ML LAB 1

Create a generic segregation of any business scenario data into training and testingpart with 70-30% proportions and analyze missing values. Further statistically summarize the data also.

In [1]:
```
import pandas as pd
import numpy as np
df=pd.read_csv('E:/DS/Datasets/daily-bike-share.csv')
```

In [2]:
```
df.columns
```

Out[2]:
```
Index(['day', 'mnth', 'year', 'season', 'holiday', 'weekday', 'workingday',
       'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'rentals'],
      dtype='object')
```

In [3]:
```
df.shape
```

Out[3]:
```
(731, 13)
```

In [4]:
```
df.describe()
```

Out[4]:

	day	mnth	year	season	holiday	weekday	workingday	wea
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.(
mean	15.738714	6.519836	2011.500684	2.496580	0.028728	2.997264	0.683995	1.3
std	8.809949	3.451913	0.500342	1.110807	0.167155	2.004787	0.465233	0.5
min	1.000000	1.000000	2011.000000	1.000000	0.000000	0.000000	0.000000	1.(
25%	8.000000	4.000000	2011.000000	2.000000	0.000000	1.000000	0.000000	1.(
50%	16.000000	7.000000	2012.000000	3.000000	0.000000	3.000000	1.000000	1.(
75%	23.000000	10.000000	2012.000000	3.000000	0.000000	5.000000	1.000000	2.(
max	31.000000	12.000000	2012.000000	4.000000	1.000000	6.000000	1.000000	3.(

In [5]:
```
df.isna().sum()
```

Out[5]:
```
day           0
mnth          0
year          0
season        0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
rentals       0
dtype: int64
```

In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
training,testing=train_test_split(df,test_size=0.30,random_state=24)
```

In [10]:
```
training.shape
```

Out[10]:
```
(511, 13)
```

In [11]:
```
testing.shape
```

Out[11]:
```
(220, 13)
```

In [13]:
```
training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 511 entries, 338 to 418
Data columns (total 13 columns):
day           511 non-null int64
mnth          511 non-null int64
year          511 non-null int64
season        511 non-null int64
holiday       511 non-null int64
weekday       511 non-null int64
workingday    511 non-null int64
weathersit    511 non-null int64
temp          511 non-null float64
atemp         511 non-null float64
hum           511 non-null float64
windspeed     511 non-null float64
rentals       511 non-null int64
dtypes: float64(4), int64(9)
memory usage: 55.9 KB
```

In [14]:
```
testing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 220 entries, 307 to 278
Data columns (total 13 columns):
day           220 non-null int64
mnth          220 non-null int64
year          220 non-null int64
season        220 non-null int64
holiday       220 non-null int64
weekday       220 non-null int64
workingday    220 non-null int64
weathersit    220 non-null int64
temp          220 non-null float64
atemp         220 non-null float64
hum           220 non-null float64
windspeed     220 non-null float64
rentals       220 non-null int64
dtypes: float64(4), int64(9)
memory usage: 24.1 KB
```

Interpretation:

The daily bike share data has been statistically described & split into 70%-30% proportion

In [ ]:

ML LAB 2

Explore and implement Linear regression algorithm in a given business scenario and comment on its efficiency and performance.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
%matplotlib inline
```

In [5]:
```
df=pd.read_csv("E:\DS\Datasets\winequalityN.csv")
```

In [7]:
```
df.head(20)
```

Out[7]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	white	7.0	0.27	0.36	20.70	0.045	45.0	170.0	1.0010	3.00	0.45	
1	white	6.3	0.30	0.34	1.60	0.049	14.0	132.0	0.9940	3.30	0.49	
2	white	8.1	0.28	0.40	6.90	0.050	30.0	97.0	0.9951	3.26	0.44	
3	white	7.2	0.23	0.32	8.50	0.058	47.0	186.0	0.9956	3.19	0.40	
4	white	7.2	0.23	0.32	8.50	0.058	47.0	186.0	0.9956	3.19	0.40	
5	white	8.1	0.28	0.40	6.90	0.050	30.0	97.0	0.9951	3.26	0.44	
6	white	6.2	0.32	0.16	7.00	0.045	30.0	136.0	0.9949	3.18	0.47	
7	white	7.0	0.27	0.36	20.70	0.045	45.0	170.0	1.0010	3.00	0.45	
8	white	6.3	0.30	0.34	1.60	0.049	14.0	132.0	0.9940	3.30	0.49	
9	white	8.1	0.22	0.43	1.50	0.044	28.0	129.0	0.9938	3.22	0.45	
10	white	8.1	0.27	0.41	1.45	0.033	11.0	63.0	0.9908	2.99	0.56	
11	white	8.6	0.23	0.40	4.20	0.035	17.0	109.0	0.9947	3.14	0.53	
12	white	7.9	0.18	0.37	1.20	0.040	16.0	75.0	0.9920	3.18	0.63	
13	white	6.6	0.16	0.40	1.50	0.044	48.0	143.0	0.9912	3.54	0.52	
14	white	8.3	0.42	0.62	19.25	0.040	41.0	172.0	1.0002	2.98	0.67	
15	white	6.6	0.17	0.38	1.50	0.032	28.0	112.0	0.9914	3.25	0.55	
16	white	6.3	0.48	0.04	1.10	0.046	30.0	99.0	0.9928	3.24	0.36	
17	white	NaN	0.66	0.48	1.20	0.029	29.0	75.0	0.9892	3.33	0.39	
18	white	7.4	0.34	0.42	1.10	0.033	17.0	171.0	0.9917	3.12	0.53	
19	white	6.5	0.31	0.14	7.50	0.044	34.0	133.0	0.9955	3.22	0.50	

In [12]:
```
df.columns
```

Out[12]:
```
Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

In [13]:
```
df.shape
```

Out[13]:
```
(6497, 13)
```

In [14]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
type                    6497 non-null object
fixed acidity           6487 non-null float64
volatile acidity        6489 non-null float64
citric acid             6494 non-null float64
residual sugar          6495 non-null float64
chlorides               6495 non-null float64
free sulfur dioxide     6497 non-null float64
total sulfur dioxide    6497 non-null float64
density                 6497 non-null float64
pH                      6488 non-null float64
sulphates               6493 non-null float64
alcohol                 6497 non-null float64
quality                 6497 non-null int64
dtypes: float64(11), int64(1), object(1)
memory usage: 659.9+ KB
None
```

In [15]:
```
df.isna().sum()
```

Out[15]:
```
type                     0
fixed acidity           10
volatile acidity         8
citric acid              3
residual sugar           2
chlorides                2
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
pH                       9
sulphates                4
alcohol                  0
quality                  0
dtype: int64
```

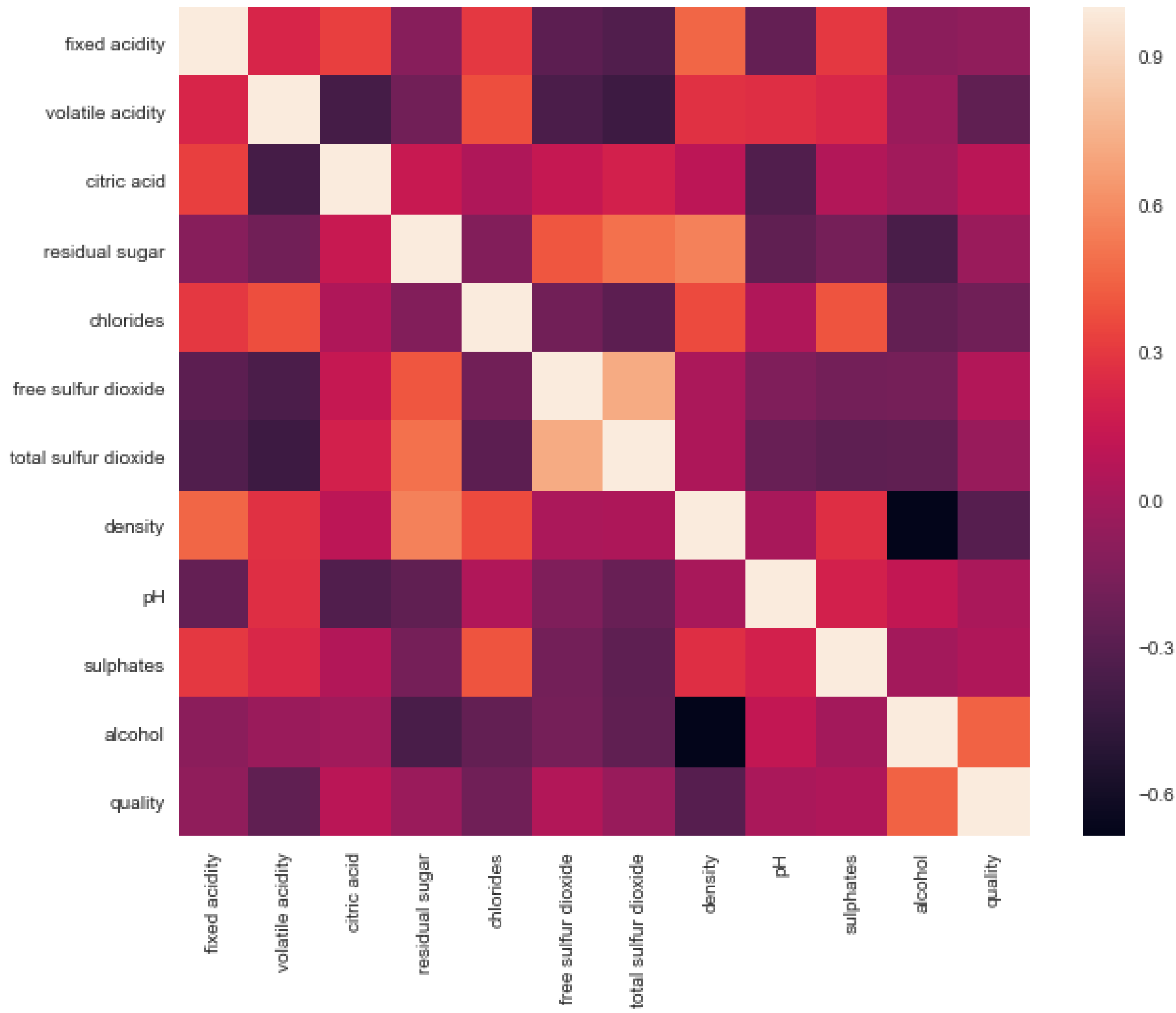
In [16]:
```
df=df.fillna(df.mean())
```

In [17]:
```
df.describe()
```

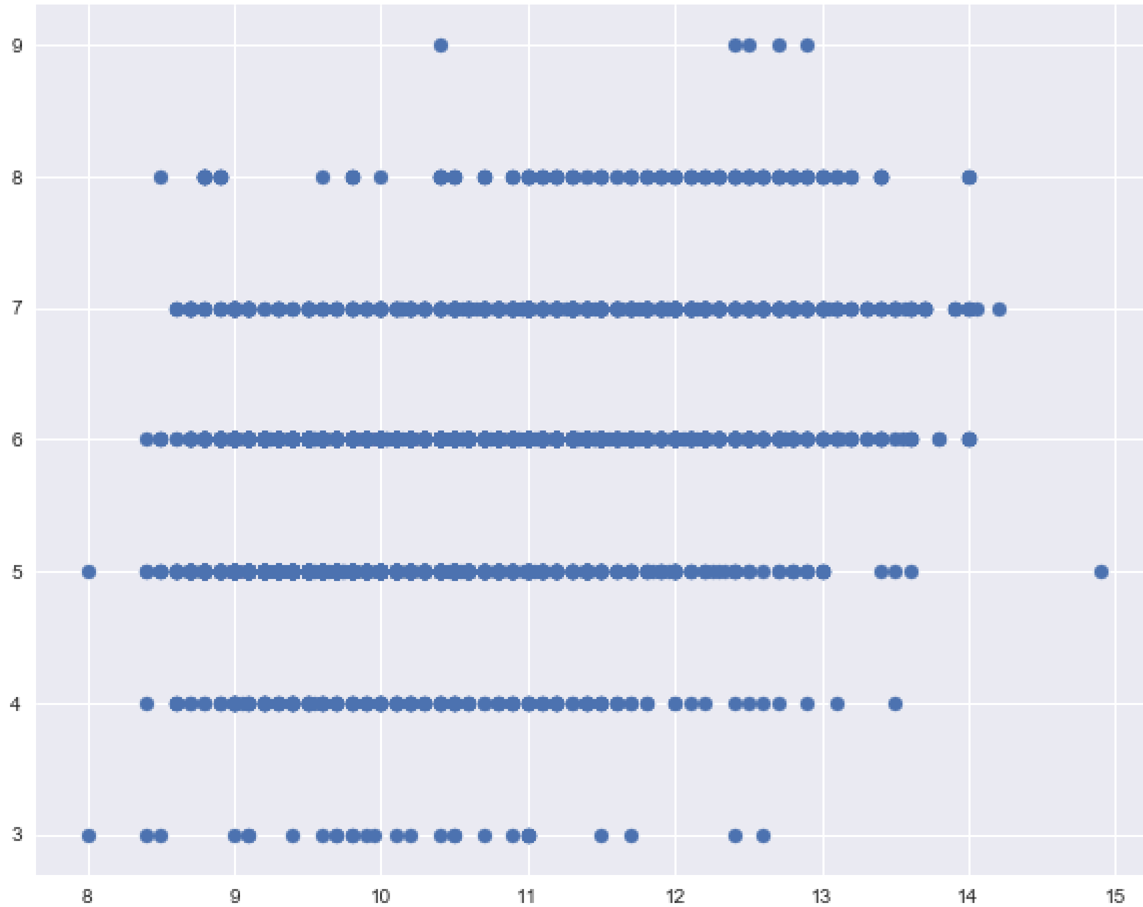
Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000
mean	7.216579	0.339691	0.318722	5.444326	0.056042	30.525319	115.744574
std	1.295751	0.164548	0.145231	4.757392	0.035031	17.749400	56.521855
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000

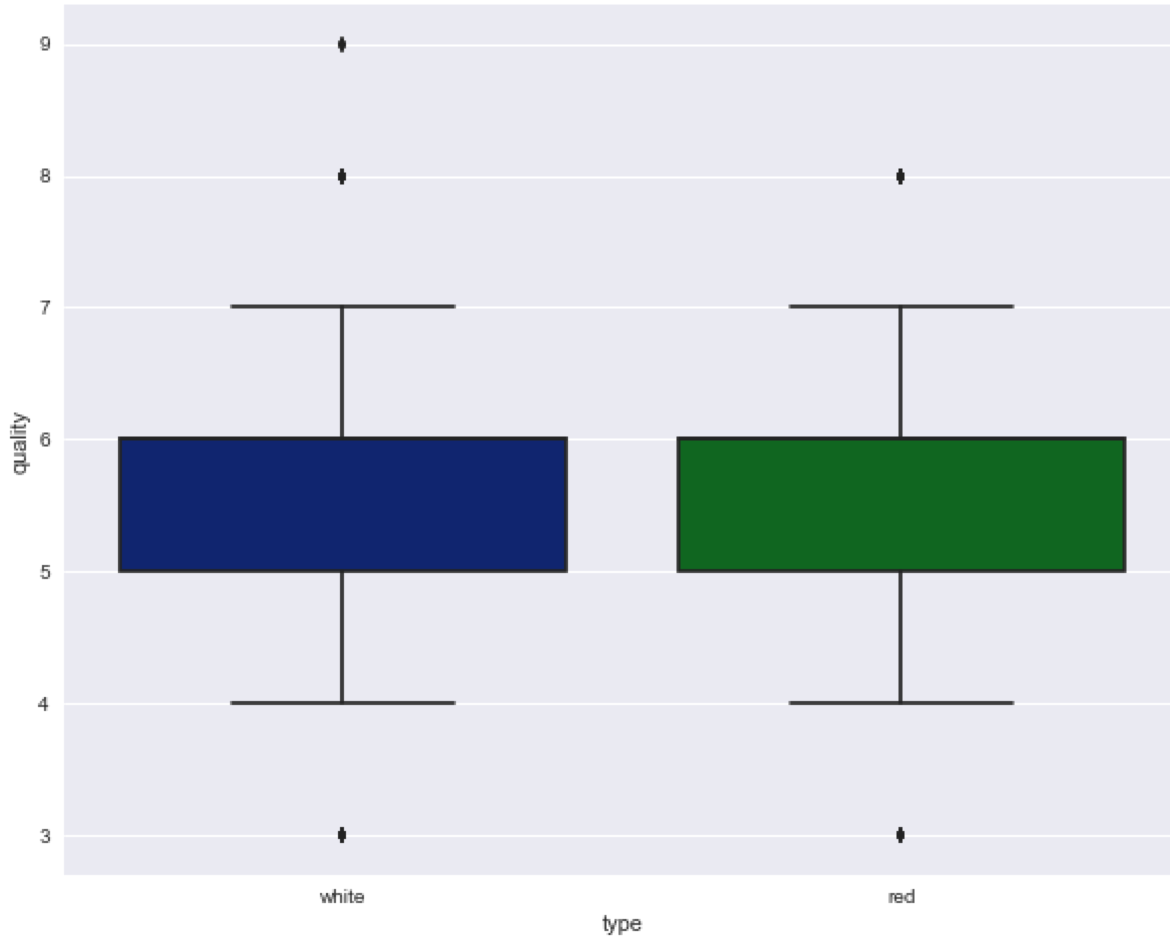
In [18]:
```
import seaborn as sns
sns.set(rc={'figure.figsize':(10,8)})
corr = df.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
plt.show()
```



In [73]:
```
plt.scatter("alcohol","quality",data=df)
plt.show()
```



In [19]:
```
sns.boxplot(x="type",y="quality",data=df, palette="dark")
plt.show()
```



In [20]:
```
df=df[df.columns.drop('type')]
```

In [21]:
```
df.head(5)
```

Out[21]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9

In [22]:
```
print(df.nunique())
```

```
fixed acidity            107
volatile acidity         188
citric acid               90
residual sugar           317
chlorides                215
free sulfur dioxide      135
total sulfur dioxide     276
density                  998
pH                       109
sulphates                112
alcohol                  111
quality                    7
dtype: int64
```

In [23]:
```
from sklearn.model_selection import train_test_split
training, testing =train_test_split(df, test_size= 0.30, random_state=24)
```

In [24]:
```
training.shape
```

Out[24]: (4547, 12)

In [25]:
```
testing.shape
```

Out[25]: (1950, 12)

In [28]:
```
X = training['alcohol']
```

In [29]:
```
X.shape
```

Out[29]: (4547,)

In [30]:
```
x= np.array(X)
```

In [31]:
```
x = x.reshape(4547,1)
```

In [32]:
```
x.shape
```
Out[32]:
```
(4547, 1)
```
In [33]:
```
Y = training['quality']
```
In [34]:
```
Y.shape
```
Out[34]:
```
(4547,)
```
In [35]:
```
Y= np.array(Y)
```
In [36]:
```
y = Y.reshape(4547,1)
```
In [37]:
```
y.shape
```
Out[37]:
```
(4547, 1)
```
In [38]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
model=lr.fit(x, y)
```
In [39]:
```
print(model)
```
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```
In [50]:
```
print(model.coef_[0][0]) ## Printing the coefficients
print(model.intercept_[0]) ### printing the Intercept term

print("The linear model is: Y = {:.5} + {:.5}X".format(model.intercept_[0], model
```
```
0.32546629798314014
2.4029999180573034
The linear model is: Y = 2.403 + 0.32547X
```
In [52]:
```
 X_test=testing['alcohol']
```
In [53]:
```
X_test.shape
```
Out[53]:
```
(1950,)
```
In [82]:
```
X_test = X_test.reshape(1950,1)
```
In [83]:
```
X_test.shape
```
Out[83]:
```
(1950, 1)
```
In [84]:
```
Y_test=testing['quality']
```
In [85]:
```
Y_test.shape
```
Out[85]:
```
(1950,)
```

In [86]:
```
Y_test = Y_test.reshape(1950,1)
```

```
C:\Users\PRANAV\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarn
ing: reshape is deprecated and will raise in a subsequent release. Please use .
values.reshape(...) instead
```

In [88]:
```
Y_test.shape
```

Out[88]:
```
(1950, 1)
```

In [94]:
```
Y_test
```

Out[94]:
```
array([[5],
       [4],
       [5],
       ...,
       [5],
       [7],
       [5]], dtype=int64)
```

In [89]:
```
Y_pred = lr.predict(X_test)
```

In [90]:
```
Y_pred
```

Out[90]:
```
array([[5.72275616],
       [5.52747638],
       [5.3321966 ],
       ...,
       [5.56002301],
       [5.75530279],
       [5.46238312]])
```

In [91]:
```
 from sklearn.metrics import mean_squared_error
```

In [92]:
```
LR_score= mean_squared_error(Y_test,Y_pred)
```

In [93]:
```
LR_score
```

Out[93]:
```
0.610874859296884
```

Interpretation:

The wine quality has been predicted using Linear Regression, with LR score of 61%

In [ ]:
```

```

ML LAB 3

Explore and implement logistic regression algorithm in a given business scenario and comment on its efficiency and performance.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from warnings import filterwarnings
filterwarnings('ignore')
```

In [3]:
```
data = pd.read_csv('E:\DS\Datasets\drug200.csv')
```

In [4]:
```
data.head()
```

Out[4]:

	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

In [5]:
```
data.isnull().sum()
```

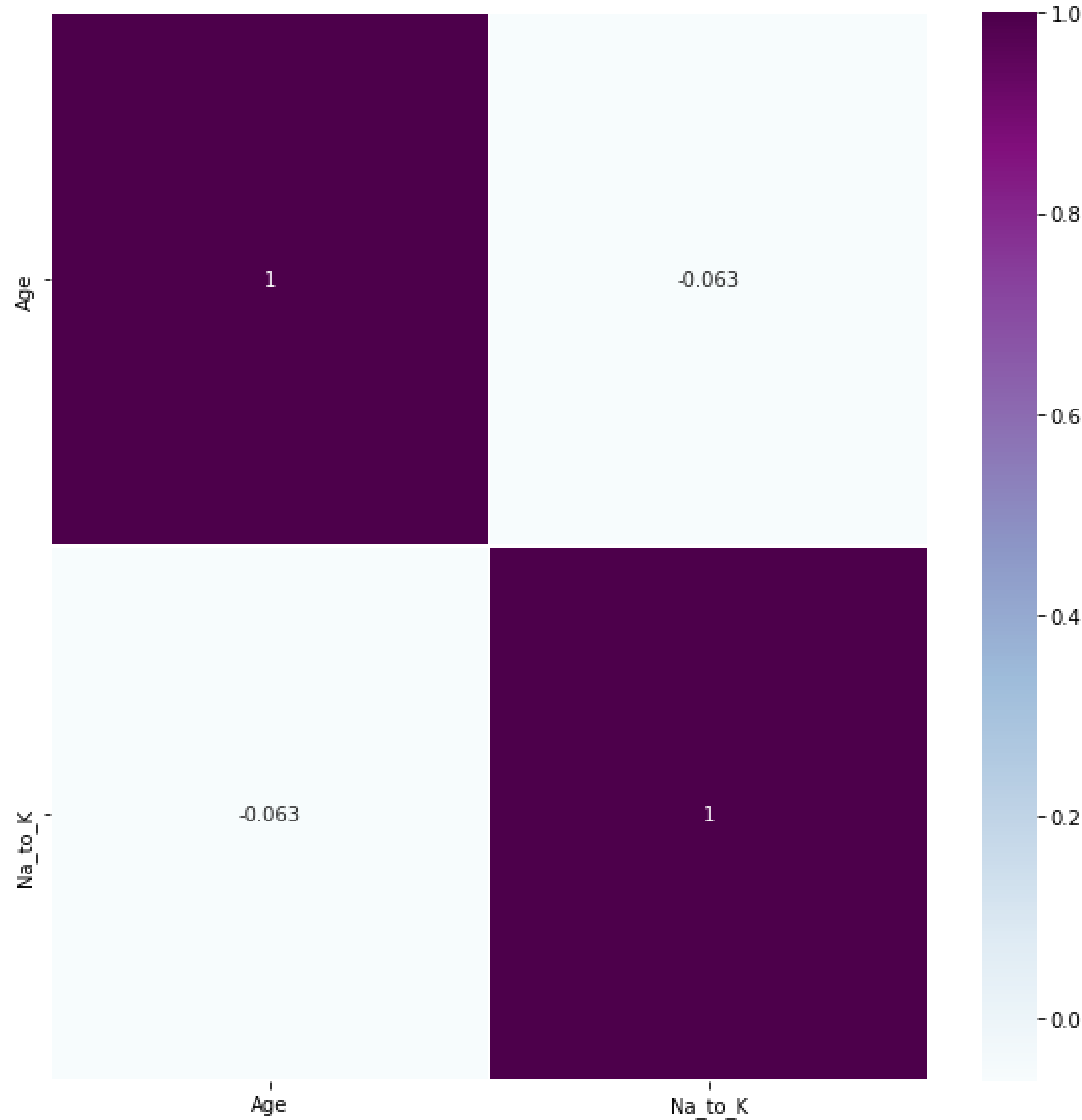
Out[5]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

In [6]:
```
data.info()
```

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

there is no missing values in the date we have 6 coulmns and 200 rows

In [7]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```

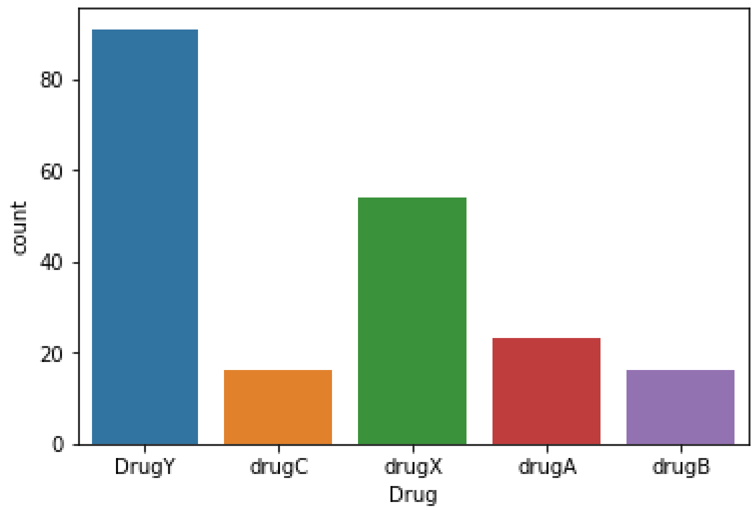


In [8]:
```
data['Drug'].value_counts()
```

Out[8]:
```
DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: Drug, dtype: int64
```

In [9]:
```
sns.countplot(x = 'Drug', data= data)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee10d437f0>

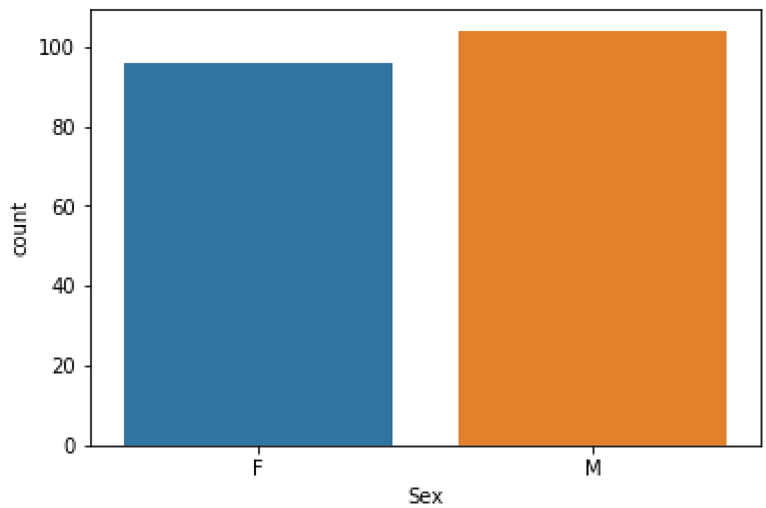


In [10]:
```
data['Sex'].value_counts()
```

Out[10]:
```
M    104
F     96
Name: Sex, dtype: int64
```

In [11]:
```
sns.countplot(x = 'Sex', data= data)
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee10ebe400>
```

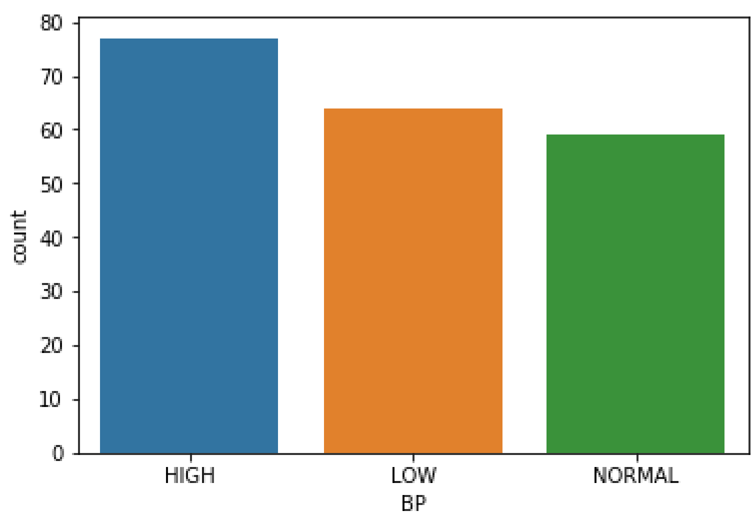


In [12]:
```
data['BP'].value_counts()
```

Out[12]:
```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

In [13]:
```
sns.countplot(x = 'BP', data= data)
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ee10ecfa58>
```

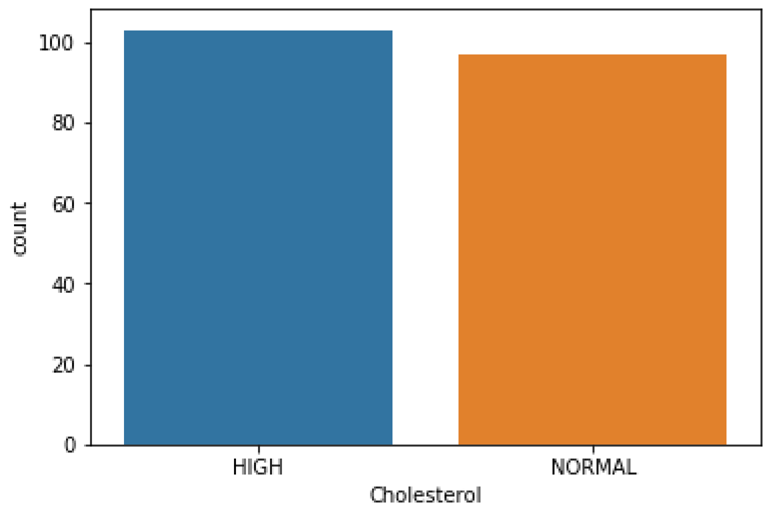


In [14]:
```
data['Cholesterol'].value_counts()
```

Out[14]:
```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

In [15]:
```
sns.countplot(x = 'Cholesterol', data= data)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee10ebe1d0>



In [16]:
```
data['Na_to_K'].describe()
```

Out[16]:
```
count    200.000000
mean      16.084485
std        7.223956
min        6.269000
25%       10.445500
50%       13.936500
75%       19.380000
max       38.247000
Name: Na_to_K, dtype: float64
```

