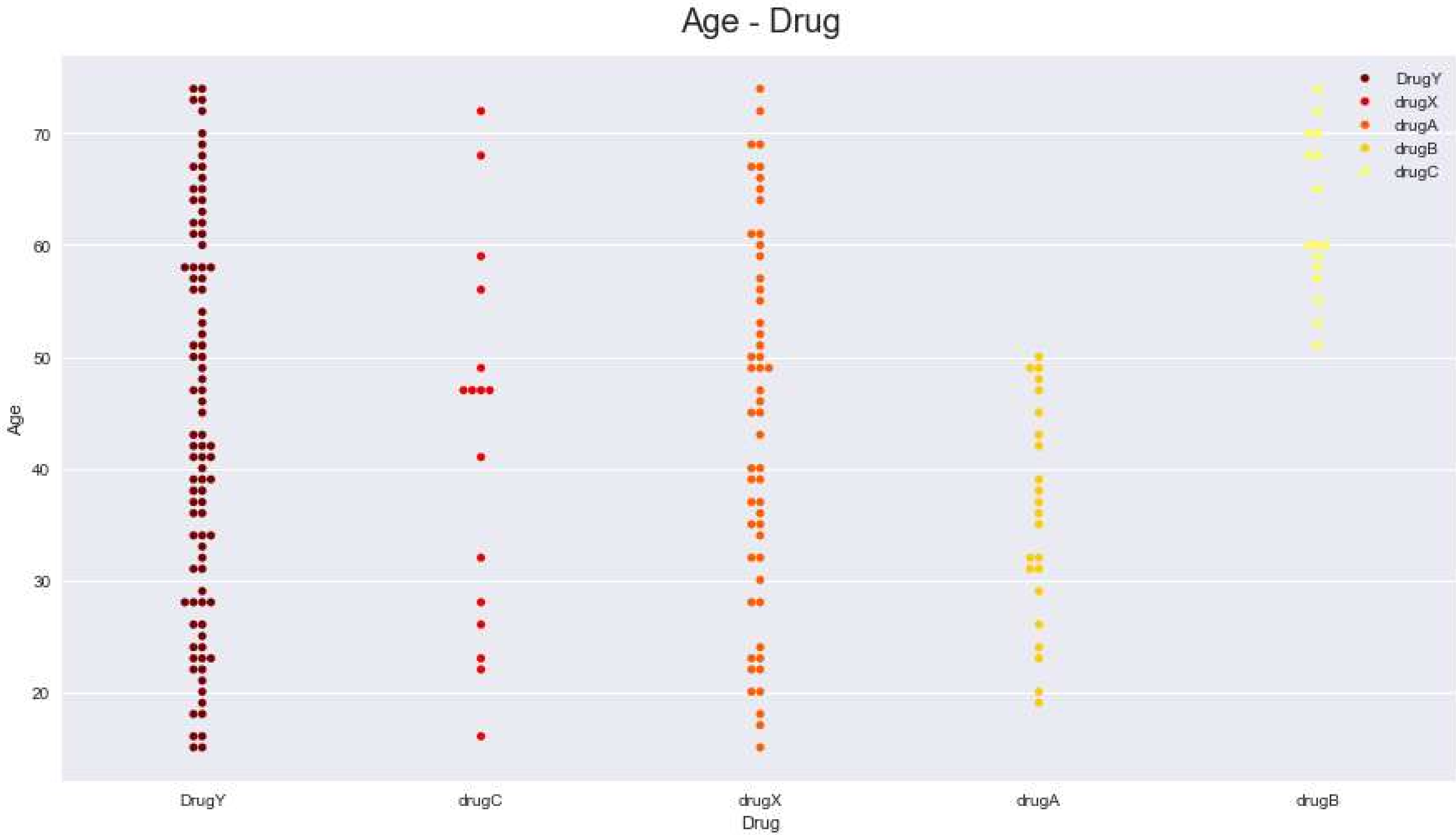
```
plt.style.use("seaborn")
fig, ax =plt.subplots(1,2, figsize=(20,8))

sns.barplot(x = "Sex", y = "Count", hue = "Drug", data = data.groupby(["Sex", "Drug"]).size().reset_index(name = "Count"), palette
ax[0].set_xlabel("Sex",fontsize=14);

sns.barplot(x = "Drug", y = "Count", hue = "BP", data = data.groupby(["Drug", "BP"]).size().reset_index(name = "Count"), palette="
ax[1].set_xlabel("Drug",fontsize=14);
```