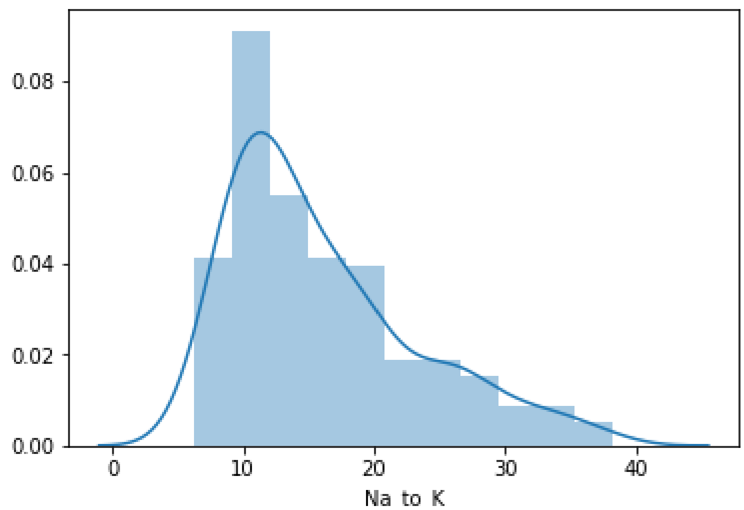
In [21]:
```
!pip install seaborn --upgrade
```

```
Requirement already satisfied: seaborn in c:\users\pranav\anaconda3\lib\site-pa
ckages (0.8.1)
Collecting seaborn
  Using cached seaborn-0.11.2-py3-none-any.whl (292 kB)
Collecting matplotlib>=2.2
  Using cached matplotlib-3.3.4-cp36-cp36m-win_amd64.whl (8.5 MB)
Requirement already satisfied: numpy>=1.15 in c:\users\pranav\anaconda3\lib\sit
e-packages (from seaborn) (1.19.5)
Requirement already satisfied: scipy>=1.0 in c:\users\pranav\anaconda3\lib\site
-packages (from seaborn) (1.5.4)
Collecting pandas>=0.23
  Using cached pandas-1.1.5-cp36-cp36m-win_amd64.whl (8.7 MB)
Requirement already satisfied: cycler>=0.10 in c:\users\pranav\anaconda3\lib\si
te-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\pranav\anaconda
3\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\u
sers\pranav\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\pranav\anaconda3\lib\s
ite-packages (from matplotlib>=2.2->seaborn) (8.3.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\pranav\anaconda3\l
ib\site-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: six in c:\users\pranav\anaconda3\lib\site-packag
es (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.11.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\pranav\anaconda3\lib\si
te-packages (from pandas>=0.23->seaborn) (2017.3)
Installing collected packages: pandas, matplotlib, seaborn
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 2.1.2
    Uninstalling matplotlib-2.1.2:


WARNING: Value for scheme.headers does not match. Please report this to <http
s://github.com/pypa/pip/issues/9617>
distutils: c:\users\pranav\anaconda3\Include\UNKNOWN
sysconfig: c:\users\pranav\anaconda3\Include
WARNING: Additional context:
user = False
home = None
root = None
prefix = None
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
```

```
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
    WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anacond
a3\lib\site-packages)
    WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\li
b\site-packages)
    WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\li
b\site-packages)
ERROR: Could not install packages due to an OSError: [WinError 5] Access is den
ied: 'c:\\users\\pranav\\anaconda3\\lib\\site-packages\\matplotlib\\backends\\_
backend_agg.cp36-win_amd64.pyd'
Consider using the `--user` option or check the permissions.

WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: You are using pip version 21.1; however, version 21.3.1 is available.
You should consider upgrading via the 'c:\users\pranav\anaconda3\python.exe -m
pip install --upgrade pip' command.
```

In [23]:
```
sns.distplot(data['Na_to_K'])
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee11365d68>



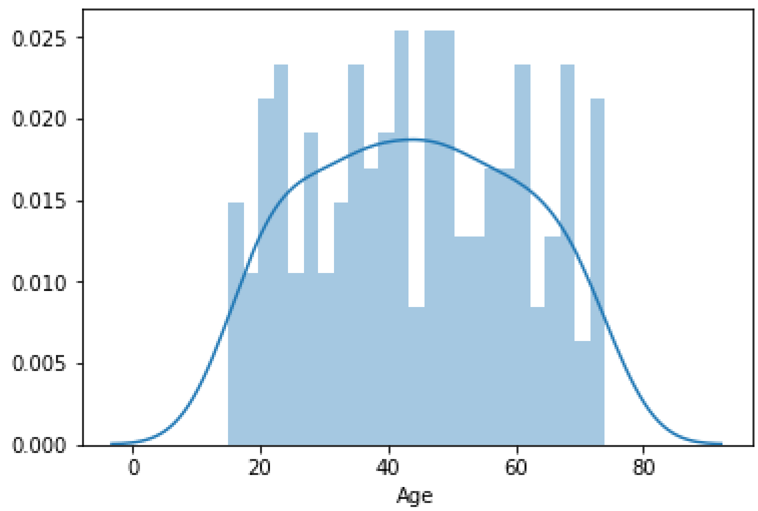
In [25]:
```
!pip install -U seaborn
```

```
Requirement already satisfied: seaborn in c:\users\pranav\anaconda3\lib\site-pa
ckages (0.8.1)
Collecting seaborn
  Using cached seaborn-0.11.2-py3-none-any.whl (292 kB)
Requirement already satisfied: numpy>=1.15 in c:\users\pranav\anaconda3\lib\sit
e-packages (from seaborn) (1.19.5)
Collecting matplotlib>=2.2
  Using cached matplotlib-3.3.4-cp36-cp36m-win_amd64.whl (8.5 MB)
Requirement already satisfied: scipy>=1.0 in c:\users\pranav\anaconda3\lib\site
-packages (from seaborn) (1.5.4)
Requirement already satisfied: pandas>=0.23 in c:\users\pranav\anaconda3\lib\si
te-packages (from seaborn) (1.1.5)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\u
sers\pranav\anaconda3\lib\site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: cycler>=0.10 in c:\users\pranav\anaconda3\lib\si
te-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\pranav\anaconda3\l
ib\site-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\pranav\anaconda
3\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\pranav\anaconda3\lib\s
ite-packages (from matplotlib>=2.2->seaborn) (8.3.1)
Requirement already satisfied: six in c:\users\pranav\anaconda3\lib\site-packag
es (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.11.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\pranav\anaconda3\lib\si
te-packages (from pandas>=0.23->seaborn) (2017.3)
Installing collected packages: matplotlib, seaborn
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 2.1.2
    Uninstalling matplotlib-2.1.2:

WARNING: Value for scheme.headers does not match. Please report this to <http
s://github.com/pypa/pip/issues/9617>
distutils: c:\users\pranav\anaconda3\Include\UNKNOWN
sysconfig: c:\users\pranav\anaconda3\Include
WARNING: Additional context:
user = False
home = None
root = None
prefix = None
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
```

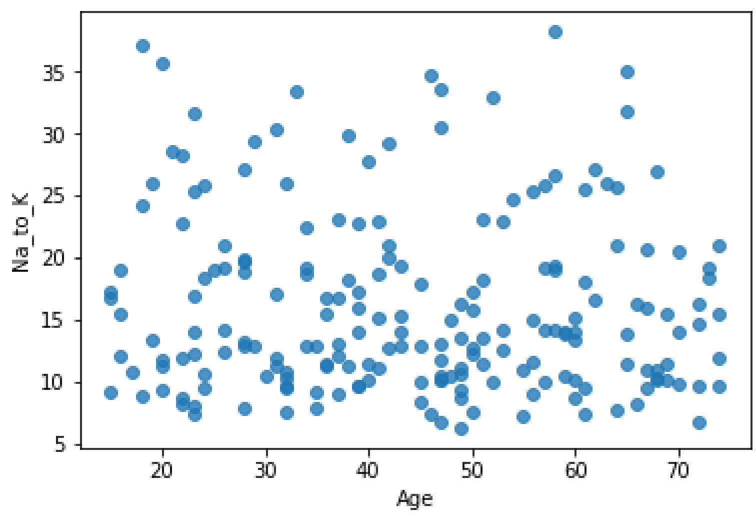
```
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
    WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anacond
a3\lib\site-packages)
    WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\li
b\site-packages)
    WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\li
b\site-packages)
ERROR: Could not install packages due to an OSError: [WinError 5] Access is den
ied: 'c:\\users\\pranav\\anaconda3\\lib\\site-packages\\matplotlib\\backends\\_
backend_agg.cp36-win_amd64.pyd'
Consider using the `--user` option or check the permissions.

WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -tatsmodels (c:\users\pranav\anaconda3\l
ib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: Ignoring invalid distribution -andas (c:\users\pranav\anaconda3\lib\si
te-packages)
WARNING: You are using pip version 21.1; however, version 21.3.1 is available.
You should consider upgrading via the 'c:\users\pranav\anaconda3\python.exe -m
pip install --upgrade pip' command.
```

In [32]:
```
sns.distplot(data['Age'], hist=True,kde=True, bins = 25)
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee11455f98>



In [42]:
```
sns.regplot(x=data["Age"], y=data["Na_to_K"], fit_reg=False, scatter=True);
```

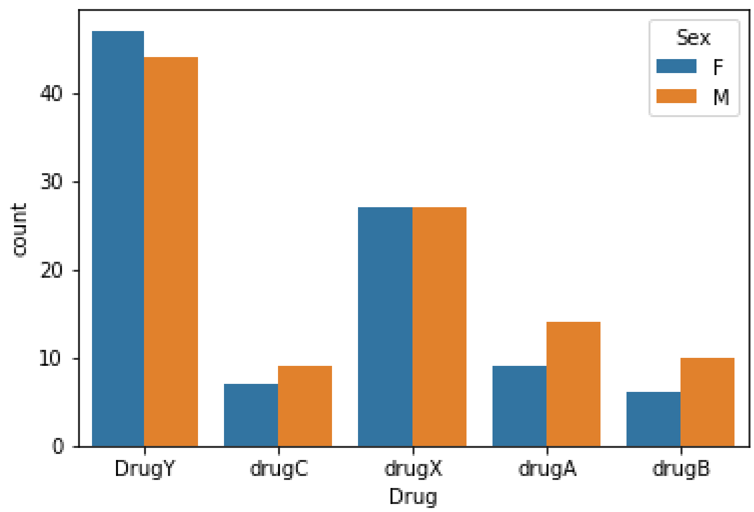


In [43]:
```
data_sex_drug = data.groupby(['Drug','Sex']).size().reset_index(name = 'count')
print(data_sex_drug)
```

```
    Drug Sex  count
0  DrugY   F     47
1  DrugY   M     44
2  drugA   F      9
3  drugA   M     14
4  drugB   F      6
5  drugB   M     10
6  drugC   F      7
7  drugC   M      9
8  drugX   F     27
9  drugX   M     27
```

In [44]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Sex')
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee123dc2b0>

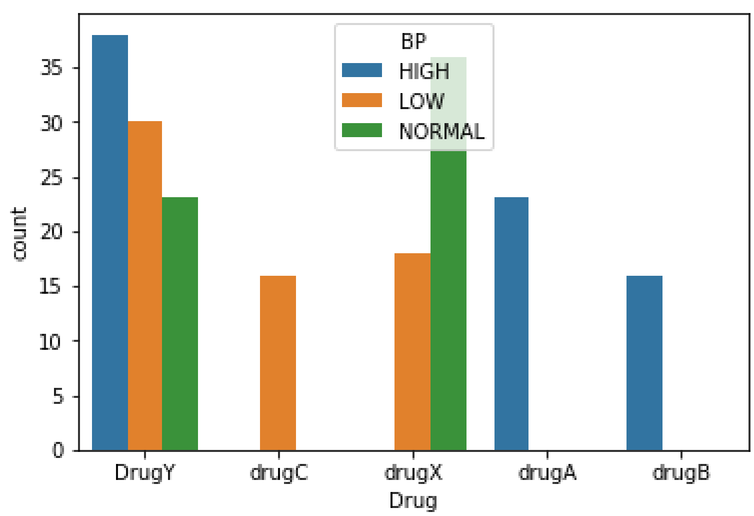


In [45]:
```
data_BP_drug = data.groupby(['Drug','BP']).size().reset_index(name = 'count')
print(data_BP_drug)
```

```
    Drug      BP  count
0  DrugY    HIGH     38
1  DrugY     LOW     30
2  DrugY  NORMAL     23
3  drugA    HIGH     23
4  drugB    HIGH     16
5  drugC     LOW     16
6  drugX     LOW     18
7  drugX  NORMAL     36
```

In [46]:
```
sns.countplot(x = 'Drug', data= data, hue = 'BP')
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee12417080>

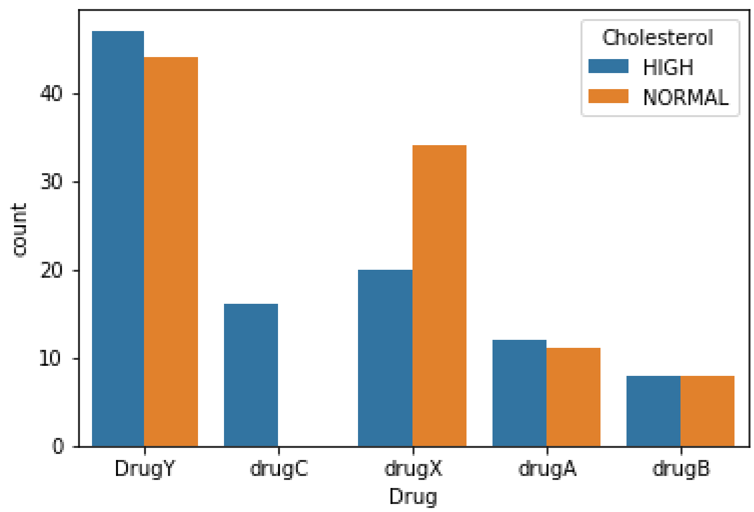


In [47]:
```
data_Cholesterol_drug = data.groupby(['Drug','Cholesterol']).size().reset_index(n
print(data_Cholesterol_drug)
```

```
    Drug Cholesterol  count
0  DrugY        HIGH     47
1  DrugY      NORMAL     44
2  drugA        HIGH     12
3  drugA      NORMAL     11
4  drugB        HIGH      8
5  drugB      NORMAL      8
6  drugC        HIGH     16
7  drugX        HIGH     20
8  drugX      NORMAL     34
```

In [48]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Cholesterol')
```

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x1ee115854e0>



In [49]:
```
data['Sex'] = data['Sex'].map({'M': 1, 'F': 0})
data['Cholesterol'] = data['Cholesterol'].map({'HIGH' : 1, 'NORMAL' : 0})
data['Drug'] = data['Drug'].map({'DrugY':1, 'drugC':2, 'drugX':3, 'drugA':4, 'dru
data.head()
```

Out[49]:

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

In [50]:
```
data.shape
```

Out[50]: (200, 6)

In [51]:
```
data = pd.get_dummies(data)
data.head()
```

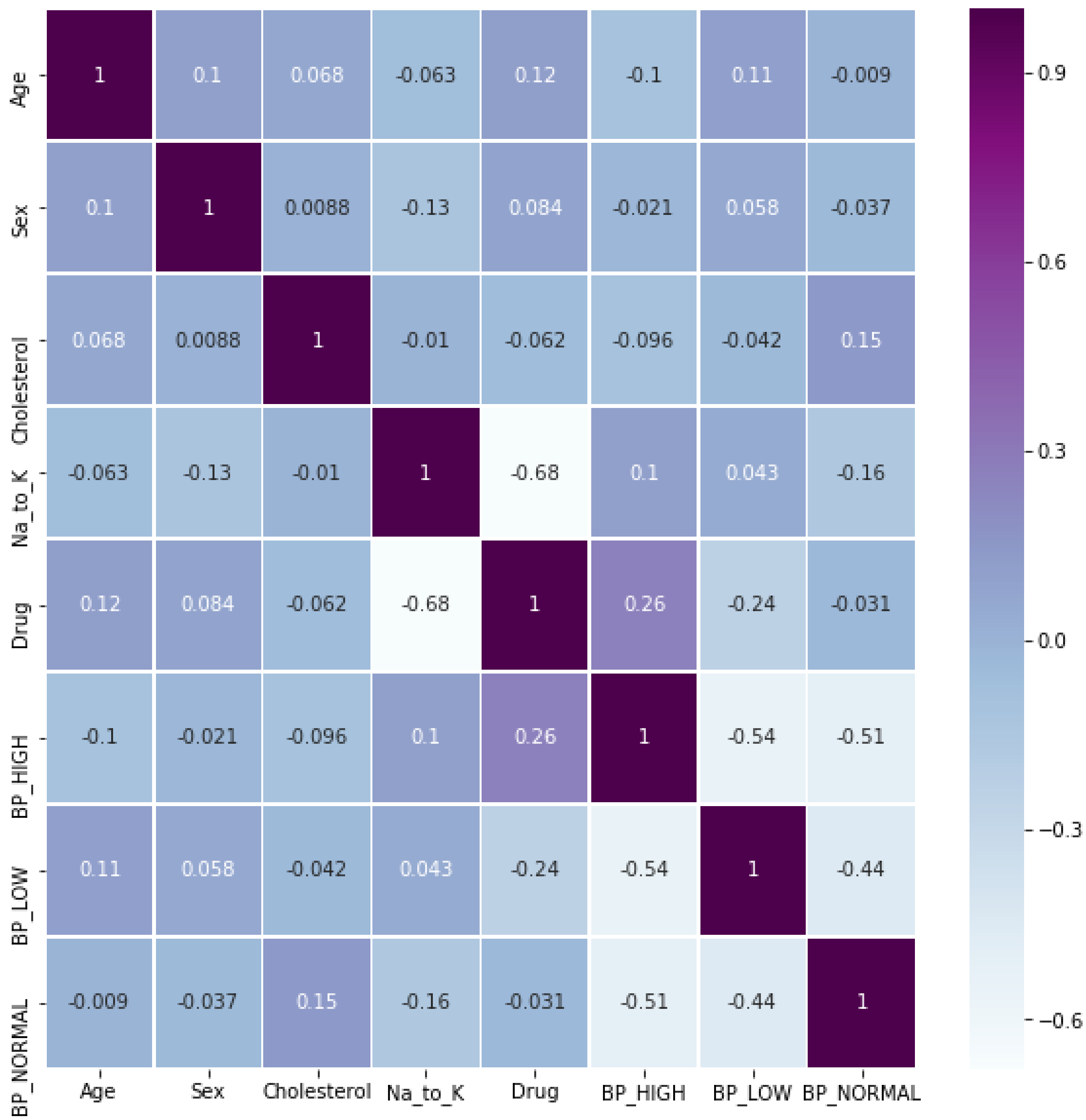
Out[51]:

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

In [52]:
```
data.shape
```

Out[52]: (200, 8)

In [53]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```



In [54]:
```
X = data.drop('Drug', axis = 1).values
y = data['Drug'].values.reshape((-1,1))
```

In [55]:
```
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_
print('x train shape {}'.format(X_train.shape))
print('x test shape  {}'.format(X_test.shape))
print('y train shape {}'.format(y_train.shape))
print('y test shape  {}'.format(y_test.shape))
```

```
x train shape (160, 7)
x test shape  (40, 7)
y train shape (160, 1)
y test shape  (40, 1)
```

In [56]:
```
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression(C = 2 ,solver = 'liblinear', tol = .001)
```

In [58]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_repo
```

In [64]:
```
logistic_model.fit(X_train, y_train)
y_pred = logistic_model.predict(X_test)
print(logistic_model.score(X_train,y_train)*100)
logistic_score = accuracy_score(y_test, y_pred)
print(logistic_score*100)
```

```
97.5
95.0
```

In [65]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[17  0  0  0  0]
 [ 0  4  0  0  0]
 [ 0  0 13  0  0]
 [ 1  0  0  2  1]
 [ 0  0  0  0  2]]
             precision    recall  f1-score   support

          1       0.94      1.00      0.97        17
          2       1.00      1.00      1.00         4
          3       1.00      1.00      1.00        13
          4       1.00      0.50      0.67         4
          5       0.67      1.00      0.80         2

avg / total       0.96      0.95      0.94        40
```

Interpretation:

The drugs have been classified using Logistic Regression, with 95% accuracy.

In [ ]:

ML LAB 4

Explore and implement Linear Regression Using Gradient Descent in a given business scenario and comment on its efficiency and performance.

In [ ]:
```
# Making the imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
from sklearn.linear_model import LinearRegression # To work on Linear Regression
from sklearn.metrics import r2_score # To Calculate Performance matrix
import statsmodels.api as sm # To calculatestats modles
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: Future
Warning: pandas.util.testing is deprecated. Use the functions in the public API
at pandas.testing instead.
  import pandas.util.testing as tm
```

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kc_house_data.csv to kc_house_data.csv
```

In [ ]:
```
import io
df = pd.read_csv(io.BytesIO(uploaded['kc_house_data.csv']))
```

In [ ]:

```
df.head(20)
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	w
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	
5	7237550310	20140512T000000	1225000.0	4	4.50	5420	101930	1.0	
6	1321400060	20140627T000000	257500.0	3	2.25	1715	6819	2.0	
7	2008000270	20150115T000000	291850.0	3	1.50	1060	9711	1.0	
8	2414600126	20150415T000000	229500.0	3	1.00	1780	7470	1.0	
9	3793500160	20150312T000000	323000.0	3	2.50	1890	6560	2.0	
10	1736800520	20150403T000000	662500.0	3	2.50	3560	9796	1.0	
11	9212900260	20140527T000000	468000.0	2	1.00	1160	6000	1.0	
12	114101516	20140528T000000	310000.0	3	1.00	1430	19901	1.5	
13	6054650070	20141007T000000	400000.0	3	1.75	1370	9680	1.0	
14	1175000570	20150312T000000	530000.0	5	2.00	1810	4850	1.5	
15	9297300055	20150124T000000	650000.0	4	3.00	2950	5000	2.0	
16	1875500060	20140731T000000	395000.0	3	2.00	1890	14040	2.0	
17	6865200140	20140529T000000	485000.0	4	1.00	1600	4300	1.5	
18	16000397	20141205T000000	189000.0	2	1.00	1200	9850	1.0	
19	7983200060	20150424T000000	230000.0	3	1.00	1250	9774	1.0	

In [ ]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
None
```

In [ ]:
```
df.isna().sum()
```

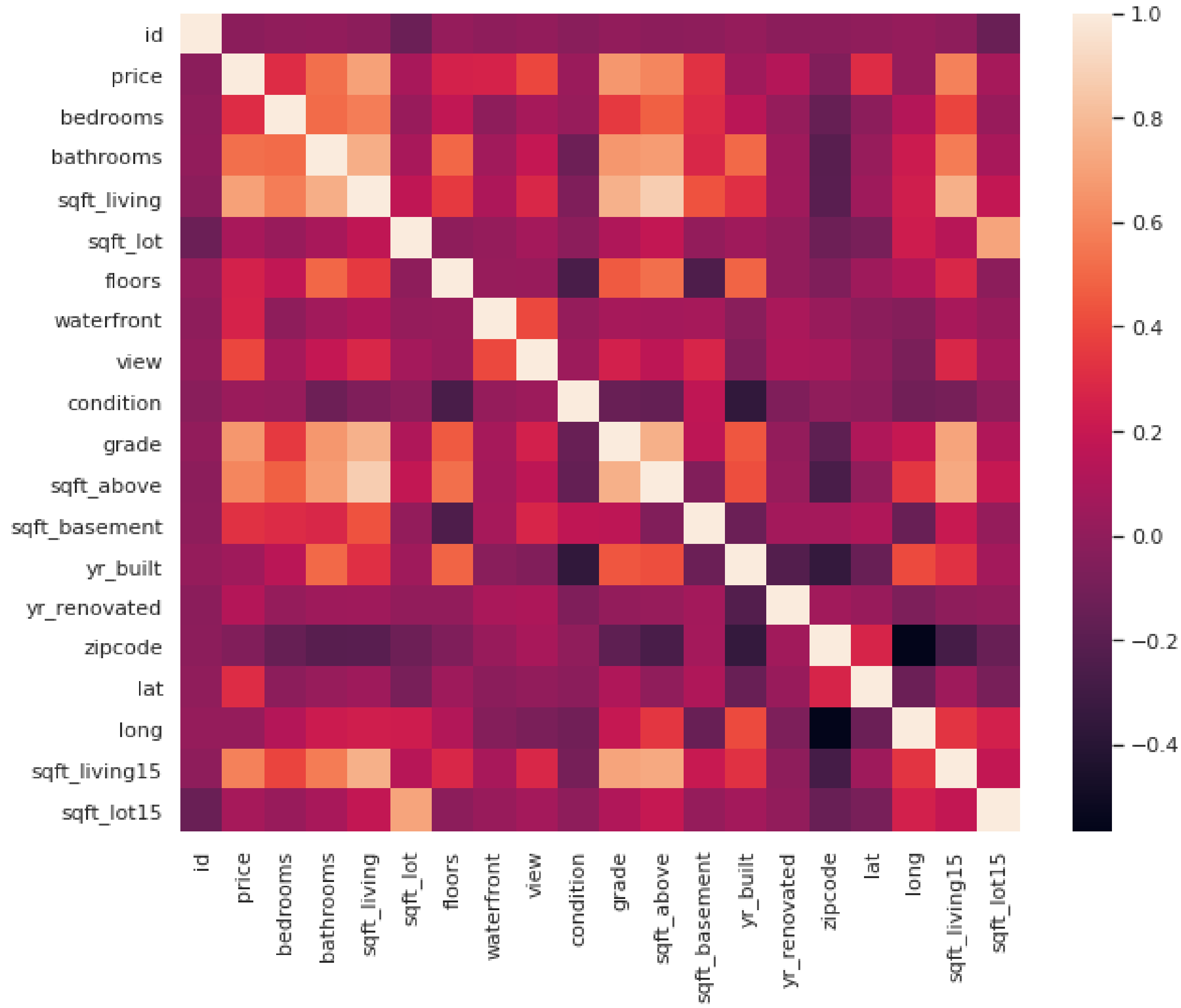
Out[8]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [ ]:
```
df.describe()
```

Out[9]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3

In [ ]:
```
import seaborn as sns
sns.set(rc={'figure.figsize':(10,8)})
corr = df.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
plt.show()
```



In [ ]:
```
data = [df["sqft_living"], df["price"]]
headers = ["sqft_living", "price"]
df1 = pd. concat(data, axis=1, keys=headers)
```

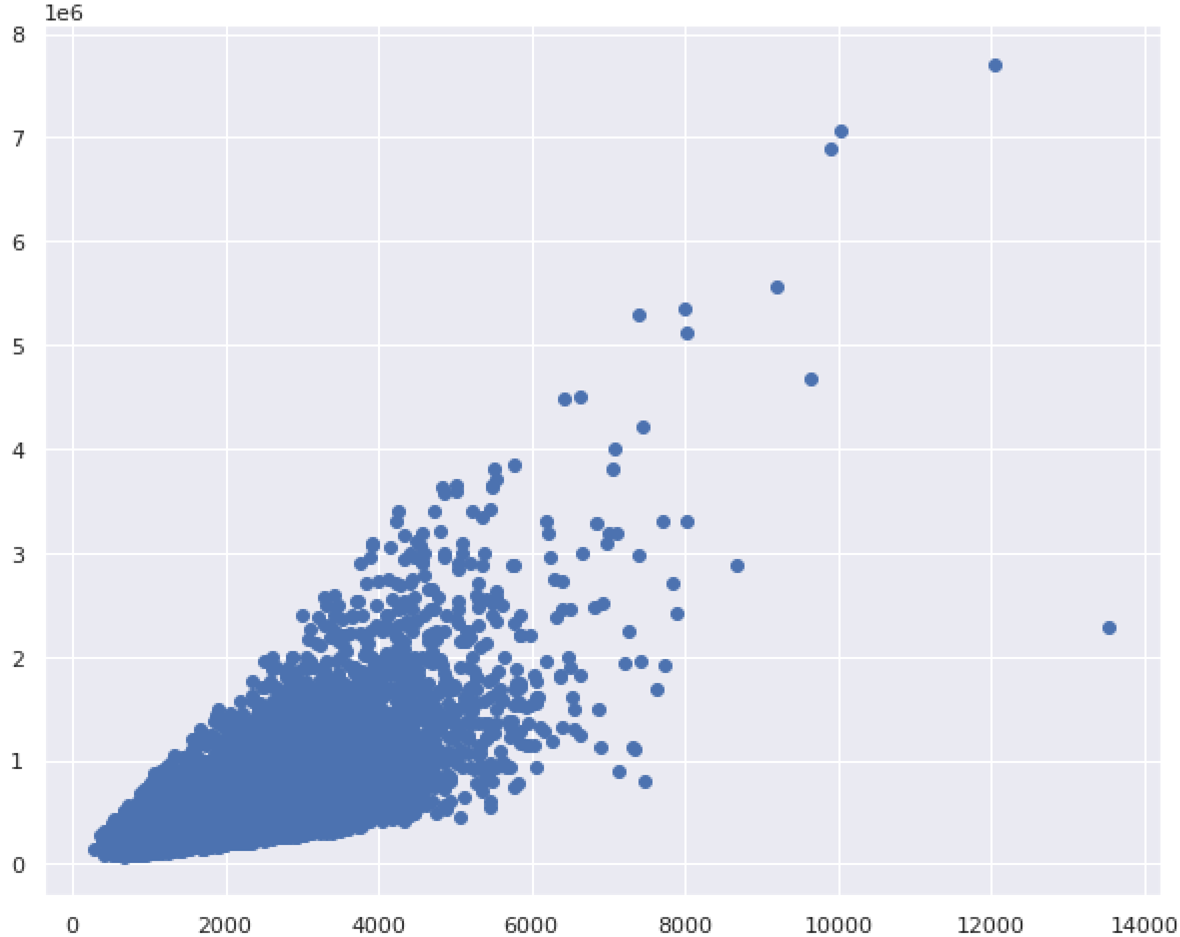
In [ ]:
```
df1
```

Out[24]:

	sqft_living	price
0	1180	221900.0
1	2570	538000.0
2	770	180000.0
3	1960	604000.0
4	1680	510000.0
...	...	...
21608	1530	360000.0
21609	2310	400000.0
21610	1020	402101.0
21611	1600	400000.0
21612	1020	325000.0

21613 rows × 2 columns

In [ ]:
```
plt.scatter("sqft_living","price",data=df1)
plt.show()
```



In [ ]:
```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
training, testing =train_test_split(df1, test_size= 0.30, random_state=24)
```

In [ ]:
```
training
```

Out[27]:

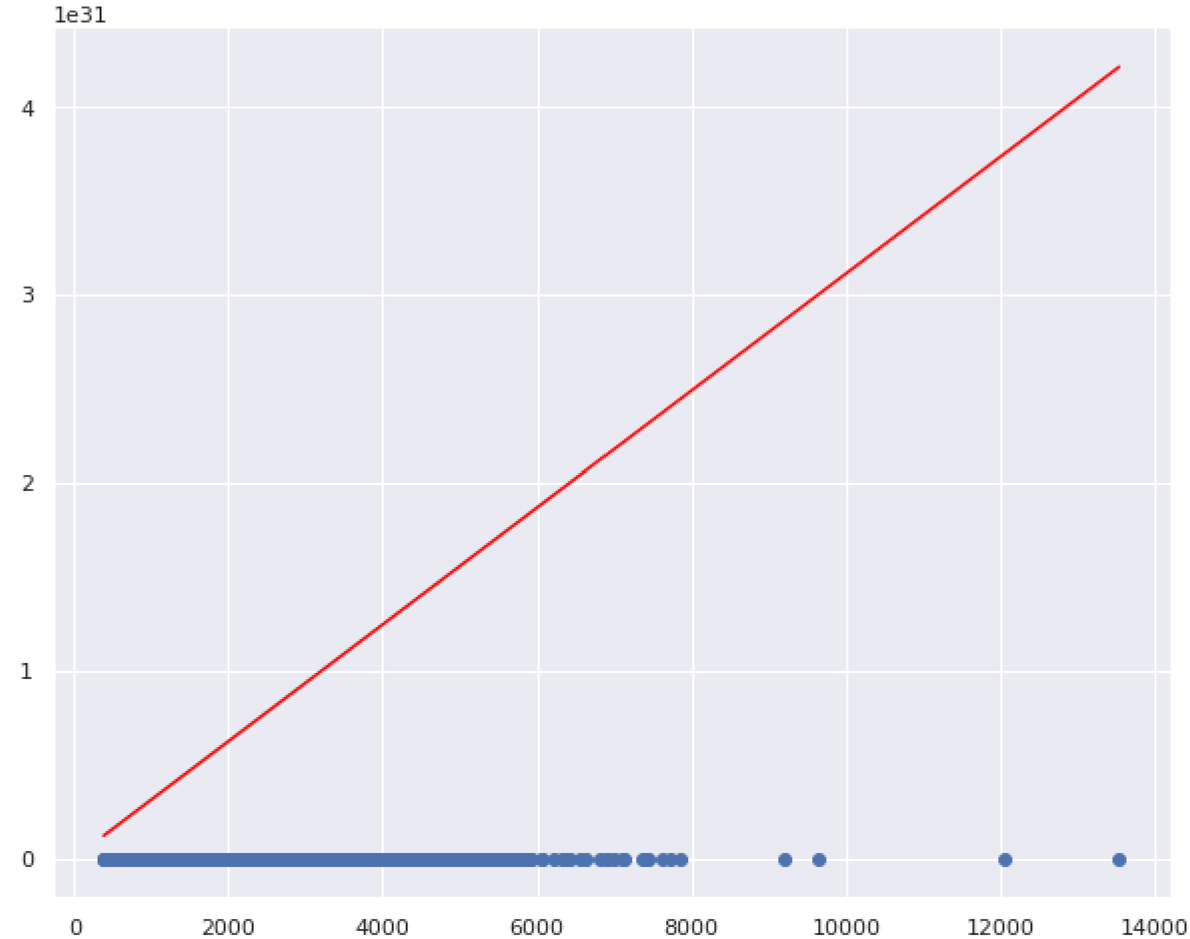
	sqft_living	price
17719	2030	572500.0
10646	3670	883000.0
1949	1008	480000.0
20322	4410	1240000.0
2072	1200	225000.0
...	...	...
6500	3450	755000.0
19857	3100	435000.0
14528	2300	294000.0
899	1260	291500.0
12706	2460	835000.0

15129 rows × 2 columns

In [ ]:
```
# Building the model
m = 0
c = 0

L = 0.01  # The learning Rate
epochs = 5  # The number of iterations to perform gradient descent

n = float(len(df1['sqft_living'])) # Number of elements in X

# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*(df1['sqft_living']) + c  # The current predicted value of Y
    D_m = (-2/n) * sum(df1['sqft_living'] * (df1['price'] - Y_pred))  # Derivativ
    D_c = (-2/n) * sum(df1['price'] - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

print (m, c)
```

```
3.107889241700975e+27 1.2504354911183835e+24
```

In [ ]:
```
# Making predictions
Y_pred = m*(testing['sqft_living']) + c

plt.scatter(testing['sqft_living'],testing['price'])
plt.plot([min(testing['sqft_living']), max(testing['sqft_living'])], [min(Y_pred)
plt.show()
```



In [ ]:
```
X = testing['sqft_living'] ## Assign TV ad value to X
y = testing['price'] ## assign sales values to y

X2 = sm.add_constant(X)# Assign stat model  constant to X2
est = sm.OLS(y, X2) # Build Ordinary Least square
est2 = est.fit() #Fitting OLS Regression
print(est2.summary()) # Printing OLS Results
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.507
Model:                            OLS   Adj. R-squared:                  0.507
Method:                 Least Squares   F-statistic:                     6656.
Date:                Sun, 29 Aug 2021   Prob (F-statistic):               0.00
Time:                        08:54:01   Log-Likelihood:                -90024.
No. Observations:                6484   AIC:                         1.801e+05
Df Residuals:                    6482   BIC:                         1.801e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -4.348e+04   7855.020     -5.535      0.000   -5.89e+04   -2.81e+04
sqft_living   279.8463      3.430     81.584      0.000     273.122     286.571
==============================================================================
Omnibus:                     4448.786   Durbin-Watson:                   2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           182253.906
Skew:                           2.762   Prob(JB):                         0.00
Kurtosis:                      28.379   Cond. No.                     5.59e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 5.59e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation:

House prices were predicted with Linear Regression using Gradient descent, along with scatterplot.