In [22]:

```
plt.figure(figsize = (15,8))
sns.swarmplot(x = "Drug", y = "Age", data = data, palette="hot")
plt.legend(data.Drug.value_counts().index)
plt.title("Age - Drug", fontsize=20, y=1.02)
plt.show()
```



In [23]:
```
data.dtypes
```

Out[23]:
```
Age              int64
Sex             object
BP              object
Cholesterol     object
Na_to_K        float64
Drug            object
dtype: object
```

In [24]:
```
# Converting the non-numeric values into numeric values
```

```
data['Sex'] = data['Sex'].map({'M': 1, 'F': 2})
data['BP'] = data['BP'].map({'HIGH': 1, "NORMAL" : 2, "LOW" : 3})
data['Cholesterol'] = data['Cholesterol'].map({'HIGH': 1, "NORMAL" : 2})
data["Drug"] = data["Drug"].map({"DrugY":1, "drugC":2, "drugX":3, "drugA":4, "drugB":5})
```

In [25]:

```
data.head()
```

Out[25]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	2	1	1	25.355	1
1	47	1	3	1	13.093	2
2	47	1	3	1	10.114	2
3	28	2	2	1	7.798	3
4	61	2	3	1	18.043	1

Splitting the data into training and test datasets

Here, we are trying to predict the Drug type that is to be prescribed to the patient using the given data. Hence, the "Drug Type" will be the y label and rest of the data will be the X or the input data.

In [26]:

```
# X data
X = data.drop("Drug", axis=1)
X.head()
```

Out[26]:

	Age	Sex	BP	Cholesterol	Na_to_K
0	23	2	1	1	25.355
1	47	1	3	1	13.093
2	47	1	3	1	10.114
3	28	2	2	1	7.798
4	61	2	3	1	18.043

In [27]:

```
# y data
y = data["Drug"]
```

```
y.head()
```

Out[27]:
```
0    1
1    2
2    2
3    3
4    1
Name: Drug, dtype: int64
```

In [28]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [29]:
```
len(X_train), len(X_test)
```

Out[29]:
```
(160, 40)
```

In [30]:
```
# Scaling the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

In [31]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

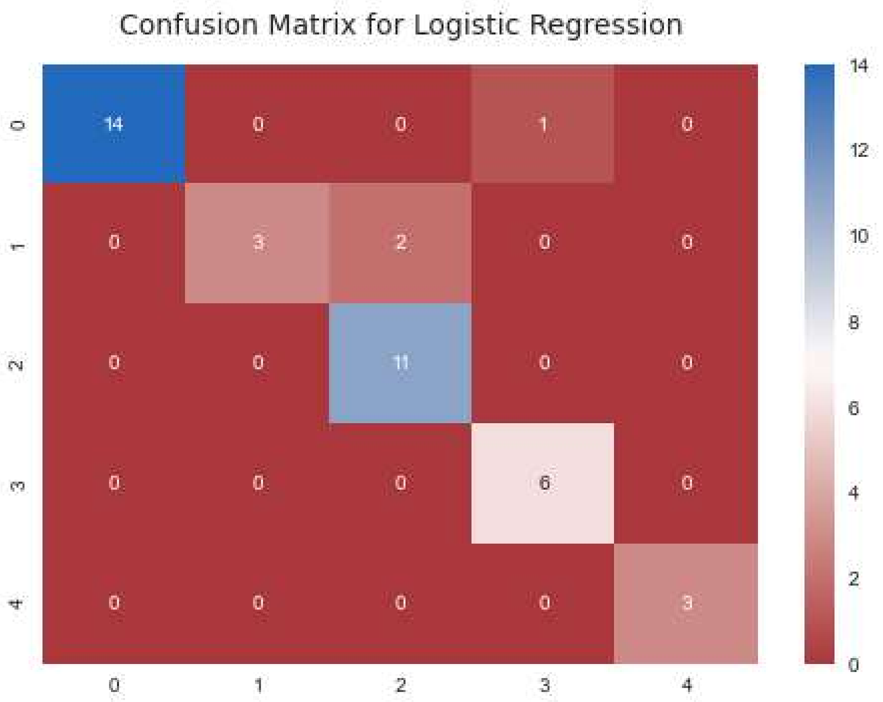
Out[31]:
```
LogisticRegression()
```

In [32]:
```
LogisticRegressionScore = lr.score(X_test, y_test)
print("Accuracy obtained by Logistic Regression model:",LogisticRegressionScore*100)
```

```
Accuracy obtained by Logistic Regression model: 92.5
```

In [49]:
```
# Having a look at the confusion matrix for Logistic Regression

from sklearn.metrics import confusion_matrix, classification_report

y_pred_lr = lr.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_lr)
```

```
sns.heatmap(cf_matrix, annot=True, cmap="vlag_r")
plt.title("Confusion Matrix for Logistic Regression", fontsize=14, fontname="Helvetica", y=1.03);
```



In [34]:
```
# Having a look at the classification report of Logistic Regression

from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

           1       1.00      0.93      0.97        15
           2       1.00      0.60      0.75         5
           3       0.85      1.00      0.92        11
           4       0.86      1.00      0.92         6
           5       1.00      1.00      1.00         3

    accuracy                           0.93        40
   macro avg       0.94      0.91      0.91        40
weighted avg       0.94      0.93      0.92        40
```

Random Forest Classifier

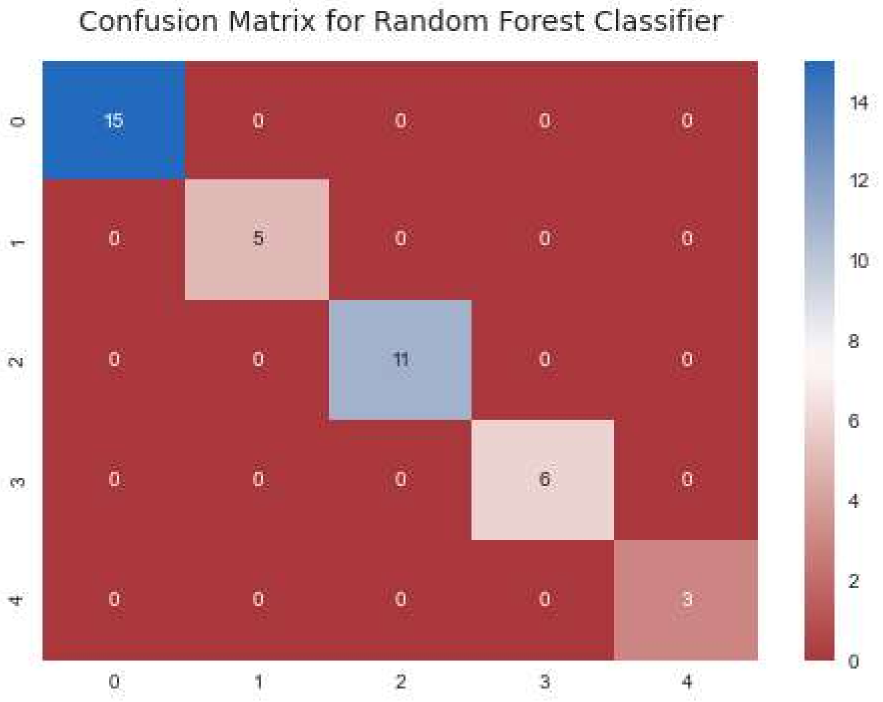
In [35]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

Out[35]:
```
RandomForestClassifier()
```

In [36]:
```
RandomForestClassifierScore = rfc.score(X_test,y_test)
print("Accacy obtained by Random Forest Classifier :", RandomForestClassifierScore*100)
```