In [ ]:

ML LAB 5

Explore and implement Logistic Regression by Stochastic Gradient Descent in a given business scenario and comment on its efficiency and performance.

In [1]:
```
#Imports for data analysis, data wrangling and visualization
import pandas as pd
import numpy as np
import random as rand
import seaborn as sns
import matplotlib.pyplot as plt

#Machine learning imports
from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import Perceptron, SGDClassifier
```

In [2]:
```
#Loading the data
train_df = pd.read_csv('C:/Users/user/Downloads/titanicpredictions-main/titanicpr
test_df = pd.read_csv('C:/Users/user/Downloads/titanicpredictions-main/titanicpre
combine = [train_df,test_df]
```

In [3]:
```
#Checking the column names
print(train_df.columns.values)

#Categorical variables - Survived, Sex, Embarked, Pclass
#Numerical variables - Age, Fare, SibSP, Parch
#Ticket is a mix of numeric and alphanumeric data types and Cabin is Alphanumeric
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

In [4]:
```
#Checking the training DF
train_df.tail(15)
#Cabin and Age contain null values
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
876	877	0	3	Gustafsson, Mr. Alfred Ossian	male	20.0	0	0	7534	9.8458
877	878	0	3	Petroff, Mr. Nedelio	male	19.0	0	0	349212	7.8958
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230433	26.0000
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

In [5]:
```
#Checking the test DF
test_df.tail(15)
#Cabin and Age contain null values
```

Out[5]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
403	1295	1	Carrau, Mr. Jose Pedro	male	17.0	0	0	113059	47.1000	NaN
404	1296	1	Frauenthal, Mr. Isaac Gerald	male	43.0	1	0	17765	27.7208	D40
405	1297	2	Nourney, Mr. Alfred (Baron von Drachstedt")"	male	20.0	0	0	SC/PARIS 2166	13.8625	D38
406	1298	2	Ware, Mr. William Jeffery	male	23.0	1	0	28666	10.5000	NaN
407	1299	1	Widener, Mr. George Dunton	male	50.0	1	1	113503	211.5000	C80
408	1300	3	Riordan, Miss. Johanna Hannah""	female	NaN	0	0	334915	7.7208	NaN
409	1301	3	Peacock, Miss. Treasteall	female	3.0	1	1	SOTON/O.Q. 3101315	13.7750	NaN
410	1302	3	Naughton, Miss. Hannah	female	NaN	0	0	365237	7.7500	NaN
411	1303	1	Minahan, Mrs. William Edward (Lillian E Thorpe)	female	37.0	1	0	19928	90.0000	C78
412	1304	3	Henriksson, Miss. Jenny Lovisa	female	28.0	0	0	347086	7.7750	NaN
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN

In [6]:
```
#Checking the data types of the features (7 features are integers or floats (6 in
train_df.info()
print('-'*40)
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

In [7]:
```
#Checking the numerical distribution of numerical features across the samples
train_df.describe()
#891 samples of 2224 that were aboard
#Around 38% survived, compared to 32% of the actual rate
```

Out[7]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [8]:
```
train_df.describe(include=['O'])
# Names are unique
# 65% are male (577/891)
# A lot of the cabins are shared (147 cabins), also duplicate values
# 3 possible embarked values, S is the most popular (644/889)
# Ticket feature has a high ratio of duplicate values (681/891)
```

Out[8]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Montvila, Rev. Juozas	male	347082	G6	S
freq	1	577	7	4	644

Assumptions based on the data analysis so far:

1) Correlating: We need to know how each of the features correlate with survival.

2) Completing: We need to complete the age and embarked features as they are probably related to survival.

3) Correcting: Ticket (high ratio of duplicates), Cabin (highly incomplete with many missing values) and passangerID (does not contribute to survival) should be dropped

4) Creating: We may need to create a new feature called 'Family' based on Parch and SibSp to get total count of family members. We may want to manipulate the name feature to extract title as a new feature. We may want to group age into bands as this turns the numerical feature into an ordinal categorical feature. We may also want to create a fare range to see if it correlates with survival.

5) Classifying: Based on the problem description we can check for some assumptions -> Woman (sex=female), Children and Upper Class Passengers (pclass=1) are more likely to have survived

In [9]:
```
#To confirm some of our assumptions we can analyze feature correlation by pivoting
train_df[['Pclass','Survived']].groupby(['Pclass'], as_index = False).mean().sort_
```

Out[9]:

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

In [10]:
```
train_df[['Sex','Survived']].groupby(['Sex'], as_index = False).mean().sort_value
```

Out[10]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

In [11]:
```
train_df[['SibSp','Survived']].groupby(['SibSp'], as_index = False).mean().sort_v
```

Out[11]:

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

In [12]:
```
train_df[['Parch','Survived']].groupby(['Parch'], as_index = False).mean().sort_v
```

Out[12]:

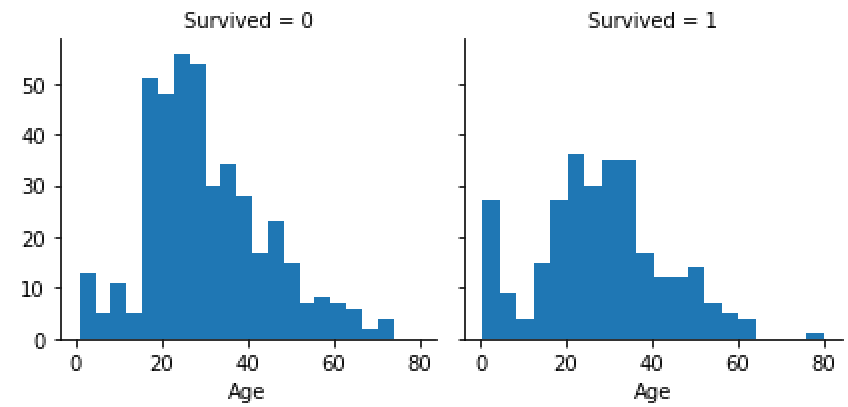
	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

Analyze by visualizing data

1) Correlating Numerical Features

In [13]:
```
graph = sns.FacetGrid(train_df,col = 'Survived')
graph.map(plt.hist, 'Age', bins = 20)
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0x1f0e4ccf4f0>



Observations:

```
1) Babies (age<4) had a high survival rate
2) Oldest passanger survived (age=80)
3) A lot with passangers age 15 to 25 did not survived
4) Most passangers are in the 15-35 age range
```

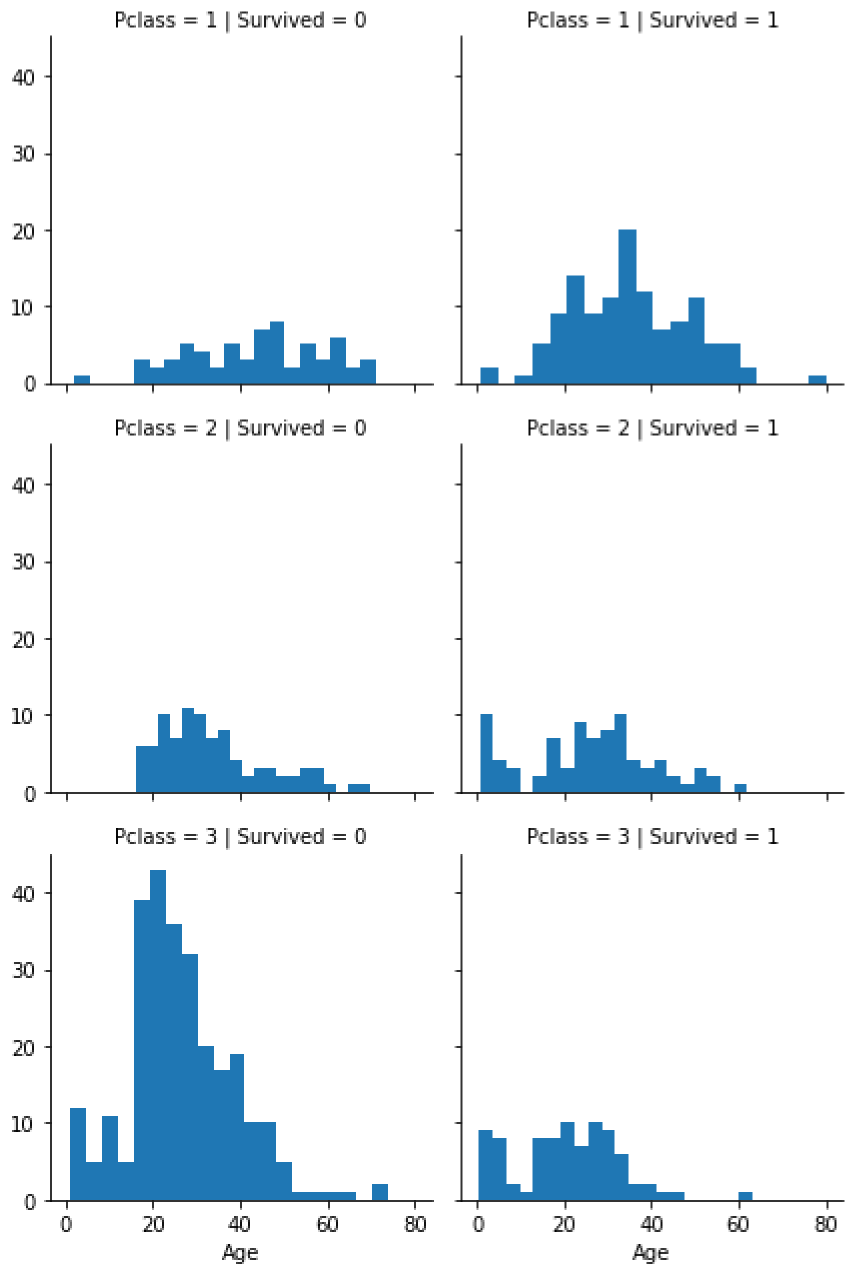
Decisions

```
1) We should consider age in our model training
2) We should complete the age feature for null values
3) We should band age groups to perform a better analysis
```

2) Correlating Numerical and Ordinal Features

In [14]:
```
graph = sns.FacetGrid(train_df, col = 'Survived',row='Pclass')
graph.map(plt.hist, 'Age', bins = 20)
graph.add_legend()
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x1f0e5520670>



Observations:

```
    1) Pclass=3 had the higher number of passangers but most of them didn
t survive
    2) Babies in pclass = 2 and 3 mostly survived so it further qualifies
our assumption about it
    3) Most passengers in pclass = 1 survived
    4) Pclass varies in terms of age distribution
```

Decisions:

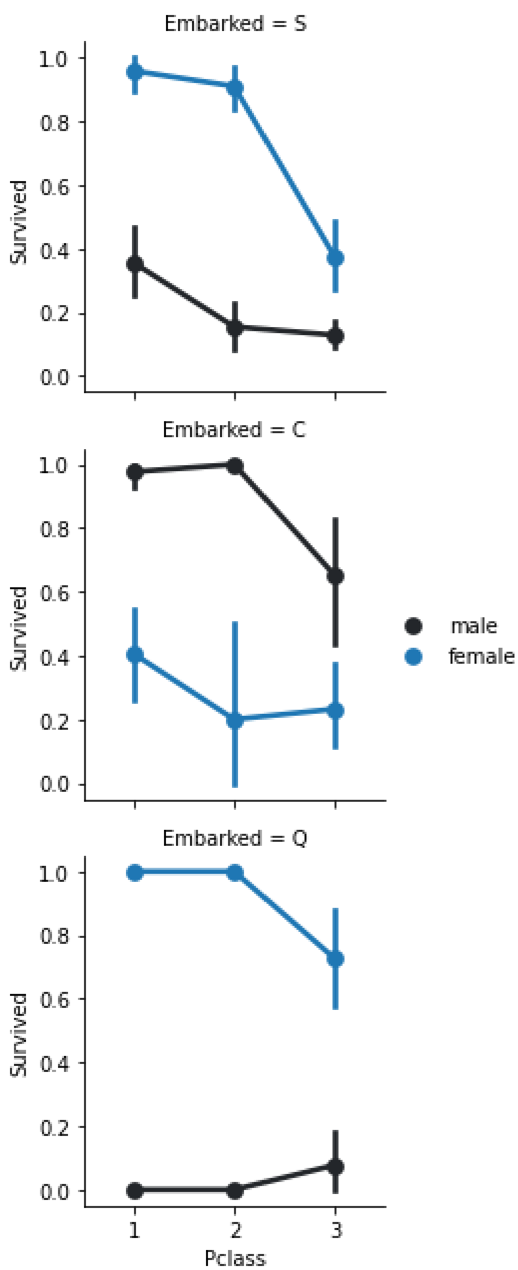
```
    1) Consider pclass for training
```

3) Correlating Categorical Features

In [15]:
```
graph = sns.FacetGrid(train_df, row = 'Embarked')
graph.map(sns.pointplot, 'Pclass','Survived','Sex')
graph.add_legend()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\axisgrid.py:643: UserWarning:
Using the pointplot function without specifying `order` is likely to produce an
incorrect plot.
  warnings.warn(warning)
C:\Users\user\anaconda3\lib\site-packages\seaborn\axisgrid.py:648: UserWarning:
Using the pointplot function without specifying `hue_order` is likely to produc
e an incorrect plot.
  warnings.warn(warning)
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x1f0e54cdc10>



Observations:

```
    1) Female passengers had a much better survival rate
    2) Exception is embarked = C where males had a higher survival rate
    3) Males had a higher survival rate in pclass=3 when compared do pcla
ss=2 for C and Q ports
```

Decisions:

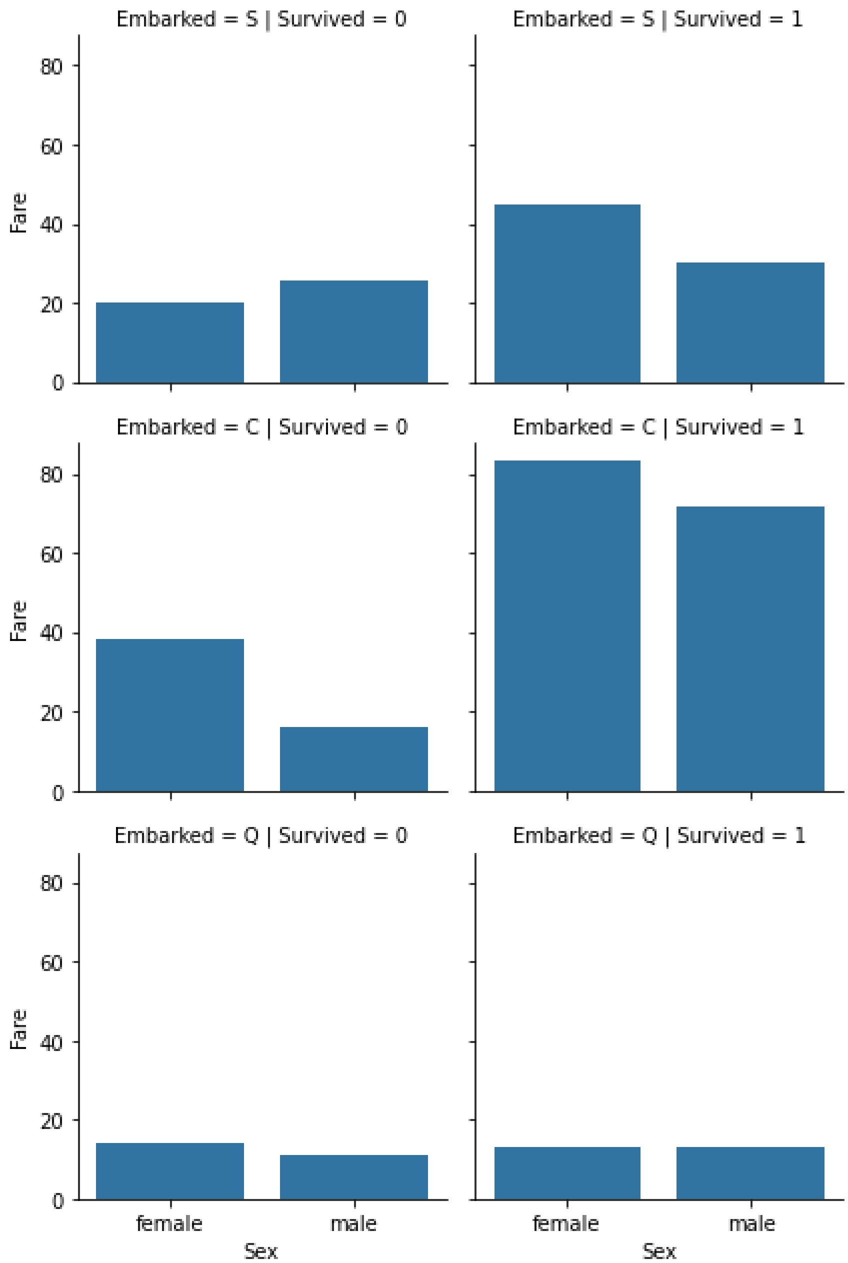
```
    1) Add sex feature to the model training
    2) Complete and add embarked feature to the model training
```

4) Correlating Categorical and Numerical Features

In [16]:
```
graph = sns.FacetGrid(train_df, col = 'Survived',row='Embarked')
graph.map(sns.barplot, 'Sex','Fare', ci = None)
graph.add_legend()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\axisgrid.py:643: UserWarning:
Using the barplot function without specifying `order` is likely to produce an i
ncorrect plot.
  warnings.warn(warning)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x1f0e5c35f70>



Observations:

```
1) Higher fare rates had higher survival rates
2) Port of embarkation correlates with the survival rates
```

Decisions:

```
1) We should band the fare rates and consider them in the model
```

Wrangle the data

In [17]:
```
#dropping unnecessary features to speed up the training
IDs = test_df['PassengerId']
train_df.drop(['Ticket','Cabin','PassengerId'], inplace=True, axis = 1)
test_df.drop(['Ticket','Cabin','PassengerId'], inplace=True, axis = 1)
combine = [train_df,test_df]
```

In [18]:
```
#creating new feature from existing - 'name' - extracting the characters of the s
for dataset in combine:
    dataset['Title'] = dataset.Name.str.extract('([A-Za-z]+)\.', expand = False)

pd.crosstab(train_df['Title'], train_df['Sex'])
```

Out[18]:

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

In [19]:
```
#we can group the uncommon titles on a category named other
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col'
    'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Other')
    dataset['Title'] = dataset['Title'].replace('Mlle','Miss')
    dataset['Title'] = dataset['Title'].replace('Ms','Miss')
    dataset['Title'] = dataset['Title'].replace('Mme','Mrs')

train_df.groupby('Title').mean()
```

Out[19]:

	Survived	Pclass	Age	SibSp	Parch	Fare
Title						
Master	0.575000	2.625000	4.574167	2.300000	1.375000	34.703125
Miss	0.702703	2.291892	21.845638	0.702703	0.540541	43.800092
Mr	0.156673	2.410058	32.368090	0.288201	0.152805	24.441560
Mrs	0.793651	1.992063	35.788991	0.690476	0.825397	45.330290
Other	0.347826	1.347826	45.545455	0.347826	0.086957	37.169748

In [20]:
```
#Then we can convert the categorical titles to ordinal
title_dict = {'Mr': 1,
              'Miss': 2,
              'Mrs': 3,
              'Master': 4,
              'Other': 5 }
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_dict)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df.head()
```

Out[20]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	1
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	3
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	2
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	3
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1

In [21]:
```
#now we can also drop the name feature
train_df.drop(['Name'], axis = 1 , inplace = True)
train_df.head()
```

Out[21]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	male	22.0	1	0	7.2500	S	1
1	1	1	female	38.0	1	0	71.2833	C	3
2	1	3	female	26.0	0	0	7.9250	S	2
3	1	1	female	35.0	1	0	53.1000	S	3
4	0	3	male	35.0	0	0	8.0500	S	1

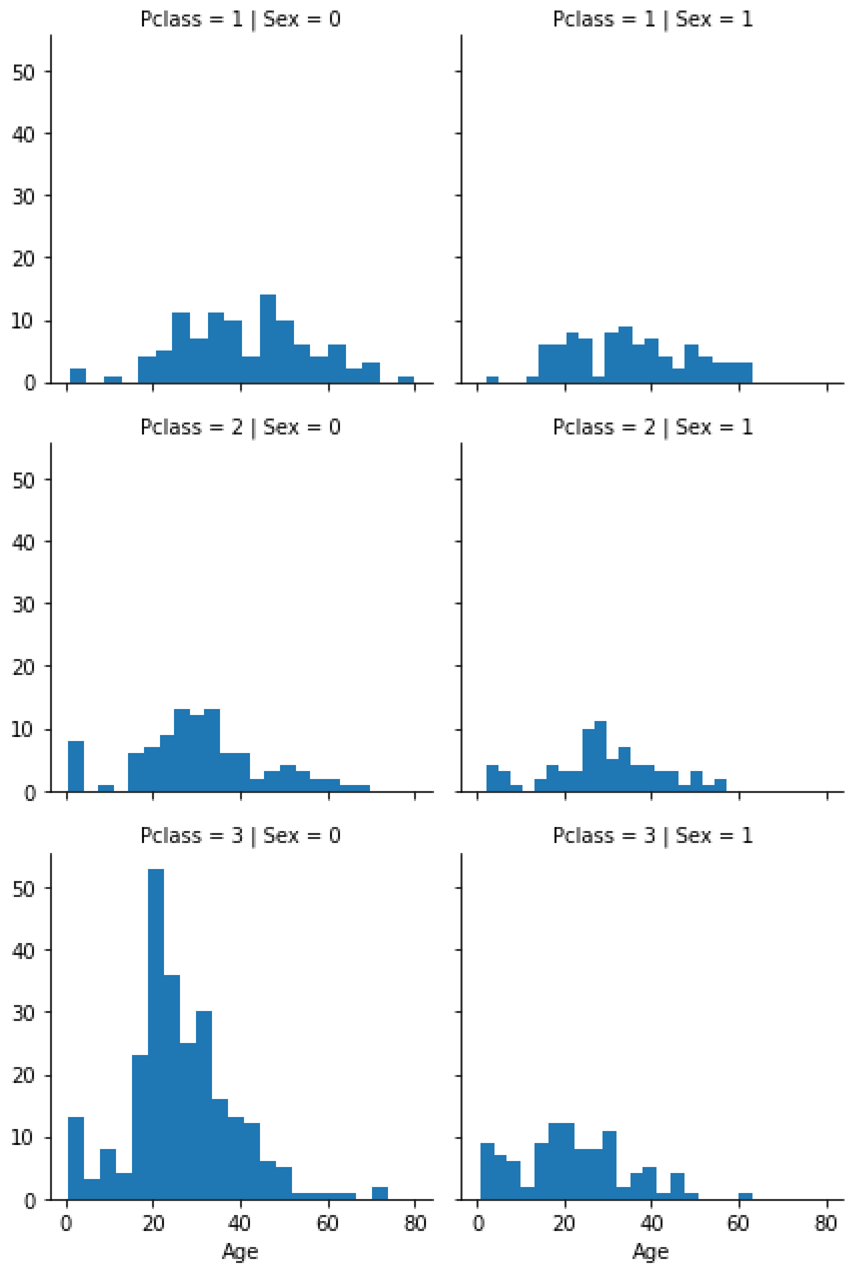
In [22]:
```
#converting the categorical feature (sex) into ordinal
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map({'female':1,
                                         'male':0})
train_df.head()
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

In [23]:
```
#now we should estimate or complete the feature with missing or null values, we'l
#we will guess the missing values for age by using other correlated features like
graph = sns.FacetGrid(train_df, row = 'Pclass', col = 'Sex')
graph.map(plt.hist, 'Age', bins = 20)
graph.add_legend()
```

Out[23]: <seaborn.axisgrid.FacetGrid at 0x1f0e5bf5bb0>



In [24]:
```
#Lets prepare an empty array to contain the guessed age values for all the 6 pcla
guess_ages = np.zeros((2,3))
guess_ages
```

Out[24]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [25]:
```
#Now we iterate to get the median of each combination of pclass and sex, and use

for dataset in combine:
    for i in range (0,2):
        for j in range(0,3):
           guess_df = dataset[(dataset['Sex'] == i) & \
                                  (dataset['Pclass'] == j+1)]['Age'].dropna()
           age_guess = guess_df.median()
           guess_ages[i,j] = int(age_guess/.5 +.5)*.5 #convert random age float to

    for i in range (0,2):
        for j in range (0,3):
            dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.P
                    'Age'] = guess_ages[i,j]

    dataset['Age'] = dataset['Age'].astype(int)

train_df.head()
```

Out[25]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22	1	0	7.2500	S	1
1	1	1	1	38	1	0	71.2833	C	3
2	1	3	1	26	0	0	7.9250	S	2
3	1	1	1	35	1	0	53.1000	S	3
4	0	3	0	35	0	0	8.0500	S	1

In [26]:
```
#Checking to see how the ages split in 5 different bands (in absolut numbers not 
train_df['AgeBand'] = pd.cut(train_df['Age'],5)
train_df[['AgeBand','Survived']].groupby(['AgeBand'], as_index = False).mean().so
```

Out[26]:

	AgeBand	Survived
0	(-0.08, 16.0]	0.550000
1	(16.0, 32.0]	0.337374
2	(32.0, 48.0]	0.412037
3	(48.0, 64.0]	0.434783
4	(64.0, 80.0]	0.090909

In [27]:
```
#Attributing a number to each of the agebands
for dataset in combine:
    dataset.loc[dataset['Age'] <= 16,'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32),'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48),'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64),'Age'] = 3
    dataset.loc[dataset['Age'] > 64,'Age'] = 4
train_df.head()
```

Out[27]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	AgeBand
0	0	3	0	1	1	0	7.2500	S	1	(16.0, 32.0]
1	1	1	1	2	1	0	71.2833	C	3	(32.0, 48.0]
2	1	3	1	1	0	0	7.9250	S	2	(16.0, 32.0]
3	1	1	1	2	1	0	53.1000	S	3	(32.0, 48.0]
4	0	3	0	2	0	0	8.0500	S	1	(32.0, 48.0]

In [28]:
```
train_df.drop(columns='AgeBand', inplace =True, axis = 1)
combine = [train_df,test_df]
train_df
```

Out[28]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	1	1	0	7.2500	S	1
1	1	1	1	2	1	0	71.2833	C	3
2	1	3	1	1	0	0	7.9250	S	2
3	1	1	1	2	1	0	53.1000	S	3
4	0	3	0	2	0	0	8.0500	S	1
...	...	...	...	...	...	...	...	...	...
886	0	2	0	1	0	0	13.0000	S	5
887	1	1	1	1	0	0	30.0000	S	2
888	0	3	1	1	1	2	23.4500	S	2
889	1	1	0	1	0	0	30.0000	C	1
890	0	3	0	1	0	0	7.7500	Q	1

891 rows × 9 columns

In [29]:
```
#Here we are aggregating the number of partners (parch) with simblings (SibSp) an
for dataset in combine:
    dataset['FamilySize'] = dataset['Parch'] + dataset['SibSp'] + 1
    dataset.drop(columns=['SibSp','Parch'],inplace=True,axis=1)
train_df[['FamilySize','Survived']].groupby(['FamilySize'], as_index=False).mean(
```

Out[29]:

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

In [30]:
```
#Creating a feature 'isAlone' will help us to correlate the fact of being alone w
for dataset in combine:
    dataset['isAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'isAlone'] = 1
    dataset.drop('FamilySize',inplace=True,axis=1) #we can also drop family size 
train_df[['isAlone','Survived']].groupby(['isAlone']).mean().sort_values(by=['Sur
#Its possible to see that the alone people had a higher survival mean rate
```

Out[30]:

	Survived
isAlone	
0	0.505650
1	0.303538

In [31]:
```
#We can create a new feature multiplying age and the pclass, so in theory the Low
for dataset in combine:
    dataset['AgeClass'] = dataset['Age'] * dataset['Pclass']

train_df[['Age','Pclass','AgeClass']].head(10)
```

Out[31]:

	Age	Pclass	AgeClass
0	1	3	3
1	2	1	2
2	1	3	3
3	2	1	2
4	2	3	6
5	1	3	3
6	3	1	3
7	0	3	0
8	1	3	3
9	0	2	0

In [32]:
```
#Checking the embarked feature we can see that S is the most common port, so we'l
train_df.Embarked.describe()
```

Out[32]:
```
count      889
unique       3
top          S
freq       644
Name: Embarked, dtype: object
```

In [33]:
```
freq_port = 'S'
for dataset in combine:
    dataset['Embarked'].fillna(freq_port,inplace=True)
train_df[['Embarked','Survived']].groupby(['Embarked']).mean().sort_values(by=['S
#Its possible to see that the S port had the lower mean survival rate and C had th
```

Out[33]:

Embarked	Survived
S	0.339009
Q	0.389610
C	0.553571

In [34]:
```
#Mapping the ports
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map({'S':0,
    'C':1,
    'Q':2}).astype(int)
train_df.head(10)
```

Out[34]:

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	0	3	0	1	7.2500	0	1	0	3
1	1	1	1	2	71.2833	1	3	0	2
2	1	3	1	1	7.9250	0	2	1	3
3	1	1	1	2	53.1000	0	3	0	2
4	0	3	0	2	8.0500	0	1	1	6
5	0	3	0	1	8.4583	2	1	1	3
6	0	1	0	3	51.8625	0	1	1	3
7	0	3	0	0	21.0750	0	4	0	0
8	1	3	1	1	11.1333	0	3	0	3
9	1	2	1	0	30.0708	1	3	0	0

In [35]:
```
#Complete fare for the single missing value on the test DF using the mode
test_df['Fare'].fillna(test_df['Fare'].dropna().median(),inplace=True)
test_df
```

Out[35]:

	Pclass	Name	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	3	Kelly, Mr. James	0	2	7.8292	2	1	1	6
1	3	Wilkes, Mrs. James (Ellen Needs)	1	2	7.0000	0	3	0	6
2	2	Myles, Mr. Thomas Francis	0	3	9.6875	2	1	1	6
3	3	Wirz, Mr. Albert	0	1	8.6625	0	1	1	3
4	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	1	1	12.2875	0	3	0	3
...	...	...	...	...	...	...	...	...	...
413	3	Spector, Mr. Woolf	0	1	8.0500	0	1	1	3
414	1	Oliva y Ocana, Dona. Fermina	1	2	108.9000	1	5	1	2
415	3	Saether, Mr. Simon Sivertsen	0	2	7.2500	0	1	1	6
416	3	Ware, Mr. Frederick	0	1	8.0500	0	1	1	3
417	3	Peter, Master. Michael J	0	1	22.3583	1	4	0	3

418 rows × 9 columns

In [36]:
```
#We can now create the bands for the fare, but as we did for the age we have to c
train_df['FareBand'] = pd.qcut(train_df['Fare'],4) #qcut divides into 4 quantiles
train_df[['FareBand','Survived']].groupby(['FareBand'],as_index=False).mean().sor
#We can see that the higher the band the higher the survival mean rate
```

Out[36]:

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

In [37]:
```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] =
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']  =
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df.drop(['FareBand'], axis=1, inplace=True)
combine = [train_df, test_df]

train_df
```

Out[37]:

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3
3	1	1	1	2	3	0	3	0	2
4	0	3	0	2	1	0	1	1	6
...	...	...	...	...	...	...	...	...	...
886	0	2	0	1	1	0	5	1	2
887	1	1	1	1	2	0	2	1	1
888	0	3	1	1	2	0	2	0	3
889	1	1	0	1	2	1	1	1	1
890	0	3	0	1	0	2	1	1	3

891 rows × 9 columns

In [38]:
```
#And now both our datasets are ready
test_df.drop(columns=['Name'],inplace=True,axis=1)
test_df.head(100)
```

Out[38]:

	Pclass	Sex	Age	Fare	Embarked	Title	isAlone	AgeClass
0	3	0	2	0	2	1	1	6
1	3	1	2	0	0	3	0	6
2	2	0	3	1	2	1	1	6
3	3	0	1	1	0	1	1	3
4	3	1	1	1	0	3	0	3
...	...	...	...	...	...	...	...	...
95	3	0	1	0	0	1	1	3
96	1	1	4	3	0	3	0	4
97	3	0	1	1	0	1	1	3
98	3	1	1	0	0	2	1	3
99	3	0	2	1	0	1	1	6

100 rows × 8 columns

In [39]:
```
X_train = train_df.drop('Survived',axis=1)
Y_train = train_df['Survived']
X_test = test_df.copy()
X_train.shape,Y_train.shape,X_test.shape
```

Out[39]:
```
((891, 8), (891,), (418, 8))
```

In [40]:
```
#Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train,Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train,Y_train) * 100,2)
print(acc_log,'%')
```

```
81.37 %
```

In [41]:
```
coeff = pd.DataFrame(train_df.columns.delete(0))
coeff.columns = ['Feature']
coeff['Correlation'] = pd.Series(logreg.coef_[0])
coeff.sort_values(by = 'Correlation', ascending = False)
```

Out[41]:

	Feature	Correlation
1	Sex	2.201057
5	Title	0.406027
4	Embarked	0.276628
6	isAlone	0.185986
7	AgeClass	-0.050260
3	Fare	-0.071665
2	Age	-0.469638
0	Pclass	-1.200309

In [42]:
```
#Stochastic Gradient Descent
sgd = SGDClassifier()
sgd.fit(X_train,Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train,Y_train)*100,2)
print(acc_sgd,'%')
```

```
69.81 %
```

In [43]:
```
models = pd.DataFrame({'Model':
['SGD','Logistic Regression'],
'Scores':
[acc_sgd,acc_log]}
)
models = models.sort_values(by='Scores',ascending=False).reset_index(drop=True)
models
```

Out[43]:

	Model	Scores
0	Logistic Regression	81.37
1	SGD	69.81

In [ ]:

ML LAB 6

Implement Decision Tree algorithm in a given business environment and comment on its efficiency and performance.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from warnings import filterwarnings
filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('C:/Users/user/Downloads/archive (2)/drug200.csv')
```

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

Out[3]:

	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

In [4]:
```
data.isnull().sum()
```

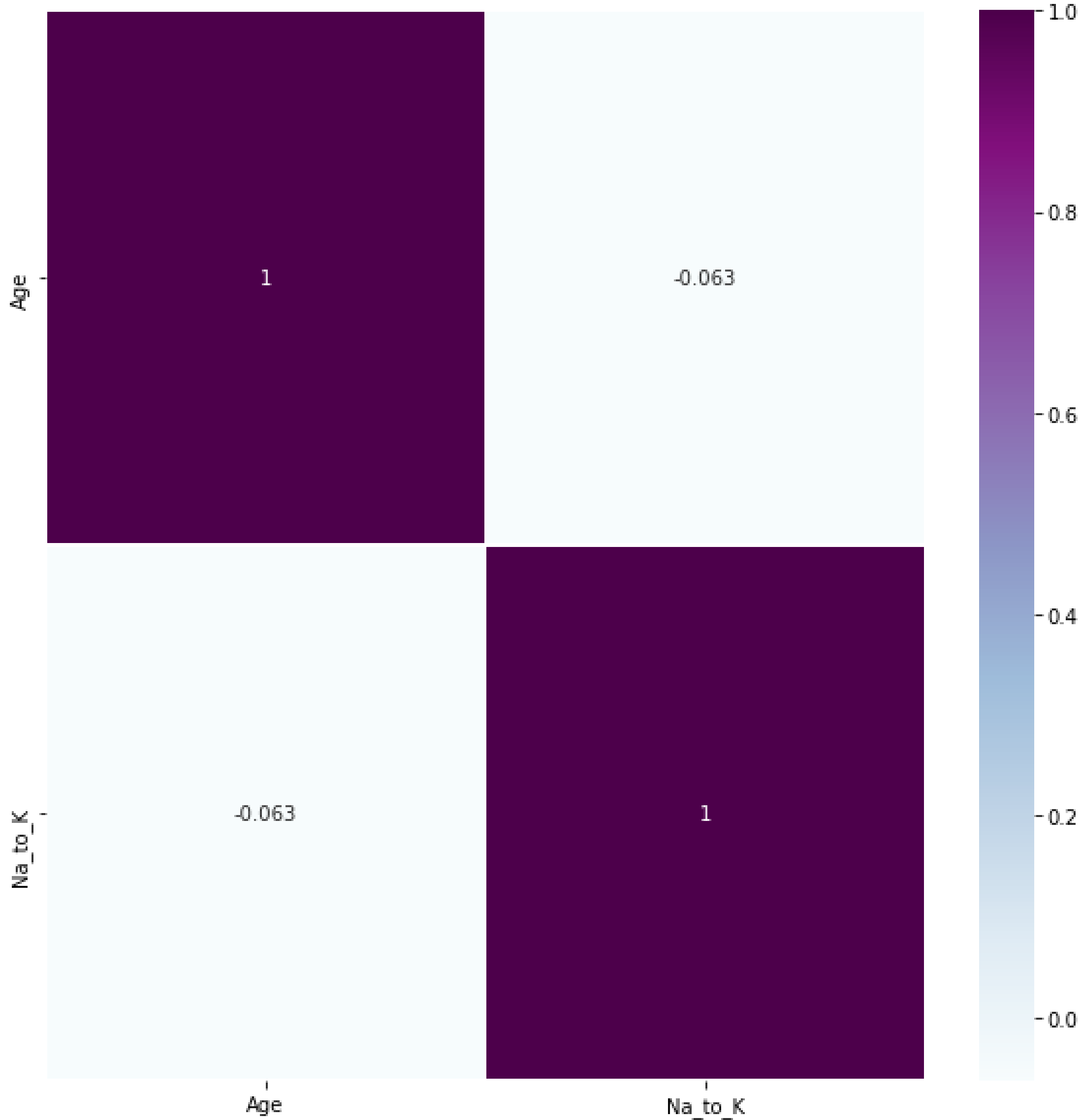
Out[4]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

In [5]:
```
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
```

there is no missing values in the date we have 6 coulmns and 200 rows

In [6]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```

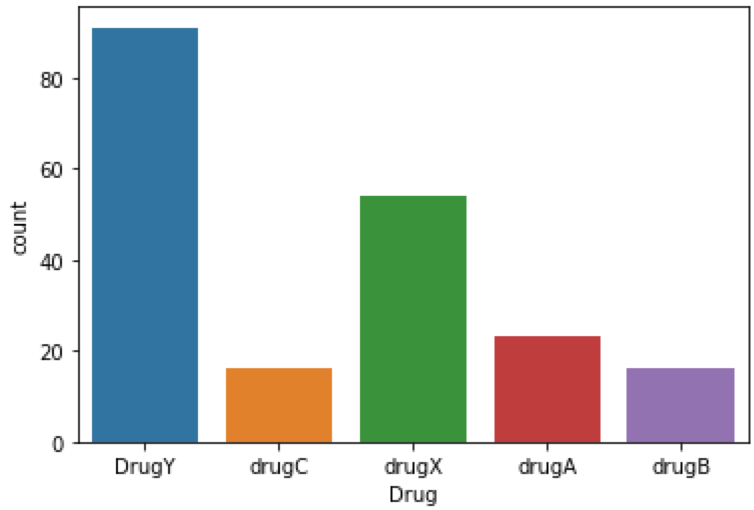


In [7]:
```
data['Drug'].value_counts()
```

Out[7]:
```
DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: Drug, dtype: int64
```

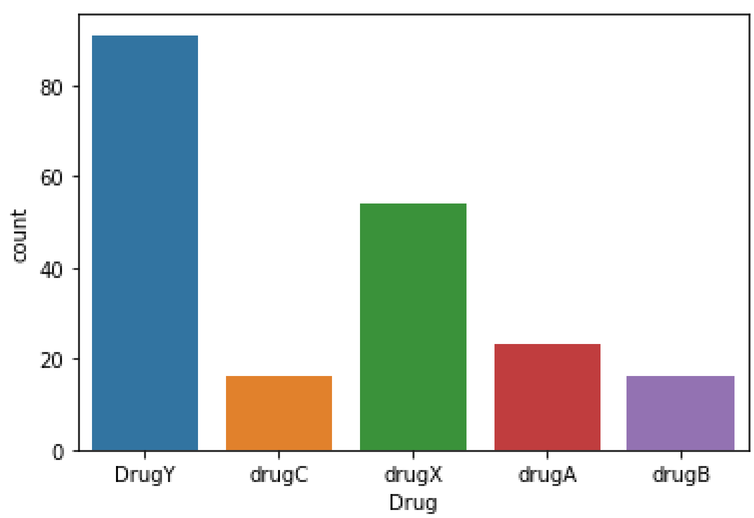
In [8]:
```
sns.countplot(x = 'Drug', data= data)
```

Out[8]: <AxesSubplot:xlabel='Drug', ylabel='count'>



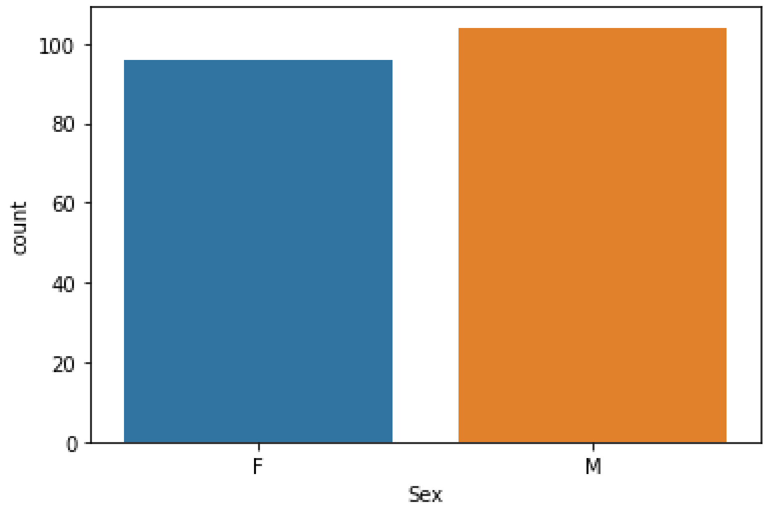
In [9]:
```
sns.countplot(x = 'Drug', data= data)
```

Out[9]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [10]:
```
sns.countplot(x = 'Sex', data= data)
```

Out[10]: <AxesSubplot:xlabel='Sex', ylabel='count'>

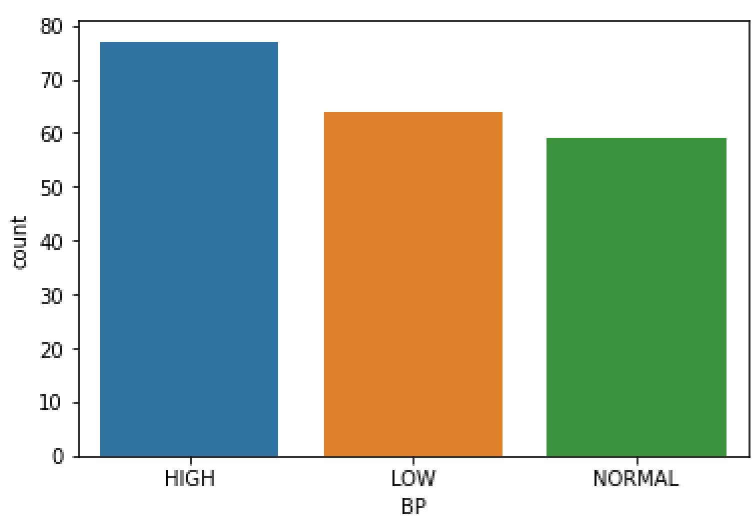


In [11]:
```
data['BP'].value_counts()
```

Out[11]:
```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

In [12]:
```
sns.countplot(x = 'BP', data= data)
```

Out[12]: <AxesSubplot:xlabel='BP', ylabel='count'>

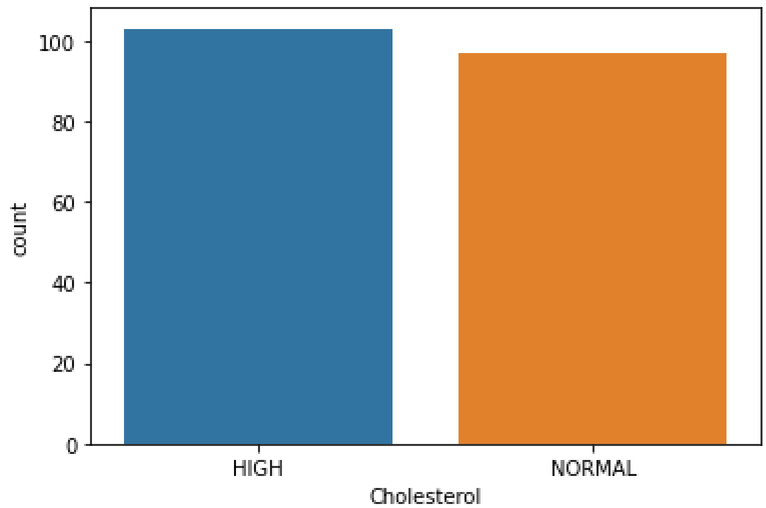


In [13]:
```
data['Cholesterol'].value_counts()
```

Out[13]:
```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

In [14]:
```
sns.countplot(x = 'Cholesterol', data= data)
```

Out[14]:
```
<AxesSubplot:xlabel='Cholesterol', ylabel='count'>
```

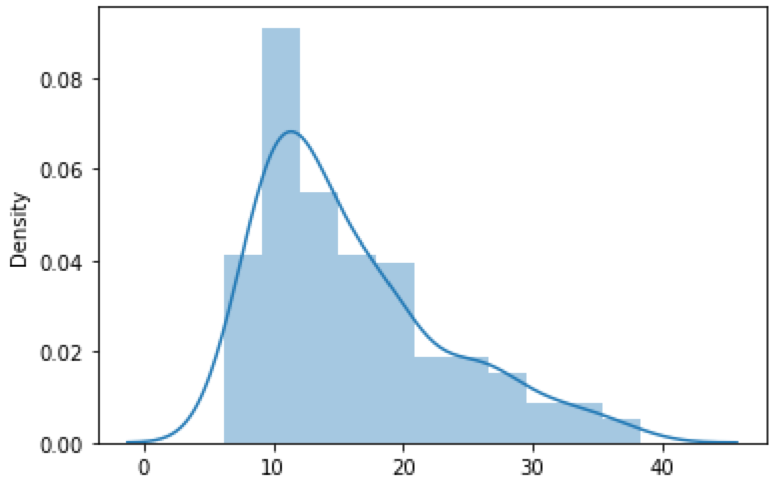


In [15]:
```
data['Na_to_K'].describe()
```

Out[15]:
```
count    200.000000
mean      16.084485
std        7.223956
min        6.269000
25%       10.445500
50%       13.936500
75%       19.380000
max       38.247000
Name: Na_to_K, dtype: float64
```

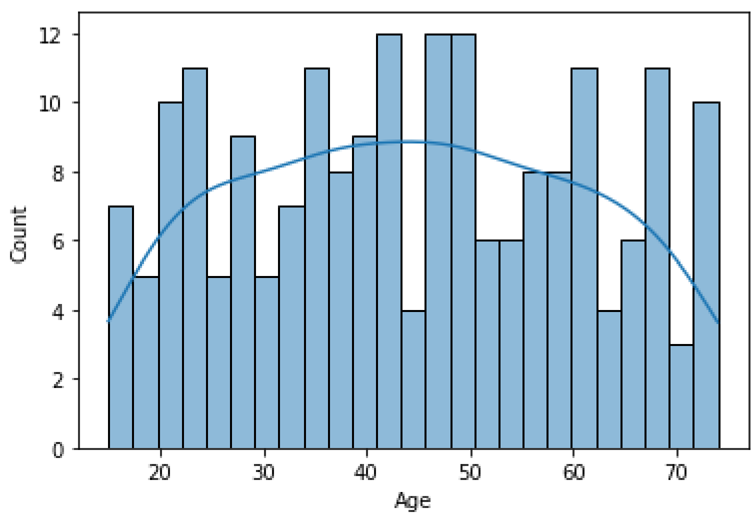
In [16]:
```
sns.distplot(x = data['Na_to_K'])
```

Out[16]: <AxesSubplot:ylabel='Density'>



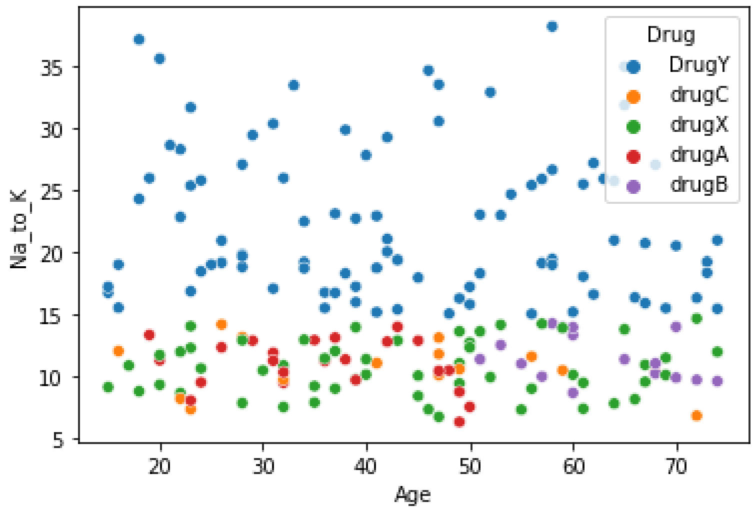
In [17]:
```
sns.histplot(x = 'Age', kde=True, bins = 25, data = data)
```

Out[17]: <AxesSubplot:xlabel='Age', ylabel='Count'>



In [18]:
```
sns.scatterplot(x = 'Age', y = 'Na_to_K', data = data, hue = 'Drug')
```

Out[18]:
```
<AxesSubplot:xlabel='Age', ylabel='Na_to_K'>
```



In the last fig we find all the items have more than 15 Na_to_K have DrugY type

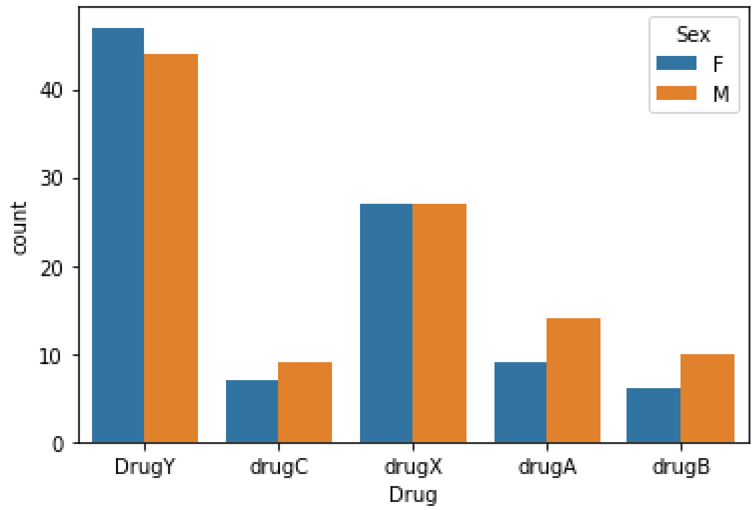
In the next We will find out the number of each Drug type per Sex

In [19]:
```
data_sex_drug = data.groupby(['Drug','Sex']).size().reset_index(name = 'count')
print(data_sex_drug)
```

```
    Drug Sex  count
0  DrugY   F     47
1  DrugY   M     44
2  drugA   F      9
3  drugA   M     14
4  drugB   F      6
5  drugB   M     10
6  drugC   F      7
7  drugC   M      9
8  drugX   F     27
9  drugX   M     27
```

In [20]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Sex')
```

Out[20]: <AxesSubplot:xlabel='Drug', ylabel='count'>

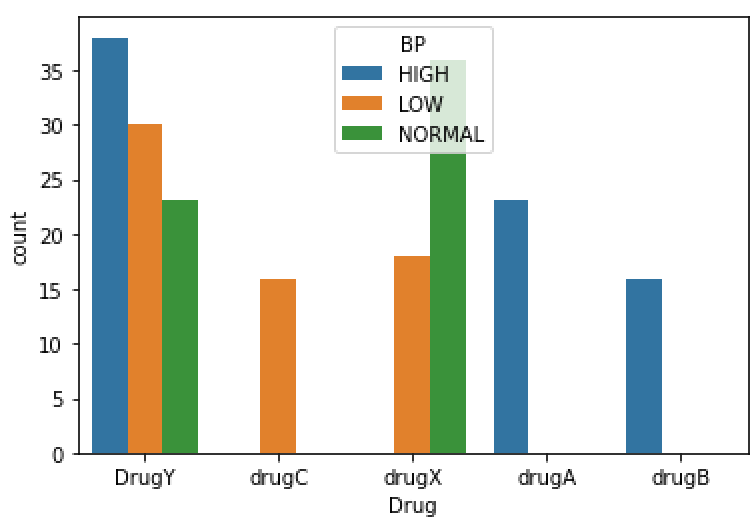


In [21]:
```
data_BP_drug = data.groupby(['Drug','BP']).size().reset_index(name = 'count')
print(data_BP_drug)
```

```
    Drug      BP  count
0  DrugY    HIGH     38
1  DrugY     LOW     30
2  DrugY  NORMAL     23
3  drugA    HIGH     23
4  drugB    HIGH     16
5  drugC     LOW     16
6  drugX     LOW     18
7  drugX  NORMAL     36
```

In [22]:
```
sns.countplot(x = 'Drug', data= data, hue = 'BP')
```

Out[22]: <AxesSubplot:xlabel='Drug', ylabel='count'>

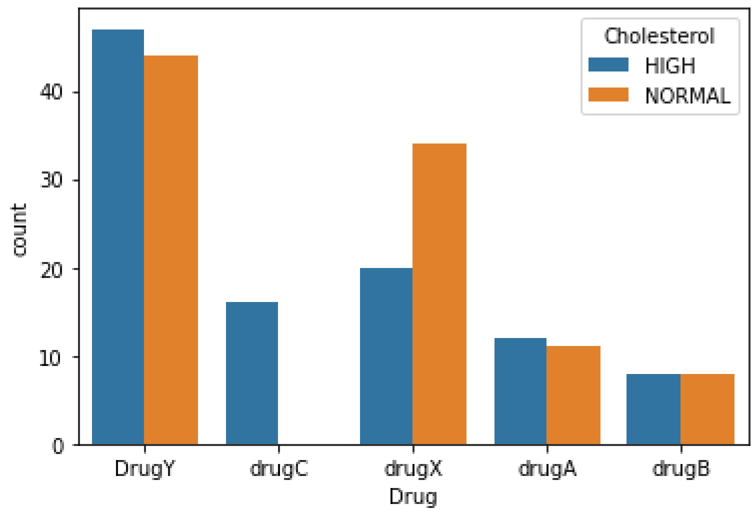


In [23]:
```
data_Cholesterol_drug = data.groupby(['Drug','Cholesterol']).size().reset_index(n
print(data_Cholesterol_drug)
```

```
    Drug Cholesterol  count
0  DrugY        HIGH     47
1  DrugY      NORMAL     44
2  drugA        HIGH     12
3  drugA      NORMAL     11
4  drugB        HIGH      8
5  drugB      NORMAL      8
6  drugC        HIGH     16
7  drugX        HIGH     20
8  drugX      NORMAL     34
```

In [24]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Cholesterol')
```

Out[24]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [25]:
```
data['Sex'] = data['Sex'].map({'M': 1, 'F': 0})
data['Cholesterol'] = data['Cholesterol'].map({'HIGH' : 1, 'NORMAL' : 0})
data['Drug'] = data['Drug'].map({'DrugY':1, 'drugC':2, 'drugX':3, 'drugA':4, 'dru
data.head()
```

Out[25]:

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

In [26]:
```
data.shape
```

Out[26]: (200, 6)

In [27]:
```
data = pd.get_dummies(data)
data.head()
```

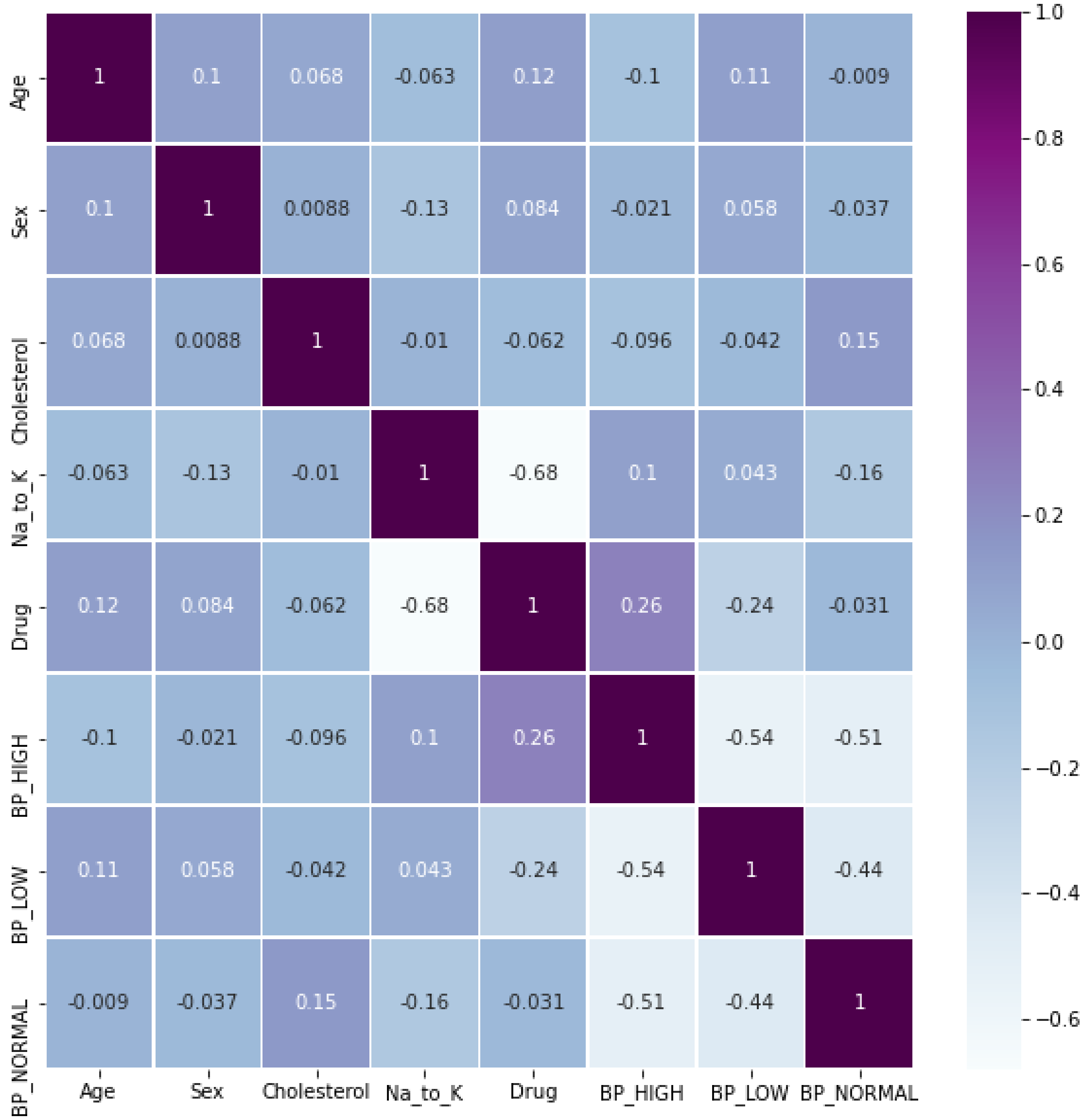
Out[27]:

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

In [28]:
```
data.shape
```

Out[28]: (200, 8)

In [29]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```



In [30]:
```
X = data.drop('Drug', axis = 1).values
y = data['Drug'].values.reshape((-1,1))
```

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
print('x train shape {}'.format(X_train.shape))
print('x test shape  {}'.format(X_test.shape))
print('y train shape {}'.format(y_train.shape))
print('y test shape  {}'.format(y_test.shape))
```

```
x train shape (160, 7)
x test shape  (40, 7)
y train shape (160, 1)
y test shape  (40, 1)
```