```
Accacy obtained by Random Forest Classifier : 100.0
```

In [37]:
```
# Confusion Matrix of Random Forest Classifier

y_pred_rfc = rfc.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_rfc)
sns.heatmap(cf_matrix, annot=True, cmap="vlag_r")
plt.title("Confusion Matrix for Random Forest Classifier", fontsize=14, fontname="Helvetica", y=1.03);
```



In [38]:
```
print(metrics.classification_report(y_test, y_pred_rfc))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00         6
           5       1.00      1.00      1.00         3

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

K Neighbors Classifier

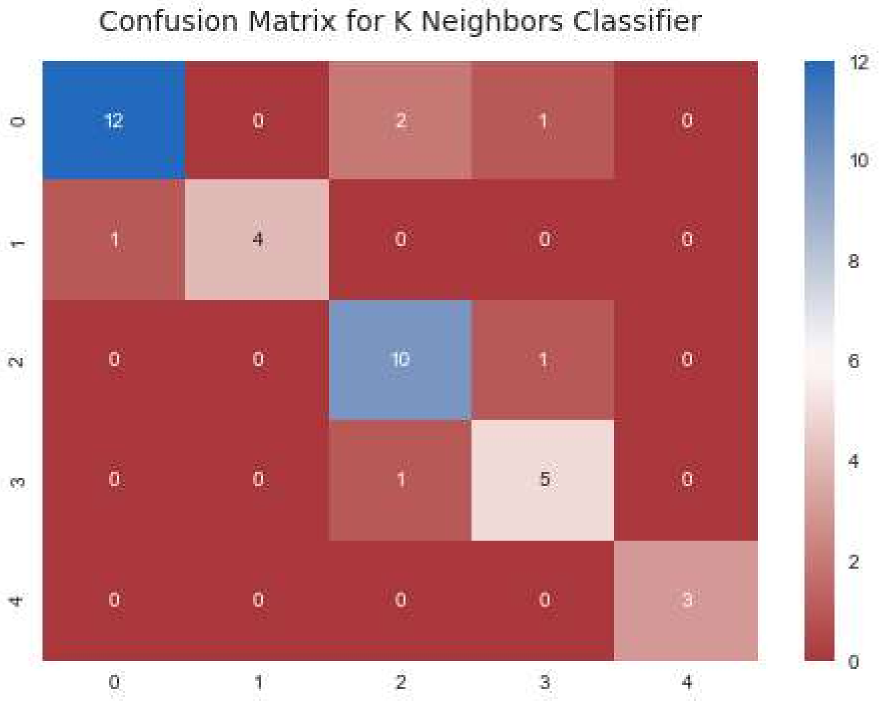
In [39]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

Out[39]:
```
KNeighborsClassifier()
```

In [40]:
```
KNeighborsClassifierScore = knn.score(X_test, y_test)
print("Accuracy obtained by K Neighbors Classifier :", KNeighborsClassifierScore*100)
```

```
Accuracy obtained by K Neighbors Classifier : 85.0
```

In [41]:
```
# Confustion Matrix

y_pred_knn = knn.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cf_matrix, annot=True, cmap="vlag_r")
plt.title("Confusion Matrix for K Neighbors Classifier", fontsize=14, fontname="Helvetica", y=1.03);
```



In [42]:
```
print(metrics.classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           1       0.92      0.80      0.86        15
           2       1.00      0.80      0.89         5
           3       0.77      0.91      0.83        11
           4       0.71      0.83      0.77         6
           5       1.00      1.00      1.00         3

    accuracy                           0.85        40
   macro avg       0.88      0.87      0.87        40
weighted avg       0.86      0.85      0.85        40
```

Decision Tree Classifier

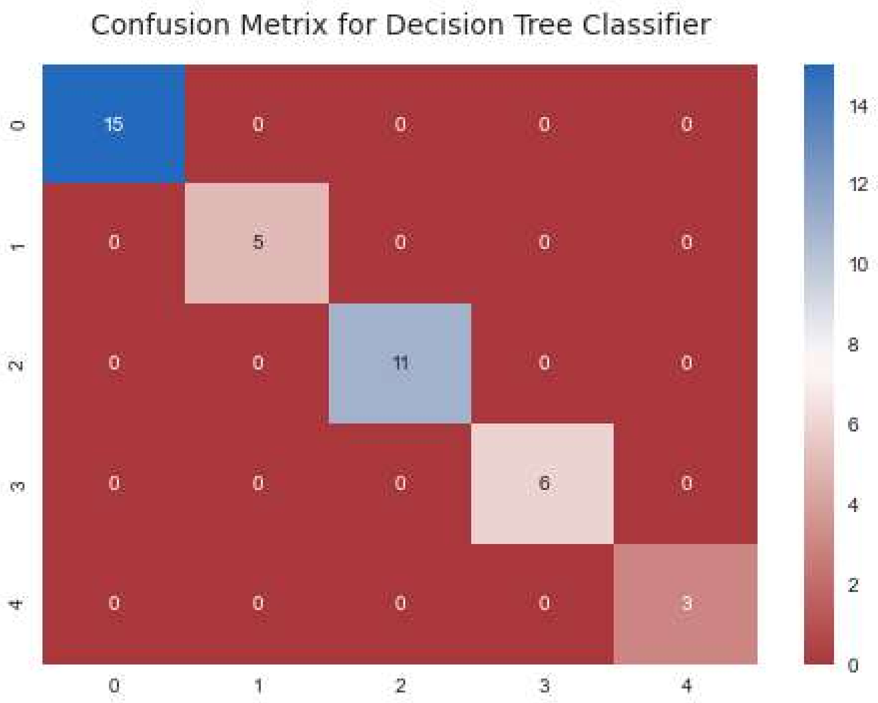
In [43]:
```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
```

Out[43]: DecisionTreeClassifier()

In [44]:
```
DecisionTreeClassifierScore = tree.score(X_test,y_test)
print("Accuracy obtained by Decision Tree Classifier :", DecisionTreeClassifierScore*100)
```

```
Accuracy obtained by Decision Tree Classifier : 100.0
```

In [45]:
```
y_pred_tree = tree.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred_tree)
sns.heatmap(cf_matrix, annot=True, cmap="vlag_r")
plt.title("Confusion Metrix for Decision Tree Classifier", fontsize=14, fontname="Helvetica", y=1.03);
```