In [33]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [34]:
```
tree_class = DecisionTreeClassifier(criterion = 'gini', max_depth = 4, splitter =
```

In [35]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_repo
```

In [36]:
```
tree_class.fit(X_train, y_train)
y_pred = tree_class.predict(X_test)
print(tree_class.score(X_train,y_train)*100)
tree_score = accuracy_score(y_test, y_pred)
print(tree_score*100)
```

```
100.0
100.0
```

In [37]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[17  0  0  0  0]
 [ 0  4  0  0  0]
 [ 0  0 13  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0  2]]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00        13
           4       1.00      1.00      1.00         4
           5       1.00      1.00      1.00         2

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

Interpretation:

Of the entire test set, 100% of the drugs were predicted correctly.

In [ ]:

ML LAB 7

Implement Naïve Bayes algorithm in a given business environment and comment on its efficiency and performance.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from warnings import filterwarnings
filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('C:/Users/user/Downloads/archive (2)/drug200.csv')
```

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

Out[3]:

	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

In [4]:
```
data.isnull().sum()
```

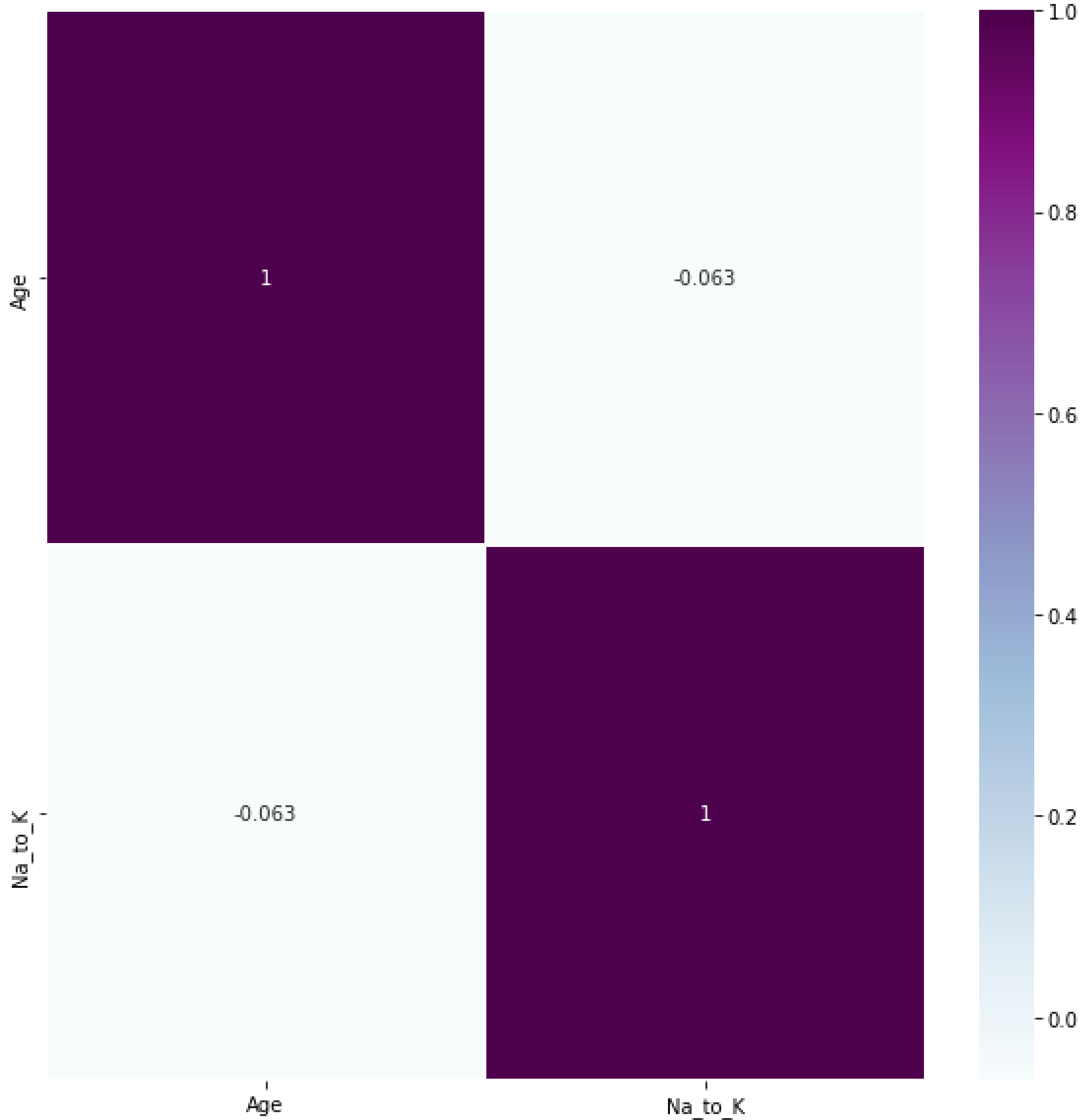
Out[4]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

In [5]:
```
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
```

there is no missing values in the date we have 6 coulmns and 200 rows

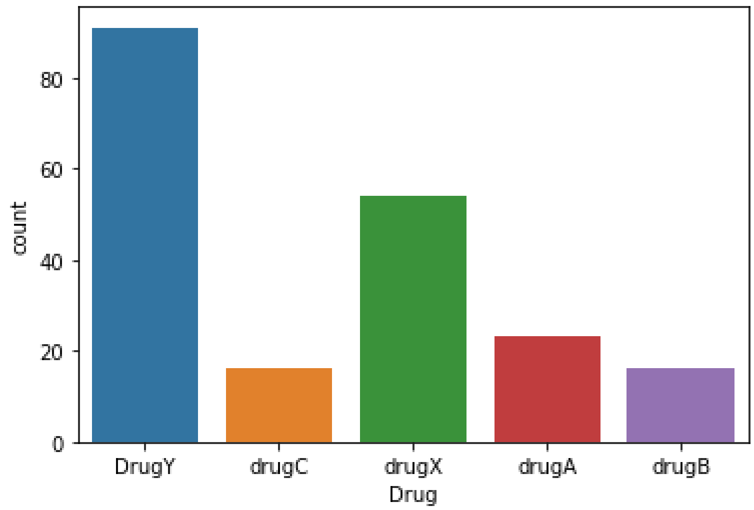
In [6]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```



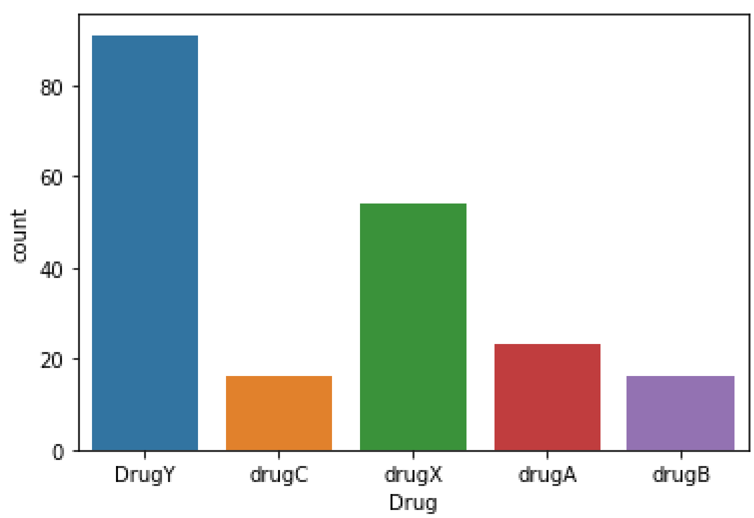
In [7]:
```
data['Drug'].value_counts()
```

Out[7]:
```
DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: Drug, dtype: int64
```

In [8]:
```
sns.countplot(x = 'Drug', data= data)
```
Out[8]: <AxesSubplot:xlabel='Drug', ylabel='count'>

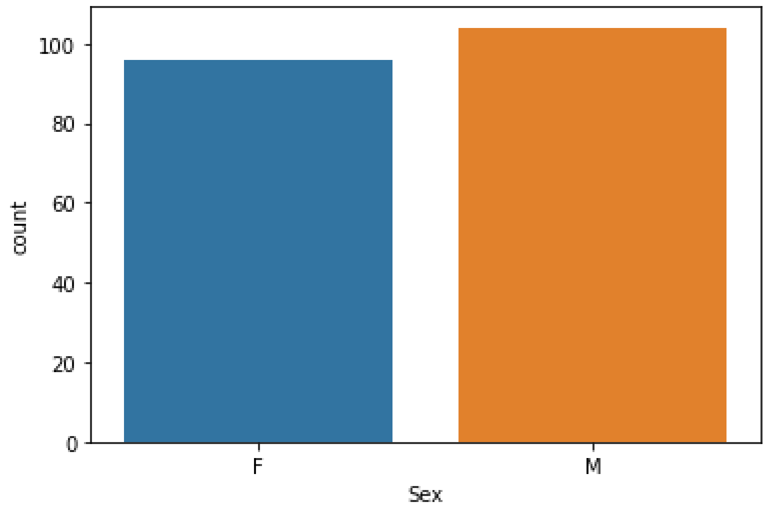


In [9]:
```
sns.countplot(x = 'Drug', data= data)
```
Out[9]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [10]:
```
sns.countplot(x = 'Sex', data= data)
```

Out[10]: <AxesSubplot:xlabel='Sex', ylabel='count'>

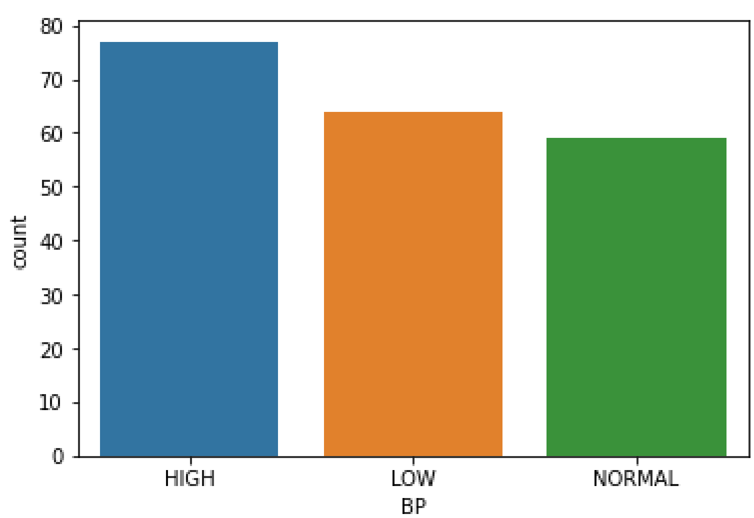


In [11]:
```
data['BP'].value_counts()
```

Out[11]:
```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

In [12]:
```
sns.countplot(x = 'BP', data= data)
```

Out[12]: <AxesSubplot:xlabel='BP', ylabel='count'>

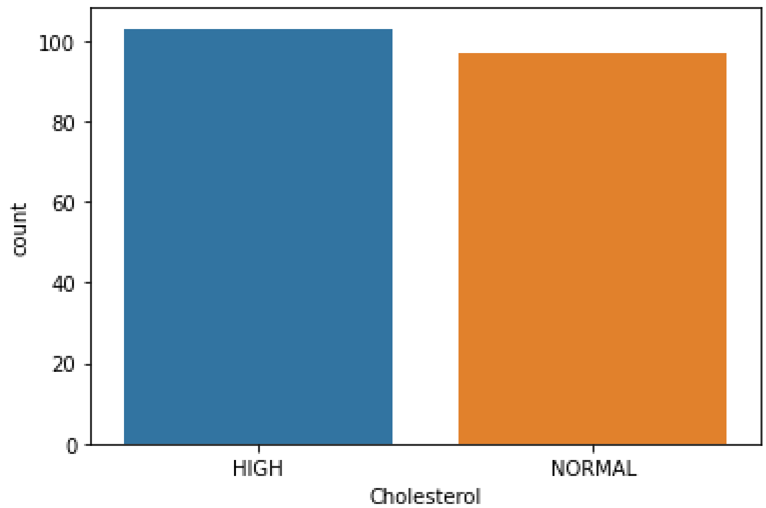


In [13]:
```
data['Cholesterol'].value_counts()
```

Out[13]:
```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

In [14]:
```
sns.countplot(x = 'Cholesterol', data= data)
```

Out[14]:
```
<AxesSubplot:xlabel='Cholesterol', ylabel='count'>
```

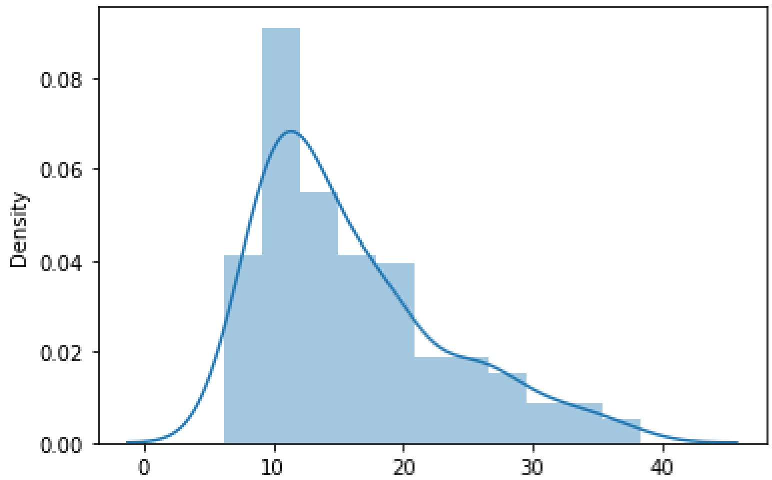


In [15]:
```
data['Na_to_K'].describe()
```

Out[15]:
```
count    200.000000
mean      16.084485
std        7.223956
min        6.269000
25%       10.445500
50%       13.936500
75%       19.380000
max       38.247000
Name: Na_to_K, dtype: float64
```

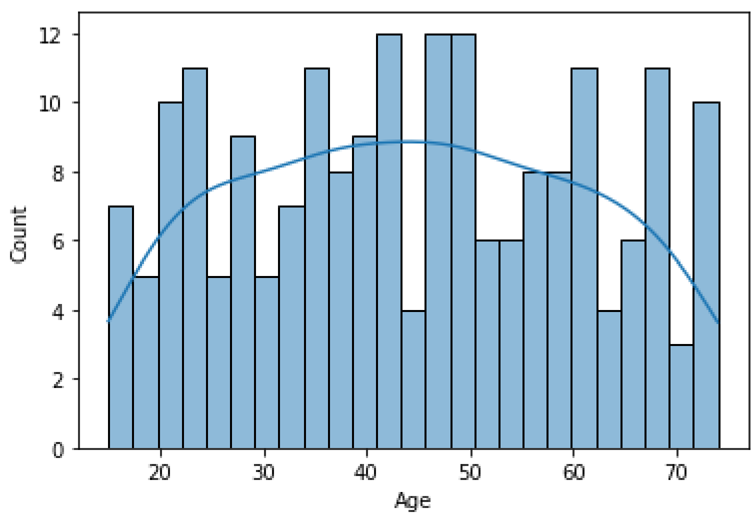
In [16]:
```
sns.distplot(x = data['Na_to_K'])
```

Out[16]: <AxesSubplot:ylabel='Density'>



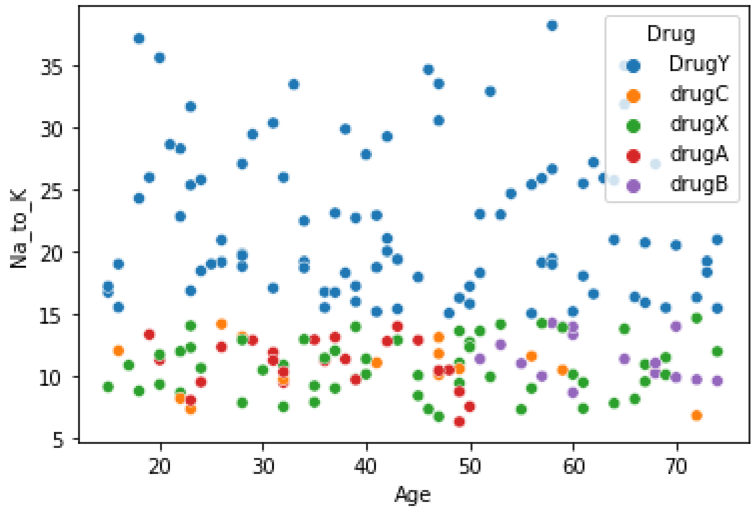
In [17]:
```
sns.histplot(x = 'Age', kde=True, bins = 25, data = data)
```

Out[17]: <AxesSubplot:xlabel='Age', ylabel='Count'>



In [18]:
```
sns.scatterplot(x = 'Age', y = 'Na_to_K', data = data, hue = 'Drug')
```

Out[18]:
```
<AxesSubplot:xlabel='Age', ylabel='Na_to_K'>
```



In the last fig we find all the items have more than 15 Na_to_K have DrugY type

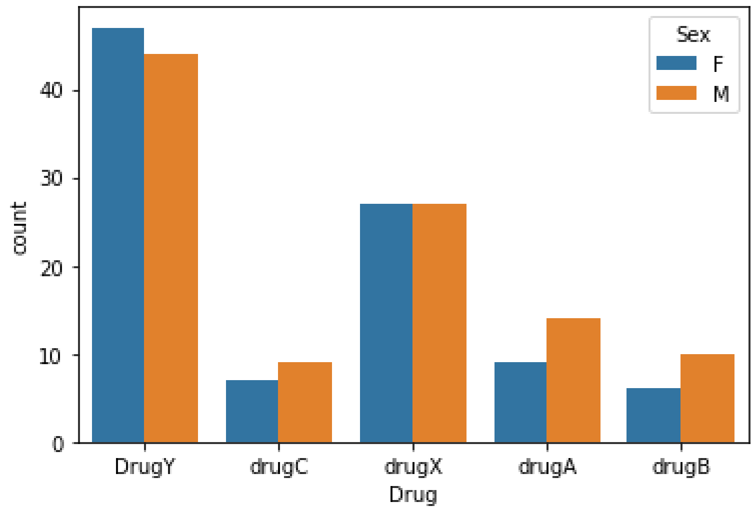
In the next We will find out the number of each Drug type per Sex

In [19]:
```
data_sex_drug = data.groupby(['Drug','Sex']).size().reset_index(name = 'count')
print(data_sex_drug)
```

```
    Drug Sex  count
0  DrugY   F     47
1  DrugY   M     44
2  drugA   F      9
3  drugA   M     14
4  drugB   F      6
5  drugB   M     10
6  drugC   F      7
7  drugC   M      9
8  drugX   F     27
9  drugX   M     27
```

In [20]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Sex')
```

Out[20]:
```
<AxesSubplot:xlabel='Drug', ylabel='count'>
```

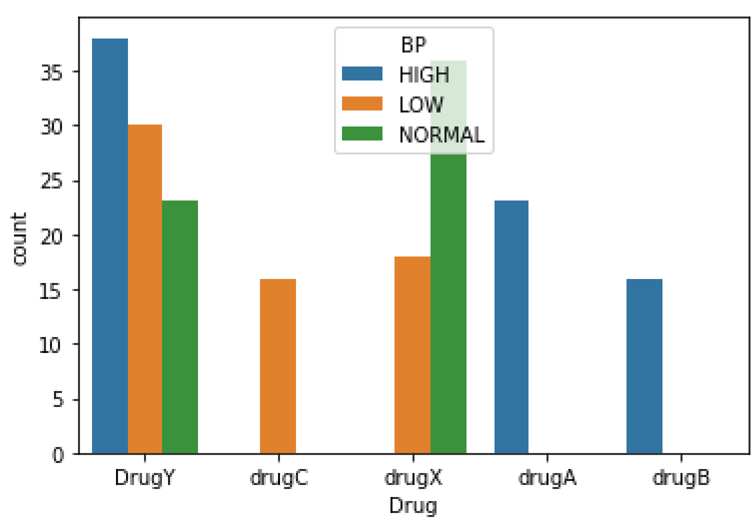


In [21]:
```
data_BP_drug = data.groupby(['Drug','BP']).size().reset_index(name = 'count')
print(data_BP_drug)
```

```
    Drug      BP  count
0  DrugY    HIGH     38
1  DrugY     LOW     30
2  DrugY  NORMAL     23
3  drugA    HIGH     23
4  drugB    HIGH     16
5  drugC     LOW     16
6  drugX     LOW     18
7  drugX  NORMAL     36
```

In [22]:
```
sns.countplot(x = 'Drug', data= data, hue = 'BP')
```

Out[22]:
```
<AxesSubplot:xlabel='Drug', ylabel='count'>
```

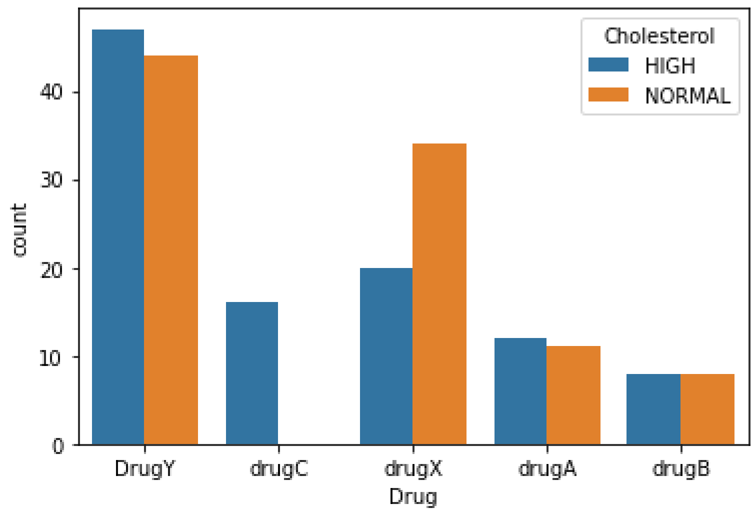


In [23]:
```
data_Cholesterol_drug = data.groupby(['Drug','Cholesterol']).size().reset_index(n
print(data_Cholesterol_drug)
```

```
    Drug Cholesterol  count
0  DrugY        HIGH     47
1  DrugY      NORMAL     44
2  drugA        HIGH     12
3  drugA      NORMAL     11
4  drugB        HIGH      8
5  drugB      NORMAL      8
6  drugC        HIGH     16
7  drugX        HIGH     20
8  drugX      NORMAL     34
```

In [24]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Cholesterol')
```

Out[24]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [25]:
```
data['Sex'] = data['Sex'].map({'M': 1, 'F': 0})
data['Cholesterol'] = data['Cholesterol'].map({'HIGH' : 1, 'NORMAL' : 0})
data['Drug'] = data['Drug'].map({'DrugY':1, 'drugC':2, 'drugX':3, 'drugA':4, 'dru
data.head()
```

Out[25]:

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

In [26]:
```
data.shape
```

Out[26]: (200, 6)

In [27]:
```
data = pd.get_dummies(data)
data.head()
```

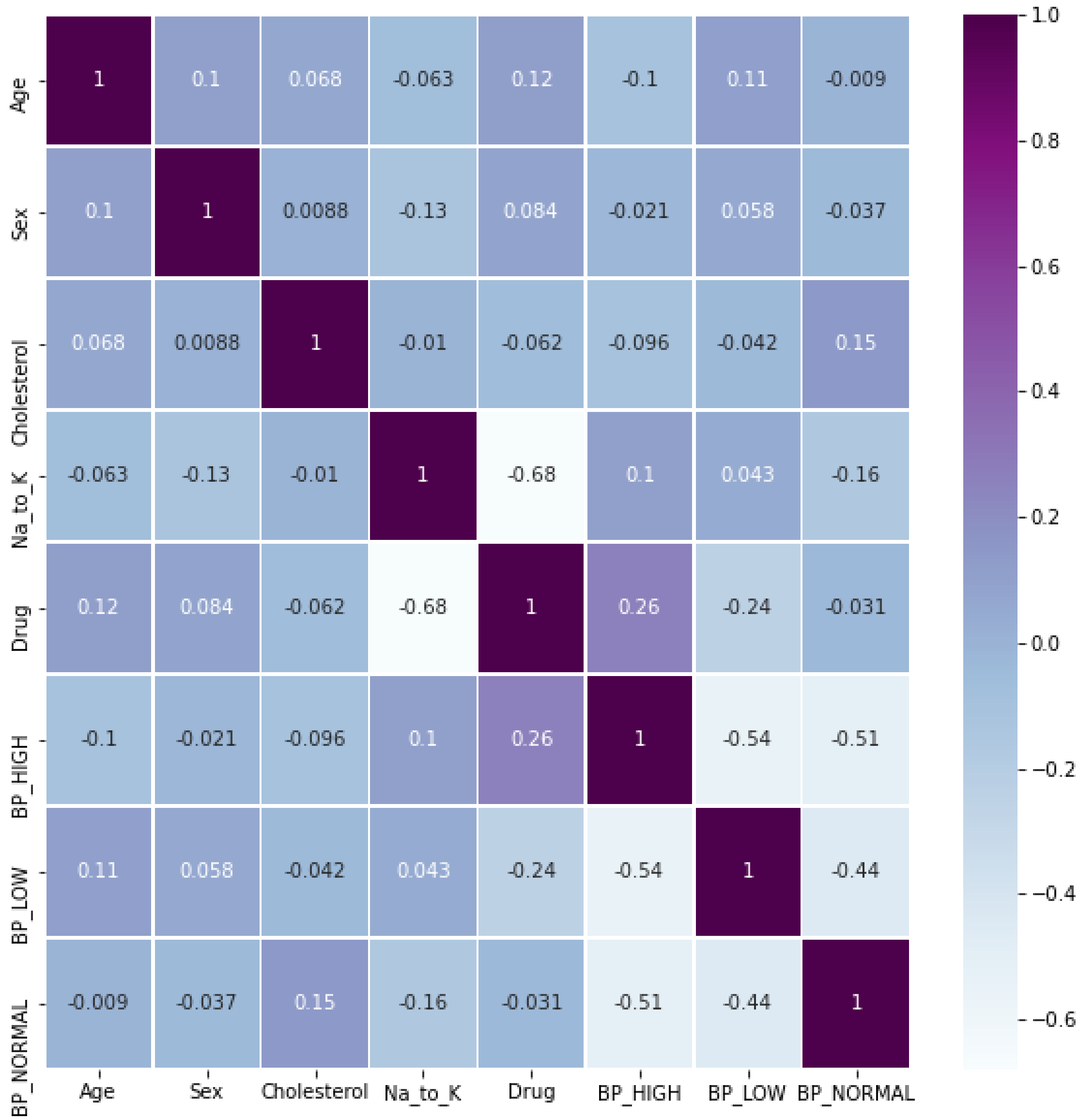
Out[27]:

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

In [28]:
```
data.shape
```

Out[28]: (200, 8)

In [29]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```



In [30]:
```
X = data.drop('Drug', axis = 1).values
y = data['Drug'].values.reshape((-1,1))
```

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random
print('x train shape {}'.format(X_train.shape))
print('x test shape  {}'.format(X_test.shape))
print('y train shape {}'.format(y_train.shape))
print('y test shape  {}'.format(y_test.shape))
```

```
x train shape (160, 7)
x test shape  (40, 7)
y train shape (160, 1)
y test shape  (40, 1)
```

In [33]:
```
from sklearn.naive_bayes import GaussianNB
```

In [34]:
```
# classificador Logreg
GNB = GaussianNB()

# Fitting with train data
model = GNB.fit(X_train, y_train)
```

In [37]:
```
# Printing the Training Score
print("Training score data: ")
print(model.score(X_train, y_train))
```

```
Training score data:
0.7625
```

In [38]:
```
y_pred = model.predict(X_test)

print('\nAccuracy of Naive Bayes classifier on test set: {:.2f}'.format(accuracy_
```

```

Accuracy of Naive Bayes classifier on test set: 0.75
```

In [39]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_repo
```

In [41]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 7  4  4  2  0]
 [ 0  4  0  0  0]
 [ 0  0 13  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0  2]]
              precision    recall  f1-score   support

           1       1.00      0.41      0.58        17
           2       0.50      1.00      0.67         4
           3       0.76      1.00      0.87        13
           4       0.67      1.00      0.80         4
           5       1.00      1.00      1.00         2

    accuracy                           0.75        40
   macro avg       0.79      0.88      0.78        40
weighted avg       0.84      0.75      0.73        40
```

Interpretation:

Of the entire test set, 84% of the drugs were predicted correctly.

In [ ]:

ML LAB 8

Implement K Nearest Neighbors algorithm in a given business environment and comment on its efficiency and performance.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from warnings import filterwarnings
filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('C:/Users/user/Downloads/archive (2)/drug200.csv')
```

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

Out[3]:

	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

In [4]:
```
data.isnull().sum()
```

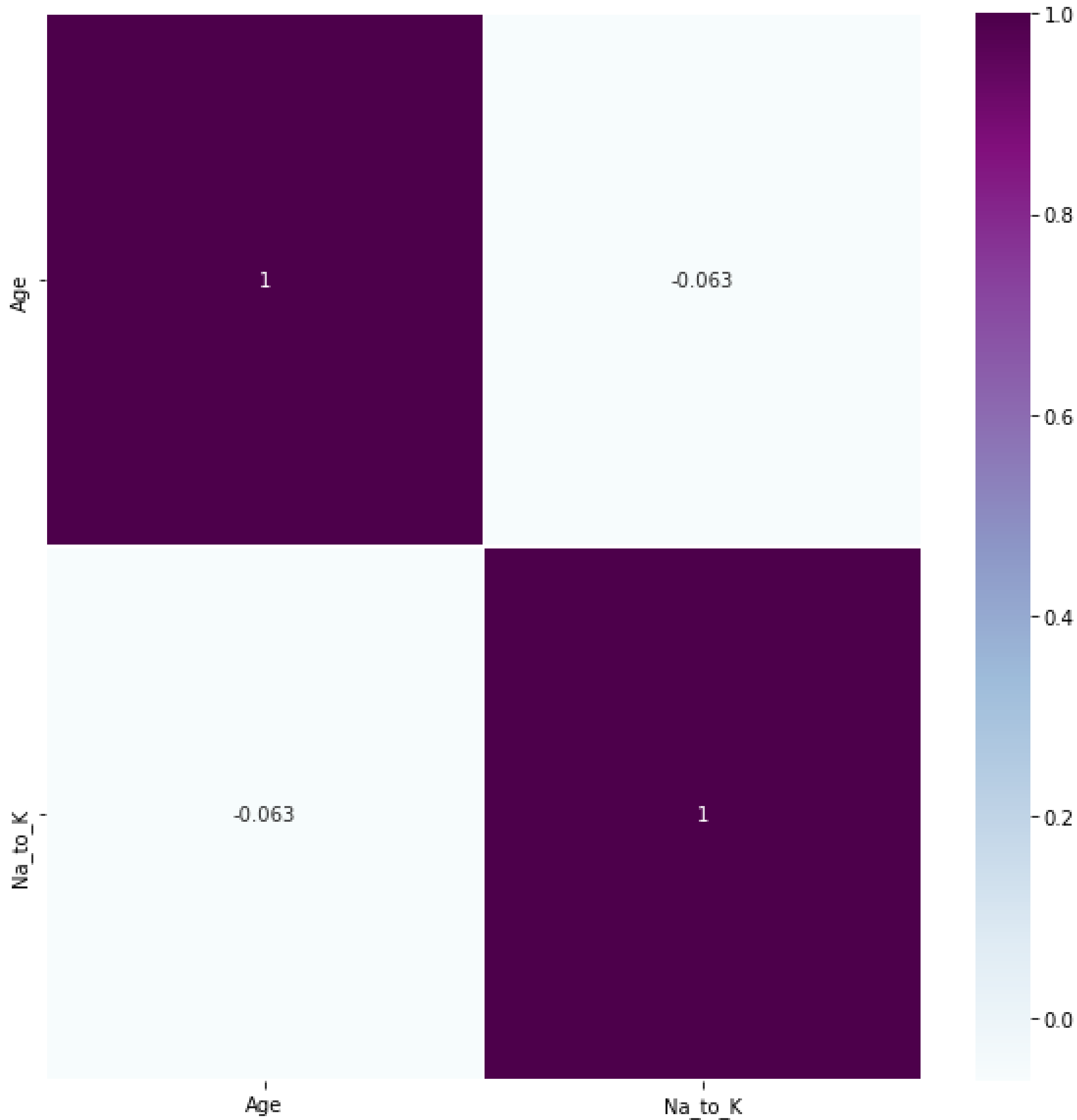
Out[4]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

In [5]:
```
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
```

there is no missing values in the date we have 6 coulmns and 200 rows

In [6]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```

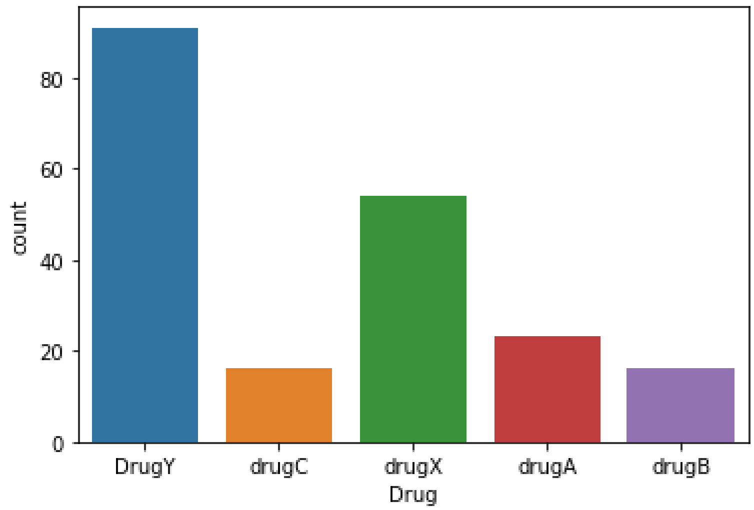


In [7]:
```
data['Drug'].value_counts()
```

Out[7]:
```
DrugY    91
drugX    54
drugA    23
drugB    16
drugC    16
Name: Drug, dtype: int64
```

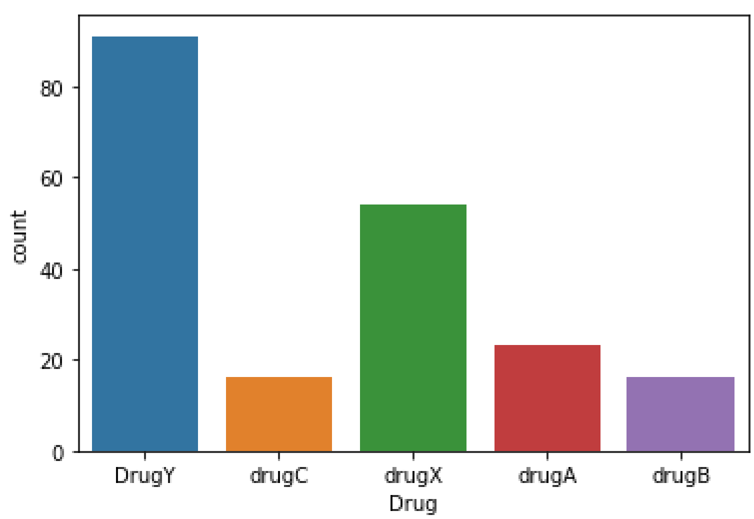
In [8]:
```
sns.countplot(x = 'Drug', data= data)
```

Out[8]: <AxesSubplot:xlabel='Drug', ylabel='count'>



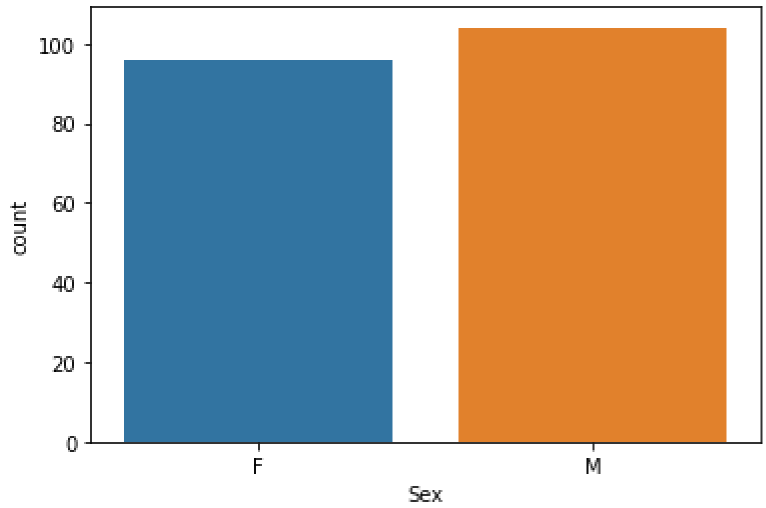
In [9]:
```
sns.countplot(x = 'Drug', data= data)
```

Out[9]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [10]:
```
sns.countplot(x = 'Sex', data= data)
```

Out[10]: <AxesSubplot:xlabel='Sex', ylabel='count'>

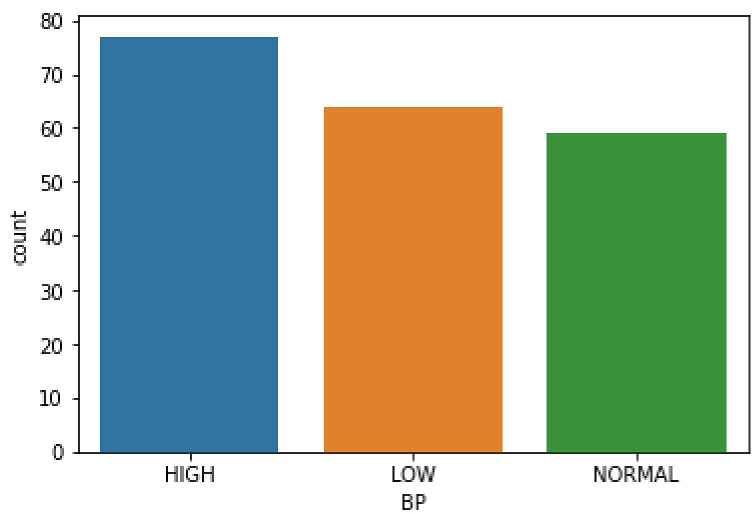


In [11]:
```
data['BP'].value_counts()
```

Out[11]:
```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

In [12]:
```
sns.countplot(x = 'BP', data= data)
```

Out[12]: <AxesSubplot:xlabel='BP', ylabel='count'>

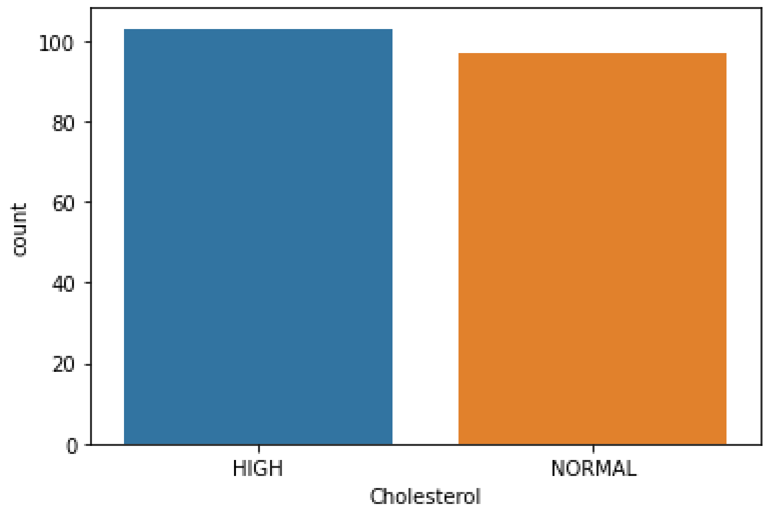


In [13]:
```
data['Cholesterol'].value_counts()
```

Out[13]:
```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

In [14]:
```
sns.countplot(x = 'Cholesterol', data= data)
```

Out[14]:
```
<AxesSubplot:xlabel='Cholesterol', ylabel='count'>
```

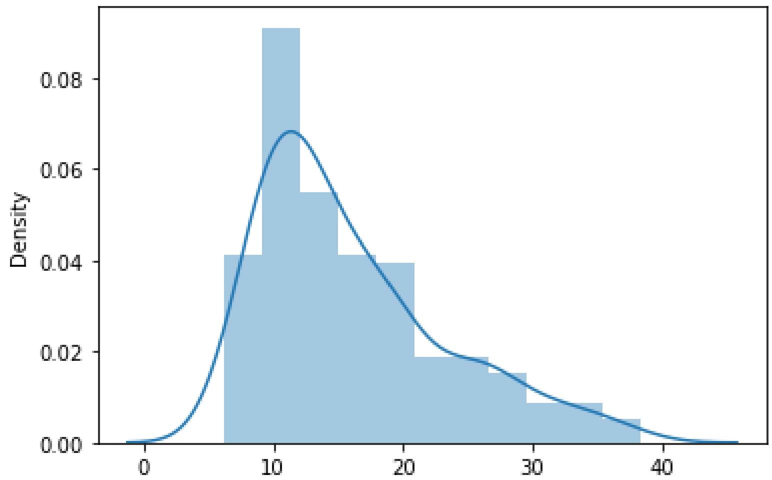


In [15]:
```
data['Na_to_K'].describe()
```

Out[15]:
```
count    200.000000
mean      16.084485
std        7.223956
min        6.269000
25%       10.445500
50%       13.936500
75%       19.380000
max       38.247000
Name: Na_to_K, dtype: float64
```

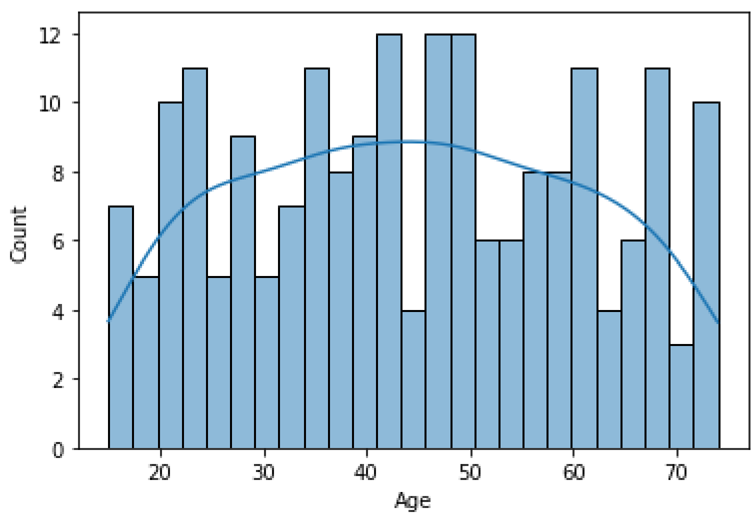
In [16]:
```
sns.distplot(x = data['Na_to_K'])
```

Out[16]: <AxesSubplot:ylabel='Density'>



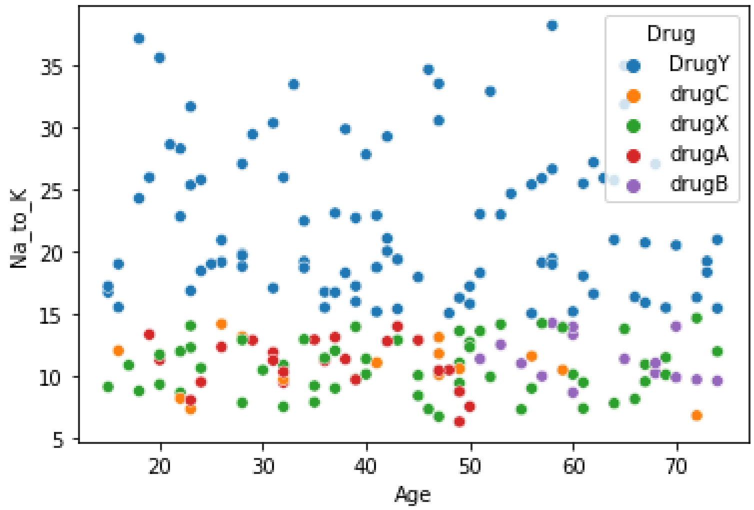
In [17]:
```
sns.histplot(x = 'Age', kde=True, bins = 25, data = data)
```

Out[17]: <AxesSubplot:xlabel='Age', ylabel='Count'>



In [18]:
```
sns.scatterplot(x = 'Age', y = 'Na_to_K', data = data, hue = 'Drug')
```

Out[18]:
```
<AxesSubplot:xlabel='Age', ylabel='Na_to_K'>
```



In the last fig we find all the items have more than 15 Na_to_K have DrugY type

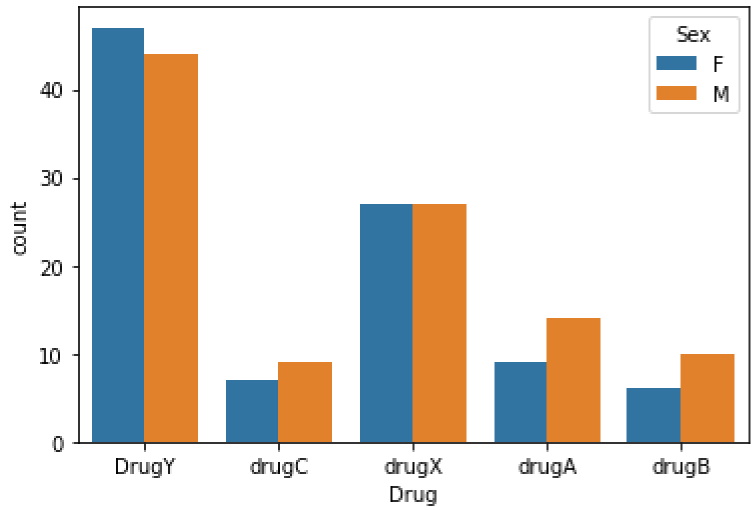
In the next We will find out the number of each Drug type per Sex

In [19]:
```
data_sex_drug = data.groupby(['Drug','Sex']).size().reset_index(name = 'count')
print(data_sex_drug)
```

```
    Drug Sex  count
0  DrugY   F     47
1  DrugY   M     44
2  drugA   F      9
3  drugA   M     14
4  drugB   F      6
5  drugB   M     10
6  drugC   F      7
7  drugC   M      9
8  drugX   F     27
9  drugX   M     27
```

In [20]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Sex')
```

Out[20]:
```
<AxesSubplot:xlabel='Drug', ylabel='count'>
```

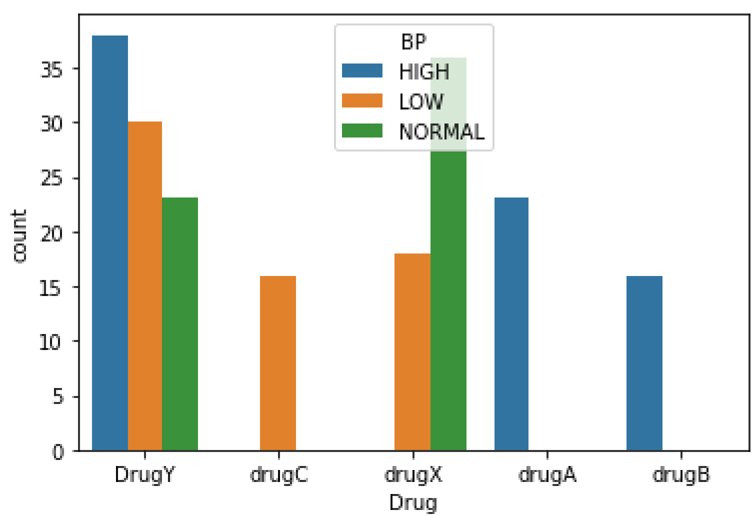


In [21]:
```
data_BP_drug = data.groupby(['Drug','BP']).size().reset_index(name = 'count')
print(data_BP_drug)
```

```
    Drug      BP  count
0  DrugY    HIGH     38
1  DrugY     LOW     30
2  DrugY  NORMAL     23
3  drugA    HIGH     23
4  drugB    HIGH     16
5  drugC     LOW     16
6  drugX     LOW     18
7  drugX  NORMAL     36
```

In [22]:
```
sns.countplot(x = 'Drug', data= data, hue = 'BP')
```

Out[22]:
```
<AxesSubplot:xlabel='Drug', ylabel='count'>
```

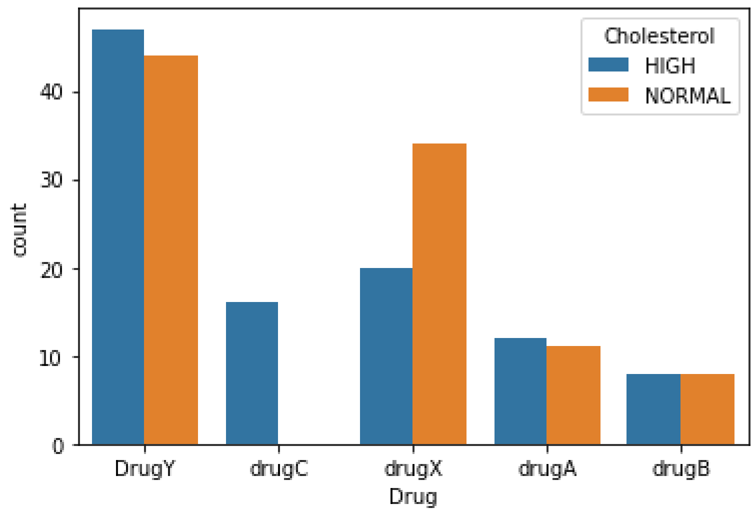


In [23]:
```
data_Cholesterol_drug = data.groupby(['Drug','Cholesterol']).size().reset_index(n
print(data_Cholesterol_drug)
```

```
    Drug Cholesterol  count
0  DrugY        HIGH     47
1  DrugY      NORMAL     44
2  drugA        HIGH     12
3  drugA      NORMAL     11
4  drugB        HIGH      8
5  drugB      NORMAL      8
6  drugC        HIGH     16
7  drugX        HIGH     20
8  drugX      NORMAL     34
```

In [24]:
```
sns.countplot(x = 'Drug', data= data, hue = 'Cholesterol')
```

Out[24]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [25]:
```
data['Sex'] = data['Sex'].map({'M': 1, 'F': 0})
data['Cholesterol'] = data['Cholesterol'].map({'HIGH' : 1, 'NORMAL' : 0})
data['Drug'] = data['Drug'].map({'DrugY':1, 'drugC':2, 'drugX':3, 'drugA':4, 'dru
data.head()
```

Out[25]:

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

In [26]:
```
data.shape
```

Out[26]: (200, 6)

In [27]:
```
data = pd.get_dummies(data)
data.head()
```

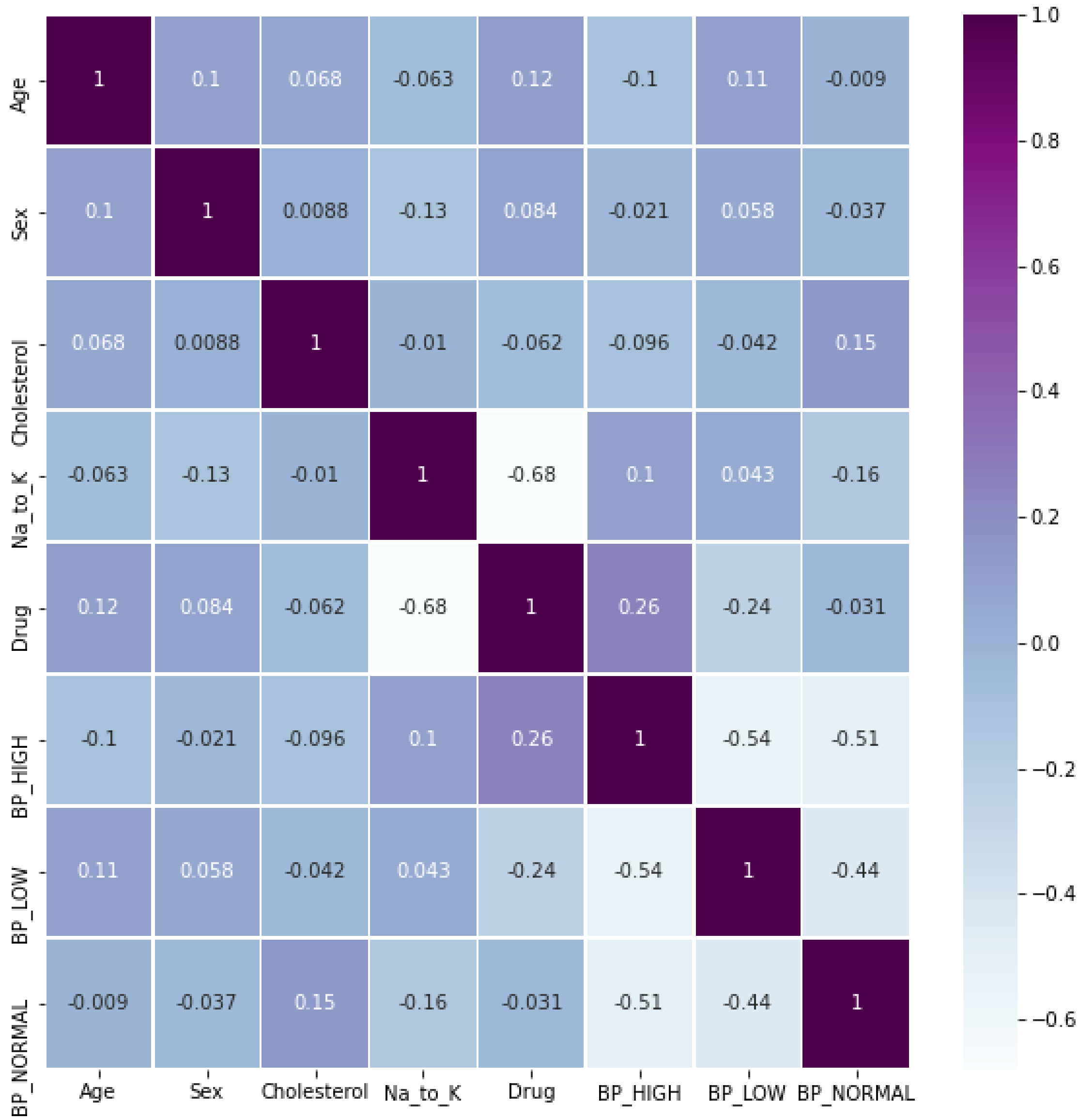
Out[27]:

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

In [28]:
```
data.shape
```

Out[28]: (200, 8)

In [29]:
```
fig, ax = plt.subplots(figsize  = (10, 10))
sns.heatmap(data.corr(), cmap = 'BuPu', cbar = True, linewidth = 0.5, annot = Tru
plt.show()
```



In [30]:
```
X = data.drop('Drug', axis = 1).values
y = data['Drug'].values.reshape((-1,1))
```

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random
print('x train shape {}'.format(X_train.shape))
print('x test shape  {}'.format(X_test.shape))
print('y train shape {}'.format(y_train.shape))
print('y test shape  {}'.format(y_test.shape))
```

```
x train shape (160, 7)
x test shape  (40, 7)
y train shape (160, 1)
y test shape  (40, 1)
```

In [33]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [34]:
```
KNN_class = KNeighborsClassifier(n_neighbors = 3)
```

In [35]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_repo
```

In [37]:
```
KNN_class.fit(X_train, y_train)
y_pred = KNN_class.predict(X_test)
print(KNN_class.score(X_train,y_train)*100)
KNN_score = accuracy_score(y_test, y_pred)
print(KNN_score*100)
```

```
83.75
70.0
```

In [38]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[16  0  0  0  1]
 [ 0  2  0  0  2]
 [ 0  3  6  2  2]
 [ 1  1  0  2  0]
 [ 0  0  0  0  2]]
              precision    recall  f1-score   support

           1       0.94      0.94      0.94        17
           2       0.33      0.50      0.40         4
           3       1.00      0.46      0.63        13
           4       0.50      0.50      0.50         4
           5       0.29      1.00      0.44         2

    accuracy                           0.70        40
   macro avg       0.61      0.68      0.58        40
weighted avg       0.82      0.70      0.72        40
```

Interpretation:

Of the entire test set, 82% of the drugs were predicted correctly.

In [ ]:

ML LAB 9

Implement Support Vector Machine algorithm for classification in a given business environment and comment on its efficiency and performance.

Support Vector Machine Tutorial Using Python Sklearn

In [1]:
```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

In [2]:
```
iris.feature_names
```

Out[2]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [3]:
```
iris.target_names
```

Out[3]:
```
array(['setosa', 'versicolor', 'virginica'],
      dtype='<U10')
```

In [6]:
```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Out[6]:

	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 [8]:
```
df['target'] = iris.target
df.head()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [9]:
```
df[df.target==1].head()
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

In [10]:
```
df[df.target==2].head()
```

Out[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

In [11]:
```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [13]:
```
df[45:55]
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

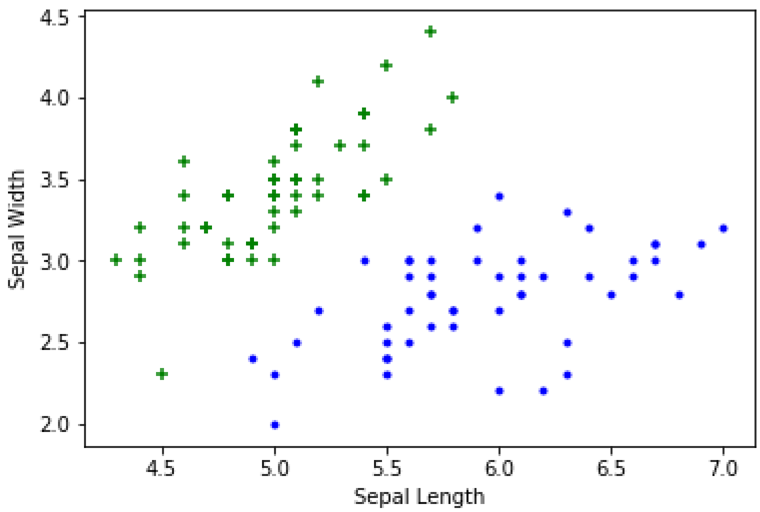
In [15]:
```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

In [14]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Sepal length vs Sepal Width (Setosa vs Versicolor)

In [17]:
```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marke
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker
```

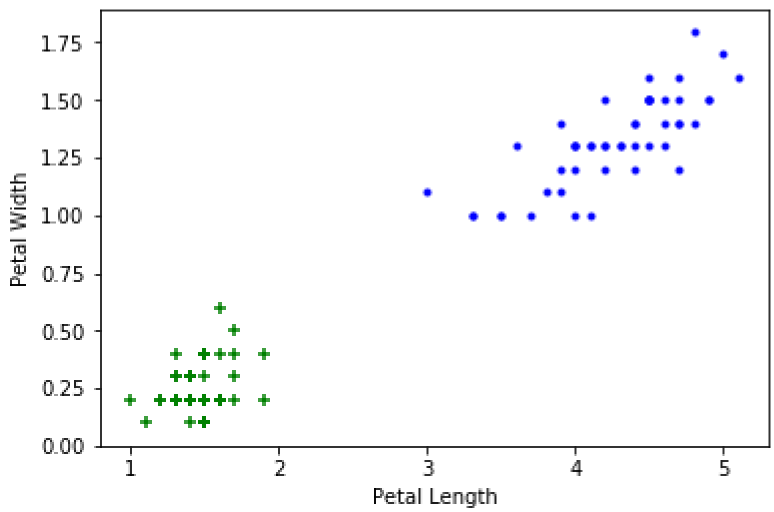
Out[17]: <matplotlib.collections.PathCollection at 0x1f1b16976a0>



Petal length vs Pepal Width (Setosa vs Versicolor)

In [18]:
```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marke
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker
```

Out[18]: <matplotlib.collections.PathCollection at 0x1f1b2018390>



Train Using Support Vector Machine (SVM)

In [49]:
```
from sklearn.model_selection import train_test_split
```

In [50]:
```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [52]:
```
len(X_train)
```

Out[52]: 120

In [53]:
```
len(X_test)
```

Out[53]: 30

In [75]:
```
from sklearn.svm import SVC
model = SVC()
```

In [76]:
```
model.fit(X_train, y_train)
```

Out[76]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [77]:
```
model.score(X_test, y_test)
```

Out[77]: 0.9333333333333335

In [78]:
```
model.predict([[4.8,3.0,1.5,0.3]])
```
Out[78]:
```
array([0])
```

Tune parameters

1. Regularization (C)

In [97]:
```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```
Out[97]:
```
0.9333333333333335
```

In [106]:
```
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```
Out[106]:
```
0.9666666666666667
```

2. Gamma

In [103]:
```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```
Out[103]:
```
0.9000000000000002
```

3. Kernel

In [104]:
```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
```
Out[104]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [105]:
```
model_linear_kernal.score(X_test, y_test)
```
Out[105]:
```
0.9666666666666667
```

Exercise

Train SVM classifier using sklearn digits dataset (i.e. from sklearn.datasets import load_digits) and then,

1. Measure accuracy of your model using different kernels such as rbf and linear.
2. Tune your model further using regularization and gamma parameters and try to come up with highest accurancy score
3. Use 80% of samples as training data size

ML LAB 10

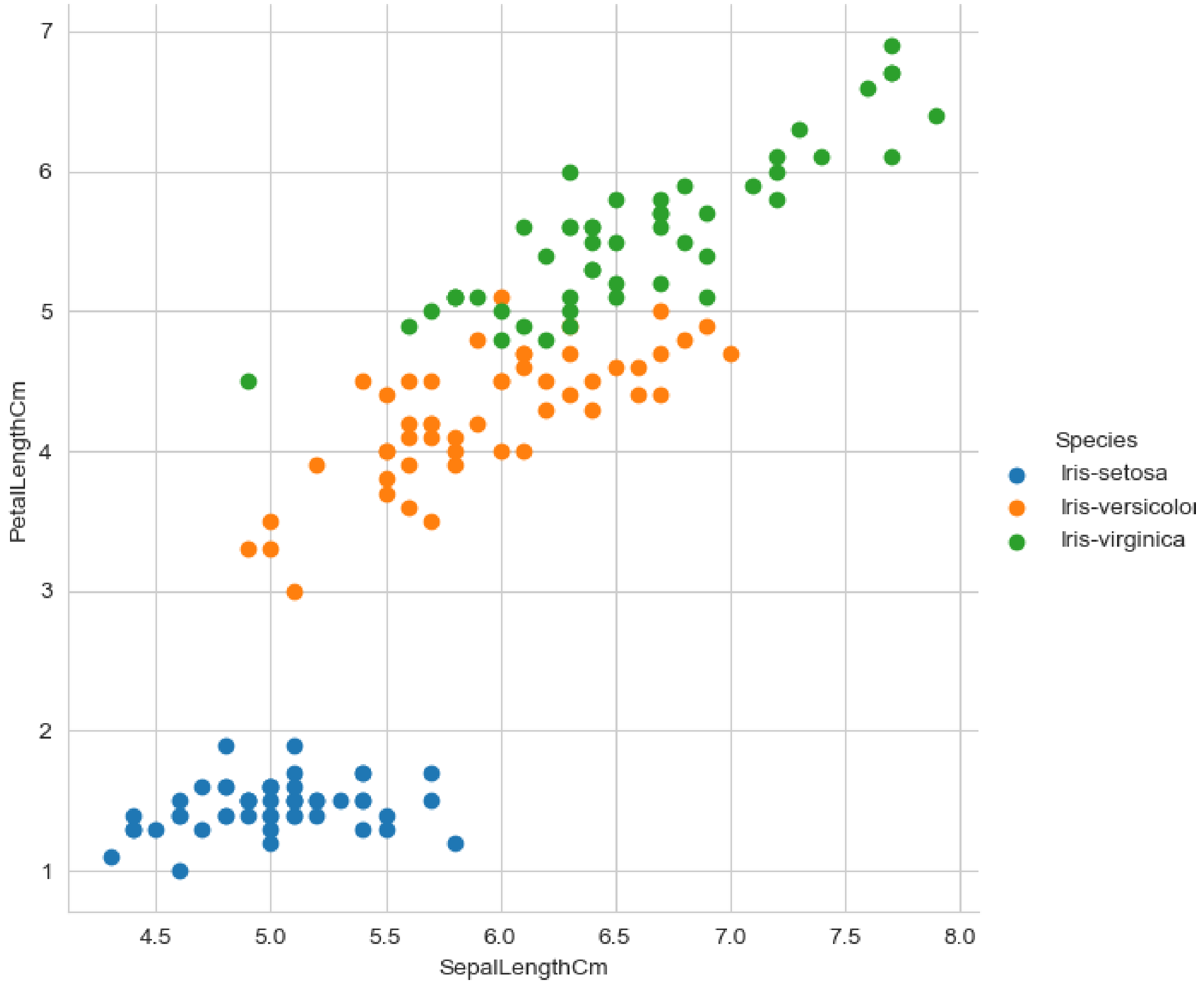
Implement Principal Component Analysis for dimensionality reduction in a given business environment and comment on its efficiency and performance.

In [16]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
iris = pd.read_csv("Iris.csv")
df=pd.DataFrame(iris)
df.head()
x=df.drop(['Id','Species'],axis=1)
print(x.head())
y=df.Species
print(y.head())
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
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
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: Species, dtype: object
```

In [17]:
```
import seaborn as sns
sns.set_style("whitegrid")
sns.FacetGrid(iris,hue='Species',height=6).map(plt.scatter,'SepalLengthCm','Petal
plt.show()
```



In [35]:
```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(x)
print(X[:5])
type(X)
print(X.shape[0])
```

```
[[-0.90068117  1.03205722 -1.3412724  -1.31297673]
 [-1.14301691 -0.1249576  -1.3412724  -1.31297673]
 [-1.38535265  0.33784833 -1.39813811 -1.31297673]
 [-1.50652052  0.10644536 -1.2844067  -1.31297673]
 [-1.02184904  1.26346019 -1.3412724  -1.31297673]]
150
```

In [36]:
```
X_mean = np.mean(X, axis=0)
print(X_mean)
# cov_mat = np.cov(X)
cov_mat = (X - X_mean).T.dot((X - X_mean)) / (X.shape[0])
print('Covariance matrix \n%s' %cov_mat)
```

```
[-4.73695157e-16 -6.63173220e-16  3.31586610e-16 -2.84217094e-16]
Covariance matrix
[[ 1.         -0.10936925  0.87175416  0.81795363]
 [-0.10936925  1.         -0.4205161  -0.35654409]
 [ 0.87175416 -0.4205161   1.          0.9627571 ]
 [ 0.81795363 -0.35654409  0.9627571   1.        ]]
```

In [37]:
```
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

Eigenvalues
[2.91081808 0.92122093 0.14735328 0.02060771]
```

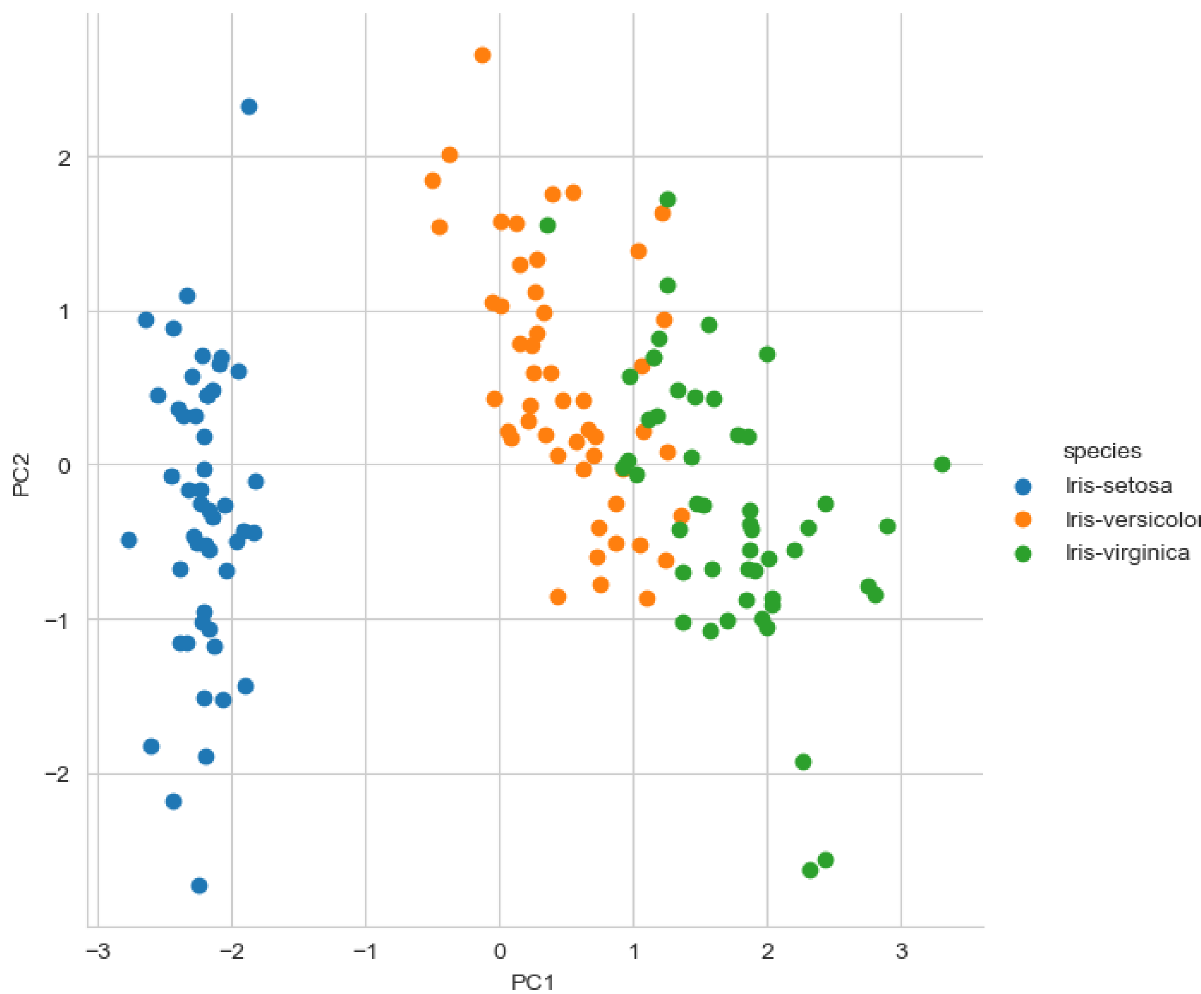
In [43]:
```
pc1=X.dot(eig_vecs.T[0])
pc2=X.dot(eig_vecs.T[1])
result = pd.DataFrame(pc1,columns=['PC1'])
result['PC2']=pc2
result['species']=y
result.head()
```

Out[43]:

	PC1	PC2	species
0	-2.264542	-0.505704	Iris-setosa
1	-2.086426	0.655405	Iris-setosa
2	-2.367950	0.318477	Iris-setosa
3	-2.304197	0.575368	Iris-setosa
4	-2.388777	-0.674767	Iris-setosa

In [64]:
```
plt.figure(figsize=(30,10))
sns.FacetGrid(result,hue='species',height=6).map(plt.scatter,'PC1','PC2').add_leg
plt.show()
```

```
<Figure size 3000x1000 with 0 Axes>
```



ML LAB 11

Perform Time Series Analysis in a given business environment exploring Horizontal Pattern, Trend Pattern, Seasonal Pattern, and moving averages and comment on Forecasting accuracy.

Time Series Analysis and forecasting using ARIMA

What is a time series problem

In the field for machine learning and data science, most of the real-life problems are based upon the prediction of future which is totally oblivious to us such as stock market prediction, future sales prediction and so on.Time series problem is basically the prediction of such problems using various machine learning tools.Time series problem is tackled efficiently when first it is analyzed properly (Time Series Analysis) and according to that observation suitable algorithm is used (Time Series Forecasting).

Objective(Business Scenario):

```
Forecast time series data using ARIMA
```

Librarys

Importing Librarys

In [1]:

```
# Load required Libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #to plot some parameters in seaborn
from sklearn.linear_model import LinearRegression # To work on Linear Regression
from sklearn.metrics import r2_score # To Calculate Performance matrix
import statsmodels.api as sm # To calculatestats modle
import seaborn as sns
```

Importing Dataset

In [82]:
```
# Reading the data
df = pd.read_csv('DataFrames/Electric_Production.csv')
```

In [7]:
```
# A glance on the data
df.head()
```

Out[7]:

	DATE	Value
0	01-01-1985	72.5052
1	02-01-1985	70.6720
2	03-01-1985	62.4502
3	04-01-1985	57.4714
4	05-01-1985	55.3151

In [8]:
```
# getting some information about dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   DATE    397 non-null    object 
 1   Value   397 non-null    float64
dtypes: float64(1), object(1)
memory usage: 6.3+ KB
```

From this you can infer two necessary things:

1. You really need to change change columns name
2. Both the columns have object datatype

In [9]:
```
# further Analysis
df.describe()
```

Out[9]:

	Value
count	397.000000
mean	88.847218
std	15.387834
min	55.315100
25%	77.105200
50%	89.779500
75%	100.524400
max	129.404800

In [10]:
```
df.columns = ["DATE", "value"]
df.head()
```

Out[10]:

	DATE	value
0	01-01-1985	72.5052
1	02-01-1985	70.6720
2	03-01-1985	62.4502
3	04-01-1985	57.4714
4	05-01-1985	55.3151

In [11]:
```
df.dtypes
```

Out[11]:
```
DATE      object
value    float64
dtype: object
```

In [15]:
```
df['value'].unique()
```

Out[15]:
```
array([ 72.5052,  70.672 ,  62.4502,  57.4714,  55.3151,  58.0904,
        62.6202,  63.2485,  60.5846,  56.3154,  58.0005,  68.7145,
        73.3057,  67.9869,  62.2221,  57.0329,  55.8137,  59.9005,
        65.7655,  64.4816,  61.0005,  57.5322,  59.3417,  68.1354,
        73.8152,  70.062 ,  65.61  ,  60.1586,  58.8734,  63.8918,
        68.8694,  70.0669,  64.1151,  60.3789,  62.4643,  70.5777,
        79.8703,  76.1622,  70.2928,  63.2384,  61.4065,  67.1097,
        72.9816,  75.7655,  67.5152,  63.2832,  65.1078,  73.8631,
        77.9188,  76.6822,  73.3523,  65.1081,  63.6892,  68.4722,
        74.0301,  75.0448,  69.3053,  65.8735,  69.0706,  84.1949,
        84.3598,  77.1726,  73.1964,  67.2781,  65.8218,  71.4654,
        76.614 ,  77.1052,  73.061 ,  67.4365,  68.5665,  77.6839,
        86.0214,  77.5573,  73.365 ,  67.15  ,  68.8162,  74.8448,
        80.0928,  79.1606,  73.5743,  68.7538,  72.5166,  79.4894,
        85.2855,  80.1643,  74.5275,  69.6441,  67.1784,  71.2078,
        77.5081,  76.5374,  72.3541,  69.0286,  73.4992,  84.5159,
        87.9464,  84.5561,  79.4747,  71.0578,  67.6762,  74.3297,
        82.1048,  82.0605,  74.6031,  69.681 ,  74.4292,  84.2284,
        94.1386,  87.1607,  79.2456,  70.9749,  69.3844,  77.9831,
        83.277 ,  81.8872,  75.6826,  71.2661,  75.2458,  84.8147,
        92.4532,  87.4033,  81.2661,  73.8167,  73.2682,  78.3026,
        85.9841,  89.5467,  78.5035,  73.7066,  79.6543,  90.8251,
        98.9732,  92.8883,  86.9356,  77.2214,  76.6826,  81.9306,
        85.9606,  86.5562,  79.1919,  74.6891,  81.074 ,  90.4855,
        98.4613,  89.7795,  83.0125,  76.1476,  73.8471,  79.7645,
        88.4519,  87.7828,  81.9386,  77.5027,  82.0448,  92.101 ,
        94.792 ,  87.82  ,  86.5549,  76.7521,  78.0303,  86.4579,
        93.8379,  93.531 ,  87.5414,  80.0924,  81.4349,  91.6841,
       102.1348,  91.1829,  90.7381,  80.5176,  79.3887,  87.8431,
        97.4903,  96.4157,  87.2248,  80.6409,  82.2025,  94.5113,
       102.2301,  94.2989,  88.0927,  81.4425,  84.4552,  91.0406,
        95.9957,  99.3704,  90.9178,  83.1408,  88.041 , 102.4558,
       109.1081,  97.1717,  92.8283,  82.915 ,  82.5465,  90.3955,
        96.074 ,  99.5534,  88.281 ,  82.686 ,  82.9319,  93.0381,
       102.9955,  95.2075,  93.2556,  85.795 ,  85.2351,  93.1896,
       102.393 , 101.6293,  93.3089,  86.9002,  88.5749, 100.8003,
       110.1807, 103.8413,  94.5532,  85.062 ,  85.4653,  91.0761,
       102.22  , 104.4682,  92.9135,  86.5047,  88.5735, 103.5428,
       113.7226, 106.159 ,  95.4029,  86.7233,  89.0302,  95.5045,
       101.7948, 100.2025,  94.024 ,  87.5262,  89.6144, 105.7263,
       111.1614, 101.7795,  98.9565,  86.4776,  87.2234,  99.5076,
       108.3501, 109.4862,  99.1155,  89.7567,  90.4587, 108.2257,
       104.4724, 101.5196,  98.4017,  87.5093,  90.0222, 100.5244,
       110.9503, 111.5192,  95.7632,  90.3738,  92.3566, 103.066 ,
       112.0576, 111.8399,  99.1925,  90.8177,  92.0587, 100.9676,
       107.5686, 114.1036, 101.5316,  93.0068,  93.9126, 106.7528,
       114.8331, 108.2353, 100.4386,  90.9944,  91.2348, 103.9581,
       110.7631, 107.5665,  97.7183,  90.9979,  93.8057, 109.4221,
       116.8316, 104.4202,  97.8529,  88.1973,  87.5366,  97.2387,
       103.9086, 105.7486,  94.8823,  89.2977,  89.3585, 110.6844,
       119.0166, 110.533 ,  98.2672,  86.3   ,  90.8364, 104.3538,
       112.8066, 112.9014, 100.1209,  88.9251,  92.775 , 114.3266,
       119.488 , 107.3753,  99.1028,  89.3583,  90.0698, 102.8204,
       114.7068, 113.5958,  99.4712,  90.3566,  93.8095, 107.3312,
       111.9646, 103.3679,  93.5772,  87.5566,  92.7603, 101.14  ,
```

```
       113.0357, 109.8601,  96.7431,  90.3805,  94.3417, 105.2722,
       115.501 , 106.734 , 102.9948,  91.0092,  90.9634, 100.6957,
       110.148 , 108.1756,  99.2809,  91.7871,  97.2853, 113.4732,
       124.2549, 112.8811, 104.7631,  90.2867,  92.134 , 101.878 ,
       108.5497, 108.194 , 100.4172,  92.3837,  99.7033, 109.3477,
       120.2696, 116.3788, 104.4706,  89.7461,  91.093 , 102.6495,
       111.6354, 110.5925, 101.9204,  91.5959,  93.0628, 103.2203,
       117.0837, 106.6688,  95.3548,  89.3254,  90.7369, 104.0375,
       114.5397, 115.5159, 102.7637,  91.4867,  92.89  , 112.7694,
       114.8505,  99.4901, 101.0396,  88.353 ,  92.0805, 102.1532,
       112.1538, 108.9312,  98.6154,  93.6137,  97.3359, 114.7212,
       129.4048])
```

We can see here that this series consist an anamolous data which is the last one.

In [ ]:
```
df = df.drop(df.index[df['average_monthly_ridership'] == ' n=114'])
```

In [ ]:
```
df['average_monthly_ridership'].unique()
```

Out[10]:
```
array(['648', '646', '639', '654', '630', '622', '617', '613', '661',
       '695', '690', '707', '817', '839', '810', '789', '760', '724',
       '704', '691', '745', '803', '780', '761', '857', '907', '873',
       '910', '900', '880', '867', '854', '928', '1064', '1103', '1026',
       '1102', '1080', '1034', '1083', '1078', '1020', '984', '952',
       '1033', '1114', '1160', '1058', '1209', '1200', '1130', '1182',
       '1152', '1116', '1098', '1044', '1142', '1222', '1234', '1155',
       '1286', '1281', '1224', '1280', '1228', '1181', '1156', '1124',
       '1205', '1260', '1188', '1212', '1269', '1246', '1299', '1284',
       '1345', '1341', '1308', '1448', '1454', '1467', '1431', '1510',
       '1558', '1536', '1523', '1492', '1437', '1365', '1310', '1441',
       '1450', '1424', '1360', '1429', '1440', '1414', '1408', '1337',
       '1258', '1214', '1326', '1417', '1329', '1461', '1425', '1419',
       '1432', '1394', '1327'], dtype=object)
```

Now our data is clean !!!

Changing data type of both the column

- Assign int to `monthly_ridership_data` column
- Assign datetime to `month` column

In [16]:
```
df['value'] = df['value'].astype(np.int32)
```

In [19]:
```
df['DATE'] = pd.to_datetime(df['DATE'],)
```

In [22]:
```
df.dtypes
```

Out[22]:
```
DATE     datetime64[ns]
value             int32
dtype: object
```

Time Series Analysis

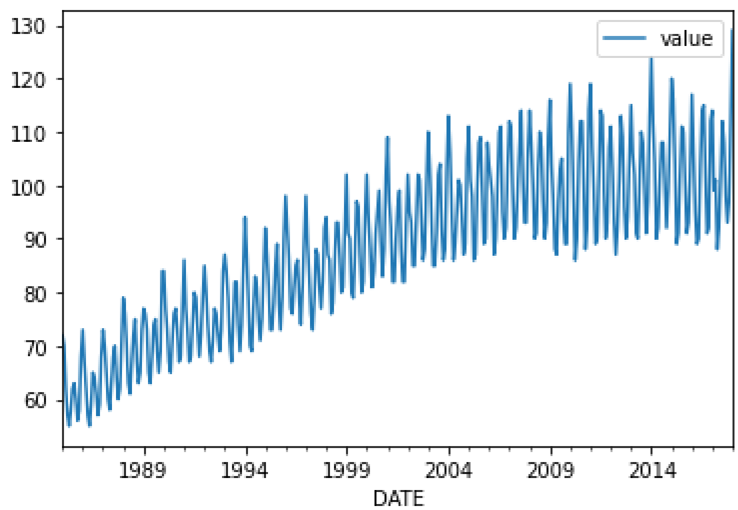
Horizental Pattern :- Horizontal pattern exists when data values fluctuate around a constant mean. This is the simplest pattern and the easiest to predict. An example is sales of a product that do not increase or decrease over time. This type of pattern is common for products in the mature stage of their life cycle, in which demand is steady and predictable.

Trend Pattern:- As the name suggests trend depicts the variation in the output as time increases.It is often non-linear. Sometimes we will refer to trend as “changing direction” when it might go from an increasing trend to a decreasing trend.

Seasonal Pattern:- As its name depicts it shows the repeated pattern over time. In layman terms, it shows the seasonal variation of data over time.

Moving Average:-As the name suggests moving average is a technique to get an overall idea of the trends in a data set; it is an average of any subset of numbers. The moving average is extremely useful for forecasting long-term trends

In [23]:
```
# Normal line plot so that we can see data variation
# We can observe that average number of riders is increasing most of the time
# We'll later see decomposed analysis of that curve
df.plot.line(x = 'DATE', y = 'value')
plt.show()
```



Ploting monthly variation of dataset

It gives us idea about seasonal variation of our data set

In [24]:
```
to_plot_monthly_variation = df
```

In [25]:
```
# only storing month for each index
mon = df['DATE']
```

In [26]:
```
# decompose yyyy-mm data-type
temp= pd.DatetimeIndex(mon)
```

In [27]:
```
# assign month part of that data to ```month``` variable
month = pd.Series(temp.month)
```

In [28]:
```
# dropping month from to_plot_monthly_variation
to_plot_monthly_variation = to_plot_monthly_variation.drop(['DATE'], axis = 1)
```

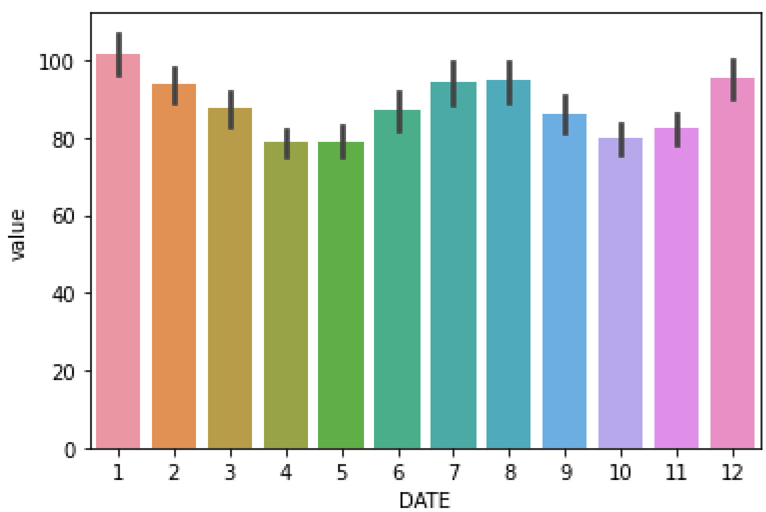
In [29]:
```
# join months so we can get month to average monthly rider mapping
to_plot_monthly_variation = to_plot_monthly_variation.join(month)
```

In [30]:
```
# A quick glance
to_plot_monthly_variation.head()
```

Out[30]:

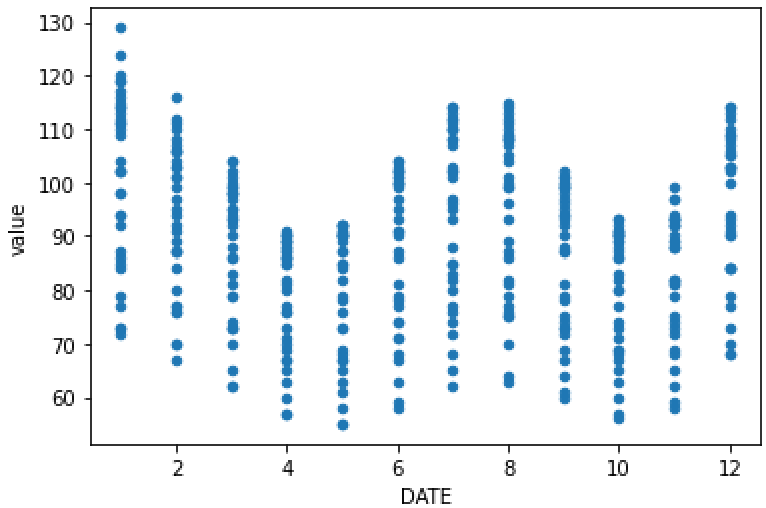
	value	DATE
0	72	1
1	70	2
2	62	3
3	57	4
4	55	5

In [33]:
```
# Plotting bar plot for each month
sns.barplot(x = 'DATE', y = 'value', data = to_plot_monthly_variation)
plt.show()
```



Well this looks tough to decode. Not a typical box plot. One can infer that data is too sparse for this graph to represent any pattern. Hence it cannot represents monthly variation effectively.In such a scenerio we can use our traditional scatter plot to understand pattern in dataset

In [34]:
```
to_plot_monthly_variation.plot.scatter(x = 'DATE', y = 'value')
plt.show()
```



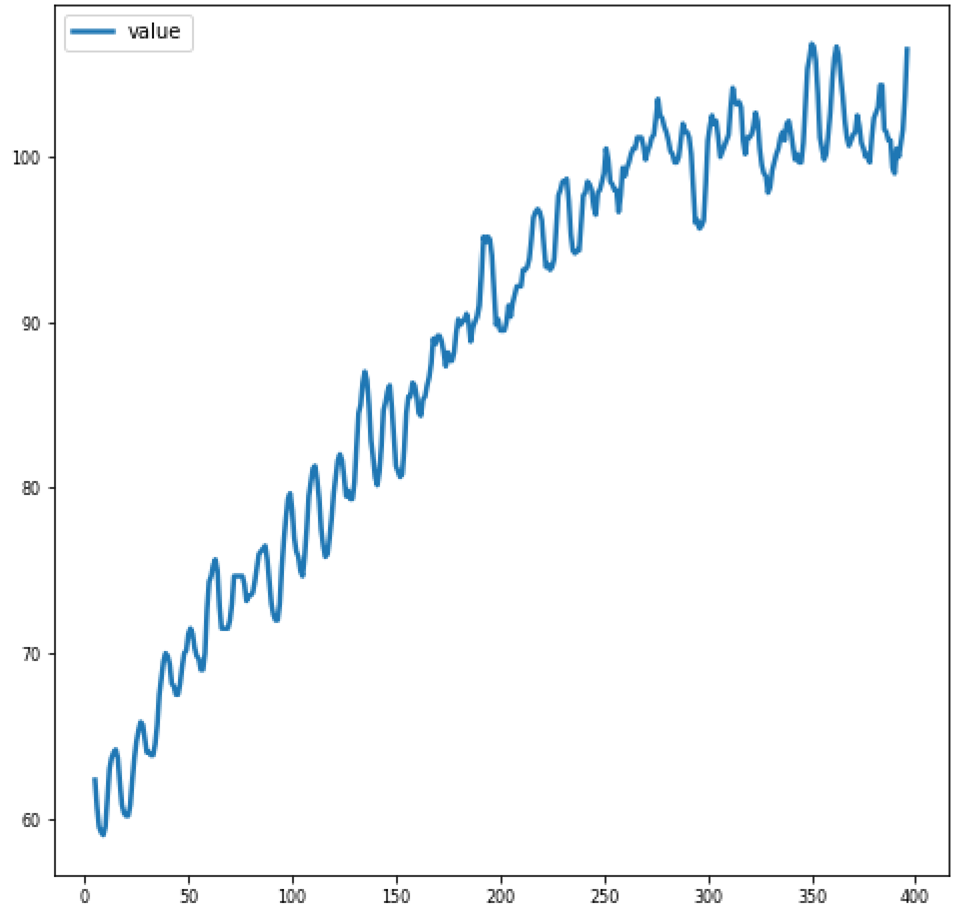
We can see here the yearly variation of data in this plot. To understand this curve more effectively first look at the every row from bottom to top and see each year's variation.To understand yearly variation take a look at each column representing a month.

Another tool to visualize the data is the seasonal_decompose function in statsmodel. With this, the trend and seasonality become even more obvious.

In [35]:
```
value = df[['value']]
```

Trend Analysis

In [39]:
```
value.rolling(6).mean().plot(figsize=(8,8), linewidth=2.5, fontsize=8)
plt.show()
```



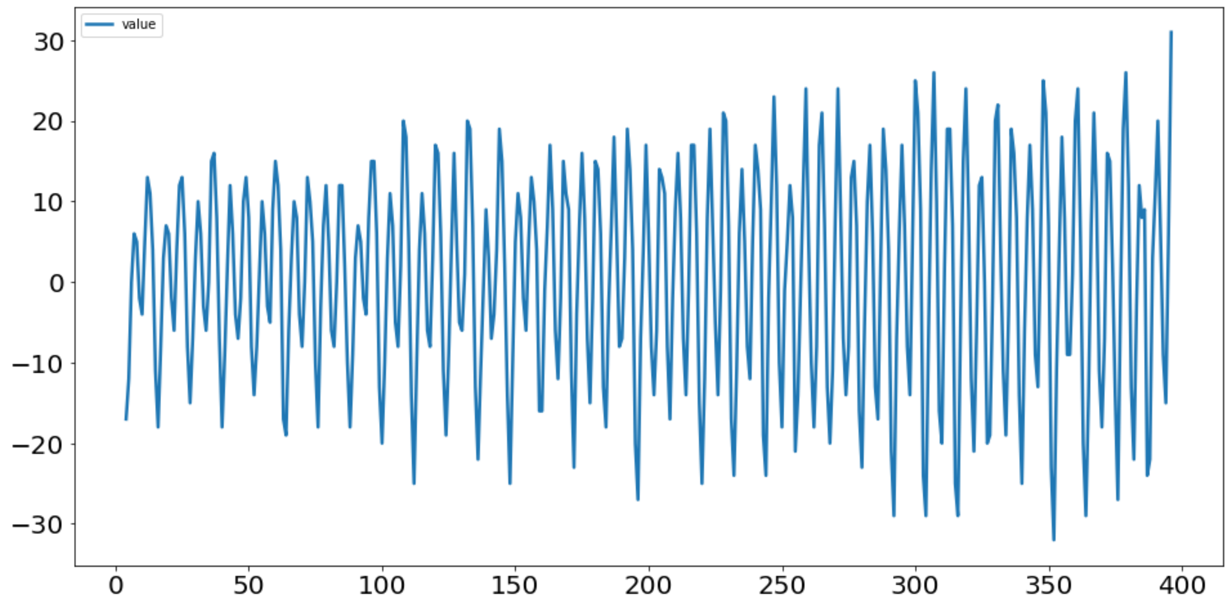
For trend analysis, we use smoothing techniques. In statistics smoothing a data set means to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal.We implement smoothing by taking moving averages. [Exponential moving average] is frequently used to compute smoothed function.Here we used the rolling method which is inbuilt in pandas and frequently used for smoothing.

Seasonability Analysis

Two most famous seasonability analysis algorithms are:-

[Using 1st discrete difference of object (https://machinelearningmastery.com/difference-time-series-dataset-python/)](https://machinelearningmastery.com/difference-time-series-dataset-python/)

In [43]:
```
value.diff(periods=4).plot(figsize=(16,8), linewidth=2.5, fontsize=20)
plt.show()
```



The above figure represents difference between average rider of a month and 4 months before that month i.e

$$d[month] = a[month] - a[month - periods].$$

This gives us idea about variation of data for a period of time.

In [44]:
```
df = df.set_index('DATE')
```

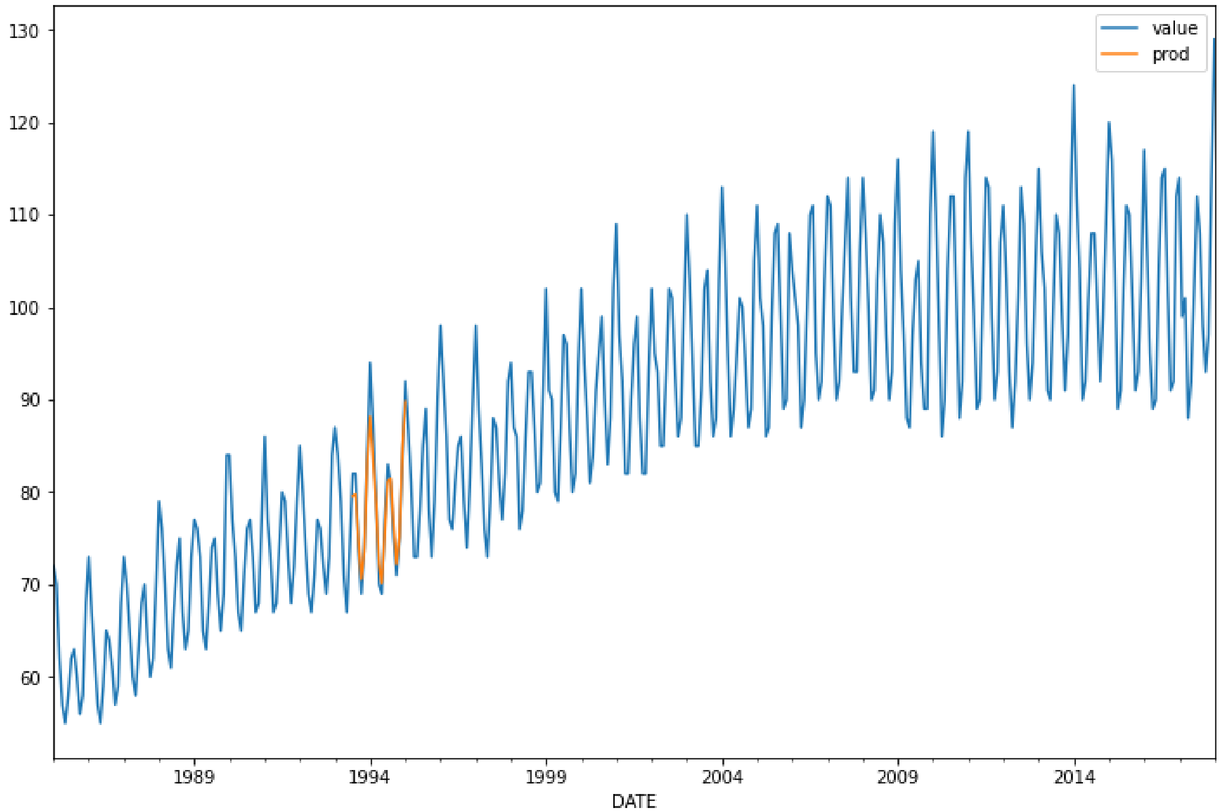
In [45]:
```
# Applying Seasonal ARIMA model to forcast the data
mod = sm.tsa.SARIMAX(df['value'], trend='n', order=(0,1,0), seasonal_order=(1,1,1
results = mod.fit()
print(results.summary())
```

```
/home/venom/.local/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.p
y:536: ValueWarning: No frequency information was provided, so inferred frequen
cy MS will be used.
  warnings.warn('No frequency information was'
/home/venom/.local/lib/python3.9/site-packages/statsmodels/tsa/base/tsa_model.p
y:536: ValueWarning: No frequency information was provided, so inferred frequen
cy MS will be used.
  warnings.warn('No frequency information was'
 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            3     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.36974D+00    |proj g|=  5.30490D-02

At iterate    5    f=  2.35525D+00    |proj g|=  1.31187D-03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    3      7      9      1     0     0   3.672D-06   2.355D+00
  F =   2.3552511416959585

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
                                      SARIMAX Results
================================================================================
=============
Dep. Variable:                                     value   No. Observations:
397
Model:             SARIMAX(0, 1, 0)x(1, 1, [1], 12)   Log Likelihood
-935.035
Date:                             Fri, 26 Nov 2021   AIC
1876.069
Time:                                     15:07:05   BIC
1887.921
Sample:                                 01-01-1985   HQIC
```

```
1880.770
                                       - 01-01-2018
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12       0.0104      0.059      0.176      0.860      -0.106       0.127
ma.S.L12      -0.7696      0.042    -18.475      0.000      -0.851      -0.688
sigma2         7.4228      0.429     17.285      0.000       6.581       8.264
==============================================================================
====
Ljung-Box (L1) (Q):                  14.41   Jarque-Bera (JB):                3
0.81
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               2.74   Skew:                            -
0.05
Prob(H) (two-sided):                  0.00   Kurtosis:
4.38
==============================================================================
====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

Forecast

In [46]:
```
df['prod'] = results.predict(start = 102, end= 120, dynamic= True)
df[['value', 'prod']].plot(figsize=(12, 8))
plt.show()
```



Forecast Accuracy

In [52]:
```
expected=df['value'].tail(12)
predictions=df['prod'].tail(12)
```

In [67]:
```
len(expected)
```

Out[67]:
```
12
```

In [75]:
```
predictions=predictions.fillna(0)
```

In [79]:
```
predictions.astype('int32')
```

Out[79]:
```
DATE
2017-02-01    0
2017-03-01    0
2017-04-01    0
2017-05-01    0
2017-06-01    0
2017-07-01    0
2017-08-01    0
2017-09-01    0
2017-10-01    0
2017-11-01    0
2017-12-01    0
2018-01-01    0
Name: prod, dtype: int32
```

In [81]:
```
expected
```

Out[81]:
```
DATE
2017-02-01     99
2017-03-01    101
2017-04-01     88
2017-05-01     92
2017-06-01    102
2017-07-01    112
2017-08-01    108
2017-09-01     98
2017-10-01     93
2017-11-01     97
2017-12-01    114
2018-01-01    129
Name: value, dtype: int32
```

In [80]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
mse = mean_squared_error(expected, predictions)
rmse = sqrt(mse)
print('Root MeanSquared Error: %f' % rmse)
```

```
Root MeanSquared Error: 103.328360
```

The RMSE error values are in the same units as the predictions. As with the mean squared error, an RMSE of zero indicates no error

ML LAB 12

Explore Holt's Linear Exponential Smoothing, Nonlinear Trend Regression, and Seasonality for the Time Series Analysis in a given business environment.

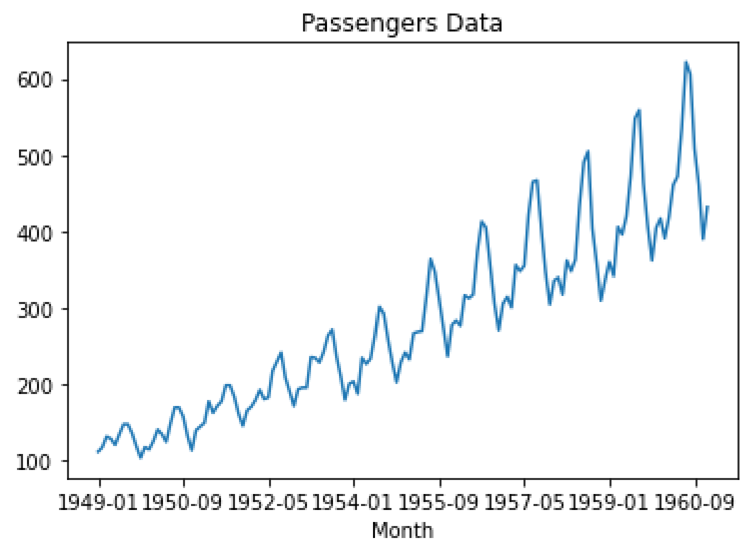
Importing the libraries

In [1]:
```
# dataframe opertations - pandas
import pandas as pd
# plotting data - matplotlib
from matplotlib import pyplot as plt
# time series - statsmodels
# Seasonality decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.seasonal import seasonal_decompose
# holt winters
# single exponential smoothing
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
# double and triple exponential smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

In [2]:
```
airline = pd.read_csv('C:/Users/user/Downloads/archive (3)/international-airline-
airline = pd.read_csv('C:/Users/user/Downloads/archive (3)/international-airline-
# finding shape of the dataframe
print(airline.shape)
# having a look at the data
print(airline.head())

# plotting the original data
airline['International airline passengers: monthly totals in thousands. Jan 49 ? I
```

```
(145, 1)
         International airline passengers: monthly totals in thousands. Jan 49
? Dec 60
Month
1949-01                                                    112.0
1949-02                                                    118.0
1949-03                                                    132.0
1949-04                                                    129.0
1949-05                                                    121.0
```

Out[2]:
```
<AxesSubplot:title={'center':'Passengers Data'}, xlabel='Month'>
```



Fitting the Data with Holt-Winters Exponential Smoothing

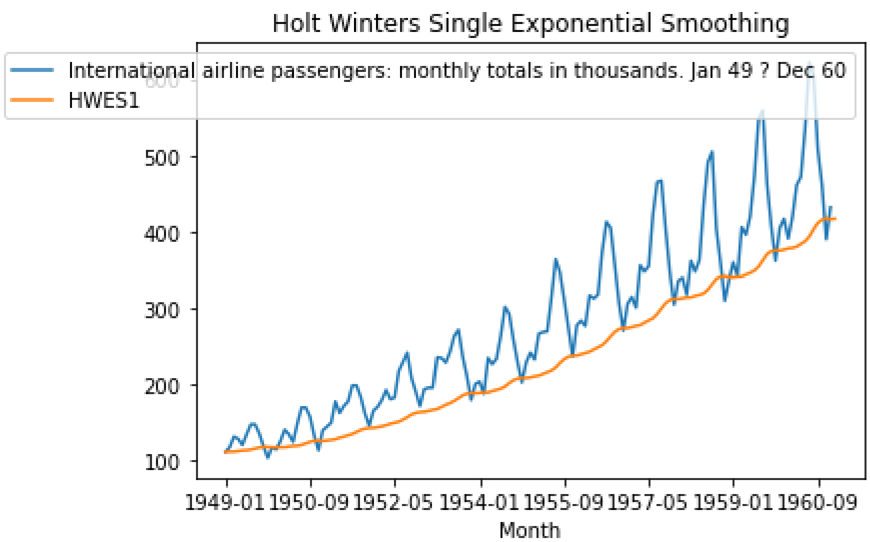
In [3]:
```
# Set the frequency of the date time index as Monthly start as indicated by the da
airline.index.freq = 'MS'
# Set the value of Alpha and define m (Time Period)
m = 12
alpha = 1/(2*m)
```

Single HWES

Now, we will fit the data on the Single Exponential Smoothing,

In [4]:
```
airline['HWES1'] = SimpleExpSmoothing(airline['International airline passengers: 
airline[['International airline passengers: monthly totals in thousands. Jan 49 ?
```

```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:57
8: ValueWarning: An unsupported index was provided and will be ignored when e.
g. forecasting.
  warnings.warn('An unsupported index was provided and will be'
C:\Users\user\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:
427: FutureWarning: After 0.13 initialization must be handled at model creation
  warnings.warn(
```

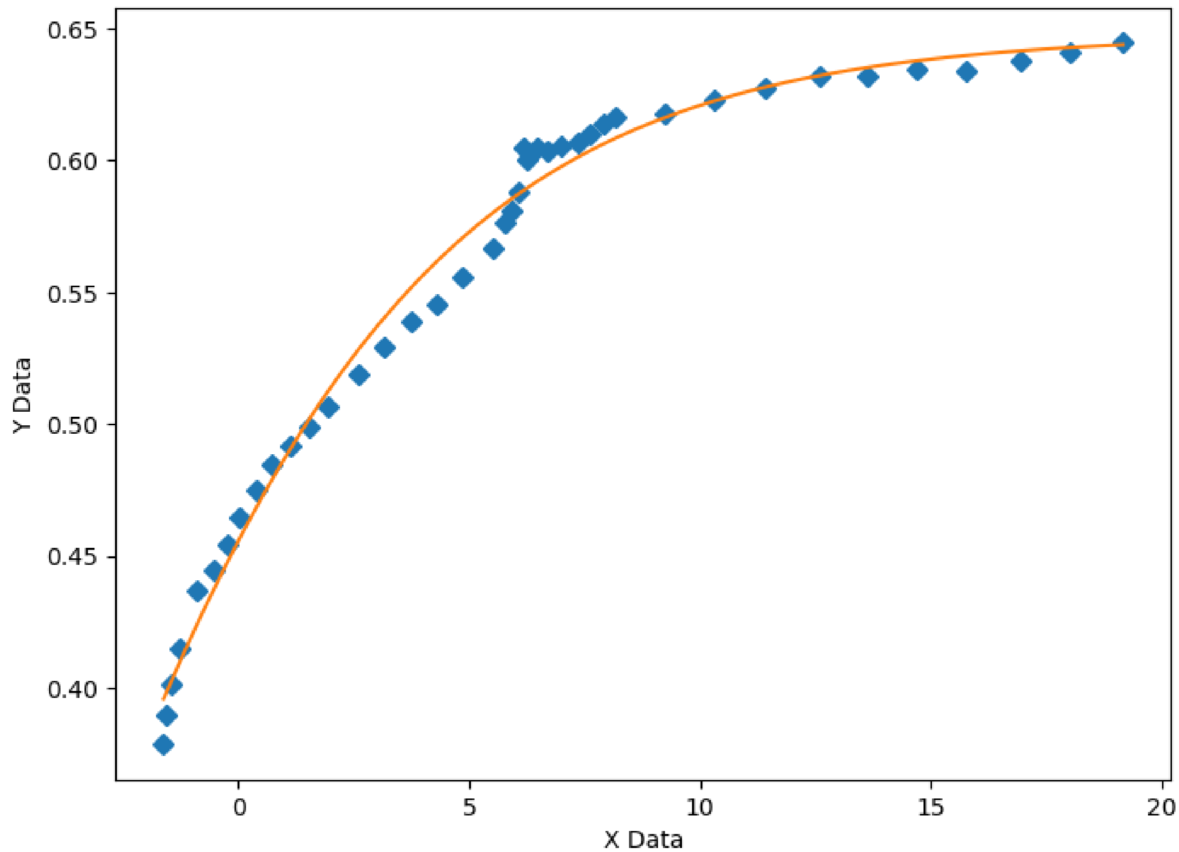


Non Linear Trend Regression

In [5]:
```
import numpy, scipy, matplotlib
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.optimize import differential_evolution
import warnings

xData = numpy.array([19.1647, 18.0189, 16.9550, 15.7683, 14.7044, 13.6269, 12.604
yData = numpy.array([0.644557, 0.641059, 0.637555, 0.634059, 0.634135, 0.631825,


def func(x, a, b, Offset): # Sigmoid A With Offset from zunzun.com
    return  1.0 / (1.0 + numpy.exp(-a * (x-b))) + Offset


# function for genetic algorithm to minimize (sum of squared error)
def sumOfSquaredError(parameterTuple):
    warnings.filterwarnings("ignore") # do not print warnings by genetic algorith
    val = func(xData, *parameterTuple)
    return numpy.sum((yData - val) ** 2.0)


def generate_Initial_Parameters():
    # min and max used for bounds
    maxX = max(xData)
    minX = min(xData)
    maxY = max(yData)
    minY = min(yData)

    parameterBounds = []
    parameterBounds.append([minX, maxX]) # search bounds for a
    parameterBounds.append([minX, maxX]) # search bounds for b
    parameterBounds.append([0.0, maxY]) # search bounds for Offset

    # "seed" the numpy random number generator for repeatable results
    result = differential_evolution(sumOfSquaredError, parameterBounds, seed=3)
    return result.x

# generate initial parameter values
geneticParameters = generate_Initial_Parameters()

# curve fit the test data
fittedParameters, pcov = curve_fit(func, xData, yData, geneticParameters)

print('Parameters', fittedParameters)

modelPredictions = func(xData, *fittedParameters)

absError = modelPredictions - yData

SE = numpy.square(absError) # squared errors
MSE = numpy.mean(SE) # mean squared errors
RMSE = numpy.sqrt(MSE) # Root Mean Squared Error, RMSE
Rsquared = 1.0 - (numpy.var(absError) / numpy.var(yData))
print('RMSE:', RMSE)
print('R-squared:', Rsquared)
```

```
##########################################################
# graphics output section
def ModelAndScatterPlot(graphWidth, graphHeight):
    f = plt.figure(figsize=(graphWidth/100.0, graphHeight/100.0), dpi=100)
    axes = f.add_subplot(111)

    # first the raw data as a scatter plot
    axes.plot(xData, yData,  'D')

    # create data for the fitted equation plot
    xModel = numpy.linspace(min(xData), max(xData))
    yModel = func(xModel, *fittedParameters)

    # now the model as a line plot
    axes.plot(xModel, yModel)

    axes.set_xlabel('X Data') # X axis data label
    axes.set_ylabel('Y Data') # Y axis data label

    plt.show()
    plt.close('all') # clean up after using pyplot

graphWidth = 800
graphHeight = 600
ModelAndScatterPlot(graphWidth, graphHeight)
```

```
Parameters [ 0.21540306 -6.67449153 -0.35241296]
RMSE: 0.008428738373451258
R-squared: 0.9886222631484034
```



Seasonality for the Time Series Analysis

In [6]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# reading the dataset using read_csv
df = pd.read_csv("C:/Users/user/Downloads/Dataset-main/Dataset-main/stock_data.cs
                 parse_dates=True,
                 index_col="Date")

# displaying the first five rows of dataset
df.head()
```

Out[6]:

	Unnamed: 0	Open	High	Low	Close	Volume	Name
Date							
2006-01-03	NaN	39.69	41.22	38.79	40.91	24232729	AABA
2006-01-04	NaN	41.22	41.90	40.77	40.97	20553479	AABA
2006-01-05	NaN	40.93	41.73	40.85	41.53	12829610	AABA
2006-01-06	NaN	42.88	43.57	42.80	43.21	29422828	AABA
2006-01-09	NaN	43.10	43.66	42.82	43.42	16268338	AABA

In [7]:
```
# deleting column
df.drop(columns='Unnamed: 0')
```

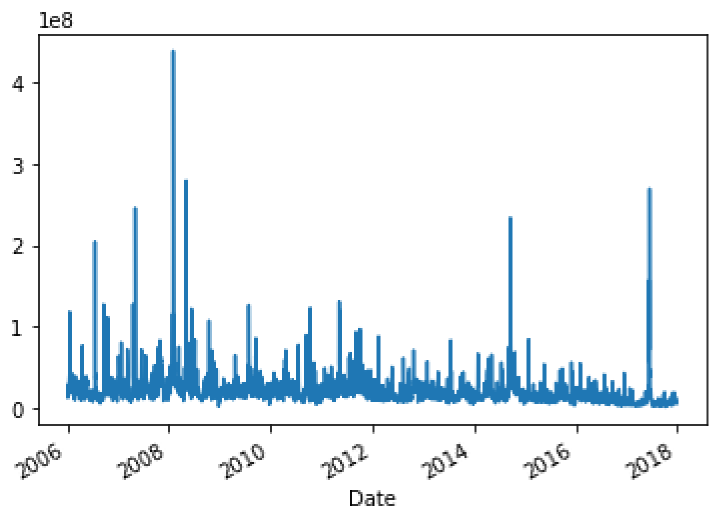
Out[7]:

Date	Open	High	Low	Close	Volume	Name
2006-01-03	39.69	41.22	38.79	40.91	24232729	AABA
2006-01-04	41.22	41.90	40.77	40.97	20553479	AABA
2006-01-05	40.93	41.73	40.85	41.53	12829610	AABA
2006-01-06	42.88	43.57	42.80	43.21	29422828	AABA
2006-01-09	43.10	43.66	42.82	43.42	16268338	AABA
...	...	...	...	...	...	...
2017-12-22	71.42	71.87	71.22	71.58	10979165	AABA
2017-12-26	70.94	71.39	69.63	69.86	8542802	AABA
2017-12-27	69.77	70.49	69.69	70.06	6345124	AABA
2017-12-28	70.12	70.32	69.51	69.82	7556877	AABA
2017-12-29	69.79	70.13	69.43	69.85	6613070	AABA

3019 rows × 6 columns

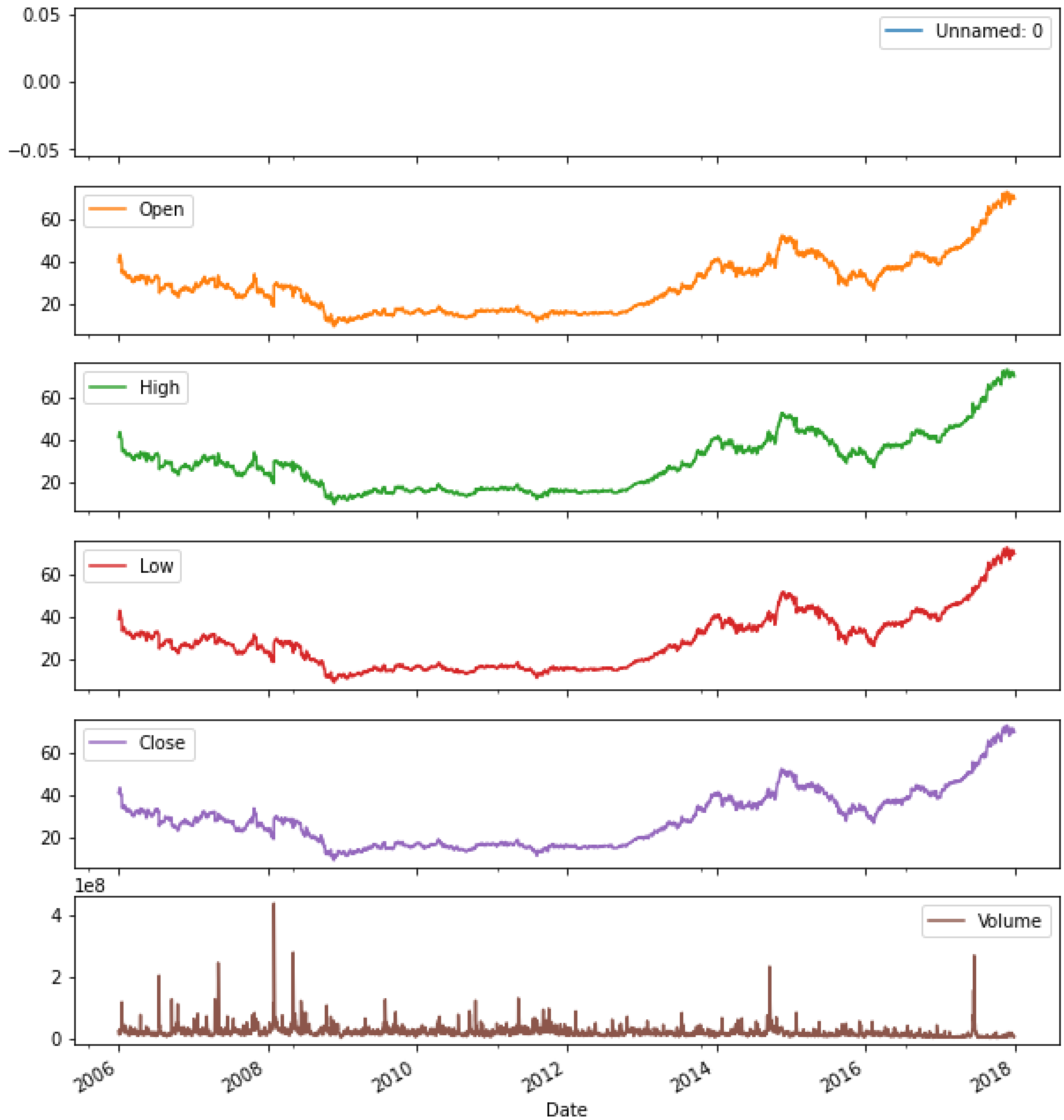
In [8]:
```
df['Volume'].plot()
```

Out[8]: <AxesSubplot:xlabel='Date'>



In [9]:
```
df.plot(subplots=True, figsize=(10, 12))
```

Out[9]:
```
array([<AxesSubplot:xlabel='Date'>, <AxesSubplot:xlabel='Date'>,
       <AxesSubplot:xlabel='Date'>, <AxesSubplot:xlabel='Date'>,
       <AxesSubplot:xlabel='Date'>, <AxesSubplot:xlabel='Date'>],
      dtype=object)
```



The line plots used above are good for showing seasonality.

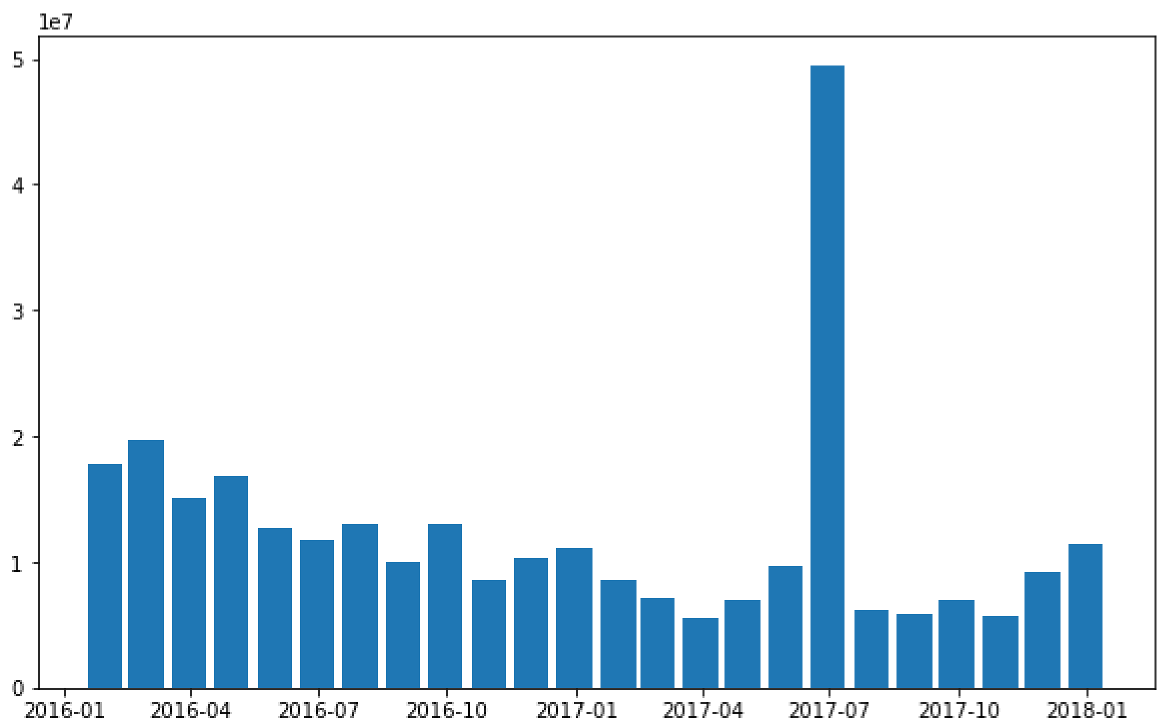
Seasonality:

In time-series data, seasonality is the presence of variations that occur at specific regular time intervals less than a year, such as weekly, monthly, or quarterly.

Resampling for months or weeks and making bar plots is another very simple and widely used method of finding seasonality. Here we are going to make a bar plot of month data for 2016 and 2017.

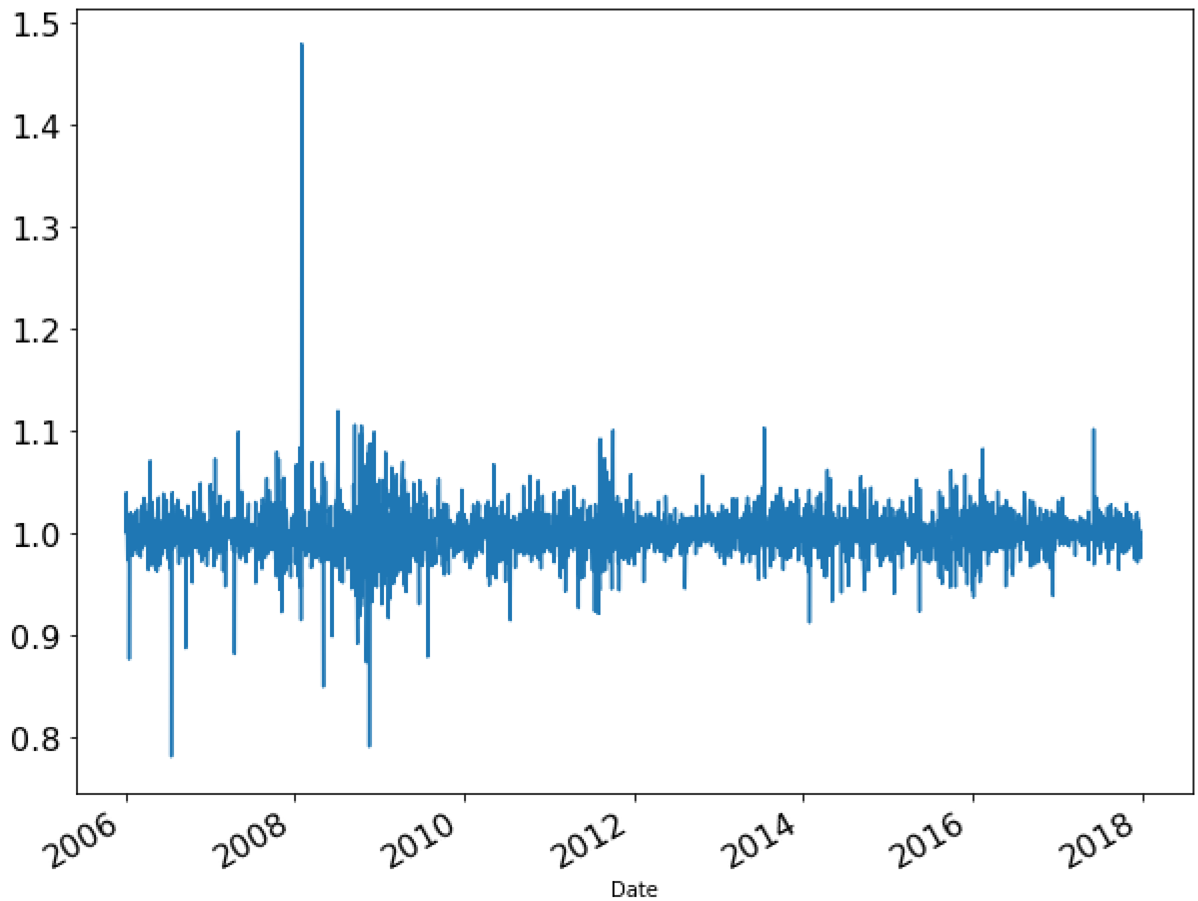
In [10]:
```
# Resampling the time series data based on monthly 'M' frequency
df_month = df.resample("M").mean()

# using subplot
fig, ax = plt.subplots(figsize=(10, 6))

# plotting bar graph
ax.bar(df_month['2016':].index,
    df_month.loc['2016':, "Volume"],
    width=25, align='center')
```

Out[10]: <BarContainer object of 24 artists>



In [12]:
```
#Plotting Chages in the data

df['Change'] = df.Close.div(df.Close.shift())
df['Change'].plot(figsize=(10, 8), fontsize=16)
```

Out[12]: <AxesSubplot:xlabel='Date'>



In [13]:
```
df['2017']['Change'].plot(figsize=(10, 6))
```

Out[13]: <AxesSubplot:xlabel='Date'>