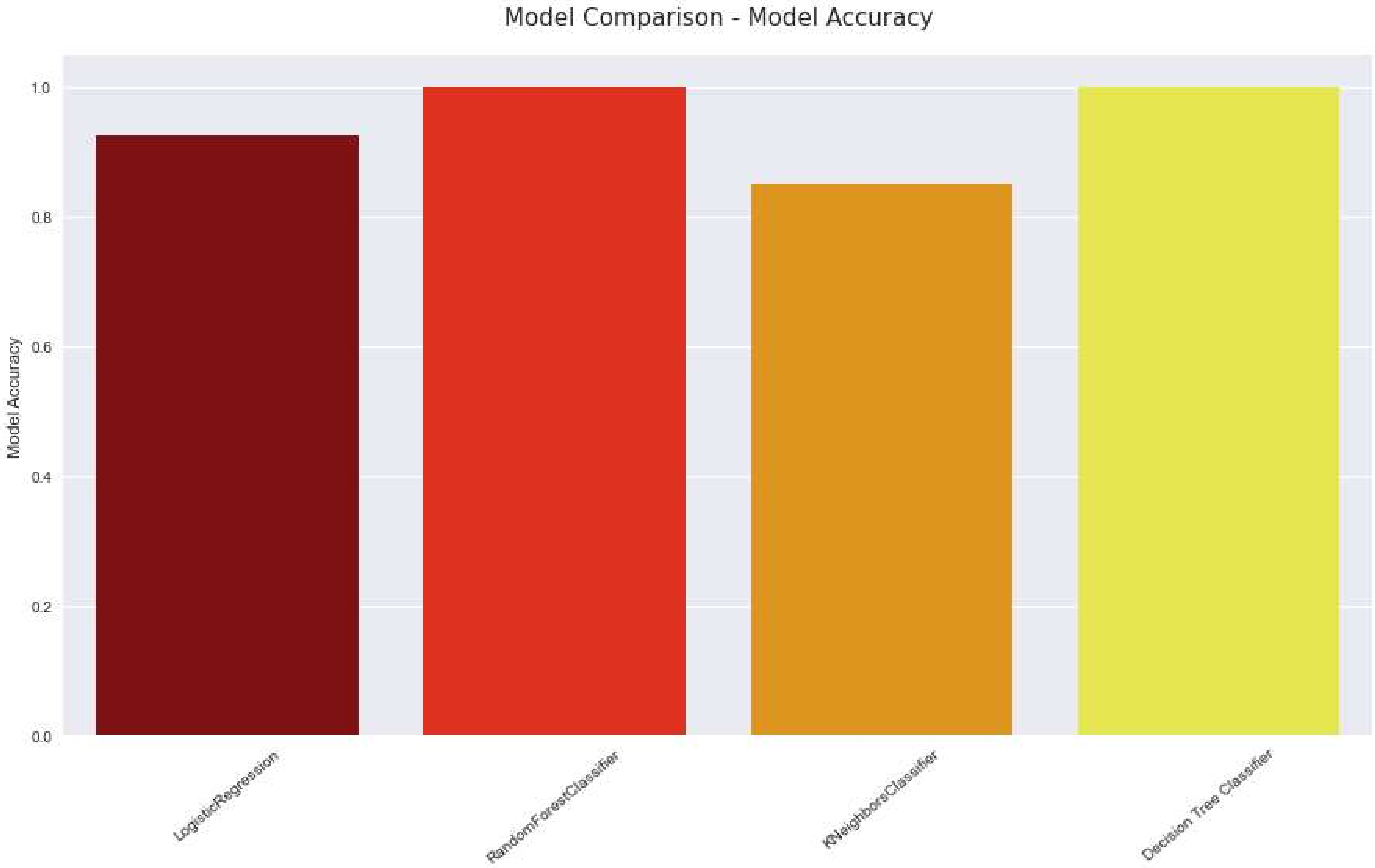
In [46]:
```
print(metrics.classification_report(y_test, y_pred_tree));
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00         6
           5       1.00      1.00      1.00         3
```

```
    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

In [48]:

```
plt.style.use("seaborn")

x = ["LogisticRegression",
     "RandomForestClassifier",
     "KNeighborsClassifier",
     "Decision Tree Classifier"]

y = [LogisticRegressionScore,
     RandomForestClassifierScore,
     KNeighborsClassifierScore,
     DecisionTreeClassifierScore]

fig, ax = plt.subplots(figsize=(15,8))
sns.barplot(x=x,y=y, palette="hot");
plt.ylabel("Model Accuracy")
plt.xticks(rotation=40)
plt.title("Model Comparison - Model Accuracy", fontsize=15, fontname="Helvetica", y=1.03);
```


Model Comparison - Model Accuracy
Model Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
LogisticRegression
RandomForestClassifier
KNeighborsClassifier
Decision Tree Classifier

In [ ]:

In [ ]:

In [ ]: