major-code-pdf

May 27, 2024

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
sb.set()
from matplotlib import pyplot as plt, font_manager as fm
```

[2]:
```
milkData = pd.read_csv("C:\\Users\\harsh\\Downloads\\milknew (2).csv")
milkData
```

[2]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	high
1	6.6	36	0	1	0	1	253	high
2	8.5	70	1	1	1	1	246	low
3	9.5	34	1	1	0	1	255	low
4	6.6	37	0	0	0	0	255	medium
…	…	…	…	…	…	…	…	…
1054	6.7	45	1	1	0	0	247	medium
1055	6.7	38	1	0	1	0	255	high
1056	3.0	40	1	1	1	1	255	low
1057	6.8	43	1	0	1	0	250	high
1058	8.6	55	0	1	1	1	255	low

[1059 rows x 8 columns]

[3]:
```
milkData.head(5)
```

[3]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	high
1	6.6	36	0	1	0	1	253	high
2	8.5	70	1	1	1	1	246	low
3	9.5	34	1	1	0	1	255	low
4	6.6	37	0	0	0	0	255	medium

[4]:
```
milkData.tail(5)
```

[4]:
```
        pH  Temprature  Taste  Odor  Fat   Turbidity  Colour   Grade
1054  6.7          45      1     1     0          0     247  medium
1055  6.7          38      1     0     1          0     255    high
1056  3.0          40      1     1     1          1     255     low
1057  6.8          43      1     0     1          0     250    high
1058  8.6          55      0     1     1          1     255     low
```

[5]:
```
milkData.keys()
```

[5]:
```
Index(['pH', 'Temprature', 'Taste', 'Odor', 'Fat ', 'Turbidity', 'Colour',
       'Grade'],
      dtype='object')
```

[6]:
```
milkData.describe()
```

[6]:
```
                pH   Temprature        Taste         Odor          Fat   \
count  1059.000000  1059.000000  1059.000000  1059.000000  1059.000000
mean      6.630123    44.226629     0.546742     0.432483     0.671388
std       1.399679    10.098364     0.498046     0.495655     0.469930
min       3.000000    34.000000     0.000000     0.000000     0.000000
25%       6.500000    38.000000     0.000000     0.000000     0.000000
50%       6.700000    41.000000     1.000000     0.000000     1.000000
75%       6.800000    45.000000     1.000000     1.000000     1.000000
max       9.500000    90.000000     1.000000     1.000000     1.000000

         Turbidity       Colour
count  1059.000000  1059.000000
mean      0.491029   251.840415
std       0.500156     4.307424
min       0.000000   240.000000
25%       0.000000   250.000000
50%       0.000000   255.000000
75%       1.000000   255.000000
max       1.000000   255.000000
```

[7]:
```
milkData.shape
```

[7]:
```
(1059, 8)
```

[8]:
```
milkData.dtypes
```

[8]:
```
pH            float64
Temprature      int64
Taste           int64
Odor            int64
Fat             int64
Turbidity       int64
```

```
Colour          int64
Grade          object
dtype: object
```

[9]:
```
milkData.dropna()
milkData.shape
```

[9]:
```
(1059, 8)
```

[10]:
```
milkData.duplicated().sum()
```

[10]:
```
976
```

[11]:
```
milkData.loc[milkData.duplicated(),:]
```

[11]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
35	6.8	45	0	1	1	1	255	high
48	9.5	34	1	1	0	1	255	low
50	6.6	37	1	1	1	1	255	high
51	5.5	45	1	0	1	1	250	low
52	4.5	60	0	1	1	1	250	low
…	…	…	…	…	…	…	…	…
1054	6.7	45	1	1	0	0	247	medium
1055	6.7	38	1	0	1	0	255	high
1056	3.0	40	1	1	1	1	255	low
1057	6.8	43	1	0	1	0	250	high
1058	8.6	55	0	1	1	1	255	low

[976 rows x 8 columns]

[12]:
```
milkData.isnull().sum()
```

[12]:
```
pH            0
Temprature    0
Taste         0
Odor          0
Fat           0
Turbidity     0
Colour        0
Grade         0
dtype: int64
```

[13]:
```
milkData.nunique()
```

[13]:
```
pH            16
Temprature    17
Taste          2
```

```
Odor           2
Fat            2
Turbidity      2
Colour         9
Grade          3
dtype: int64
```

[14]:
```
for i in milkData.columns:
  print(i)
  print(milkData[i].unique())
  print('-----------------------------------------------------')
```

```
pH
[6.6 8.5 9.5 5.5 4.5 8.1 6.7 5.6 8.6 7.4 6.8 6.5 4.7 3.  9.  6.4]
-----------------------------------------------------
Temprature
[35 36 70 34 37 45 60 66 50 55 90 38 40 43 42 41 65]
-----------------------------------------------------
Taste
[1 0]
-----------------------------------------------------
Odor
[0 1]
-----------------------------------------------------
Fat
[1 0]
-----------------------------------------------------
Turbidity
[0 1]
-----------------------------------------------------
Colour
[254 253 246 255 250 247 245 240 248]
-----------------------------------------------------
Grade
['high' 'low' 'medium']
-----------------------------------------------------
```

[15]:
```
for i in milkData.columns:
  print(i)
  print(milkData[i].value_counts())
  print('-----------------------------------------------------')
```

```
pH
pH
6.8    249
6.5    189
6.6    159
6.7     82
```

```
3.0     70
9.0     61
8.6     40
7.4     39
4.5     37
9.5     24
8.1     24
5.5     23
8.5     22
4.7     20
5.6     19
6.4      1
Name: count, dtype: int64
------------------------------------------------------
Temprature
Temprature
45    219
38    179
40    132
37     83
43     77
36     66
50     58
55     48
34     40
41     30
66     24
35     23
70     22
65     22
60     18
90     17
42      1
Name: count, dtype: int64
------------------------------------------------------
Taste
Taste
1    579
0    480
Name: count, dtype: int64
------------------------------------------------------
Odor
Odor
0    601
1    458
Name: count, dtype: int64
------------------------------------------------------
Fat
```

```
Fat
1    711
0    348
Name: count, dtype: int64
----------------------------------------------------------
Turbidity
Turbidity
0    539
1    520
Name: count, dtype: int64
----------------------------------------------------------
Colour
Colour
255    628
250    146
245    115
247     48
246     44
240     32
248     23
253     22
254      1
Name: count, dtype: int64
----------------------------------------------------------
Grade
Grade
low       429
medium    374
high      256
Name: count, dtype: int64
----------------------------------------------------------
```

[16]:
```
milkData.info
```

[16]:
```
<bound method DataFrame.info of        pH  Temprature  Taste  Odor  Fat
Turbidity  Colour   Grade
0     6.6          35      1     0     1           0     254    high
1     6.6          36      0     1     0           1     253    high
2     8.5          70      1     1     1           1     246     low
3     9.5          34      1     1     0           1     255     low
4     6.6          37      0     0     0           0     255  medium
…    …          …     …    …    …         …     …      …
1054  6.7          45      1     1     0           0     247  medium
1055  6.7          38      1     0     1           0     255    high
1056  3.0          40      1     1     1           1     255     low
1057  6.8          43      1     0     1           0     250    high
1058  8.6          55      0     1     1           1     255     low
```

```
[1059 rows x 8 columns]>
```

[17]:
```
milkData1= pd.DataFrame(milkData)
```

[18]:
```
milkData2=milkData.drop(['Grade'], axis=1)
```

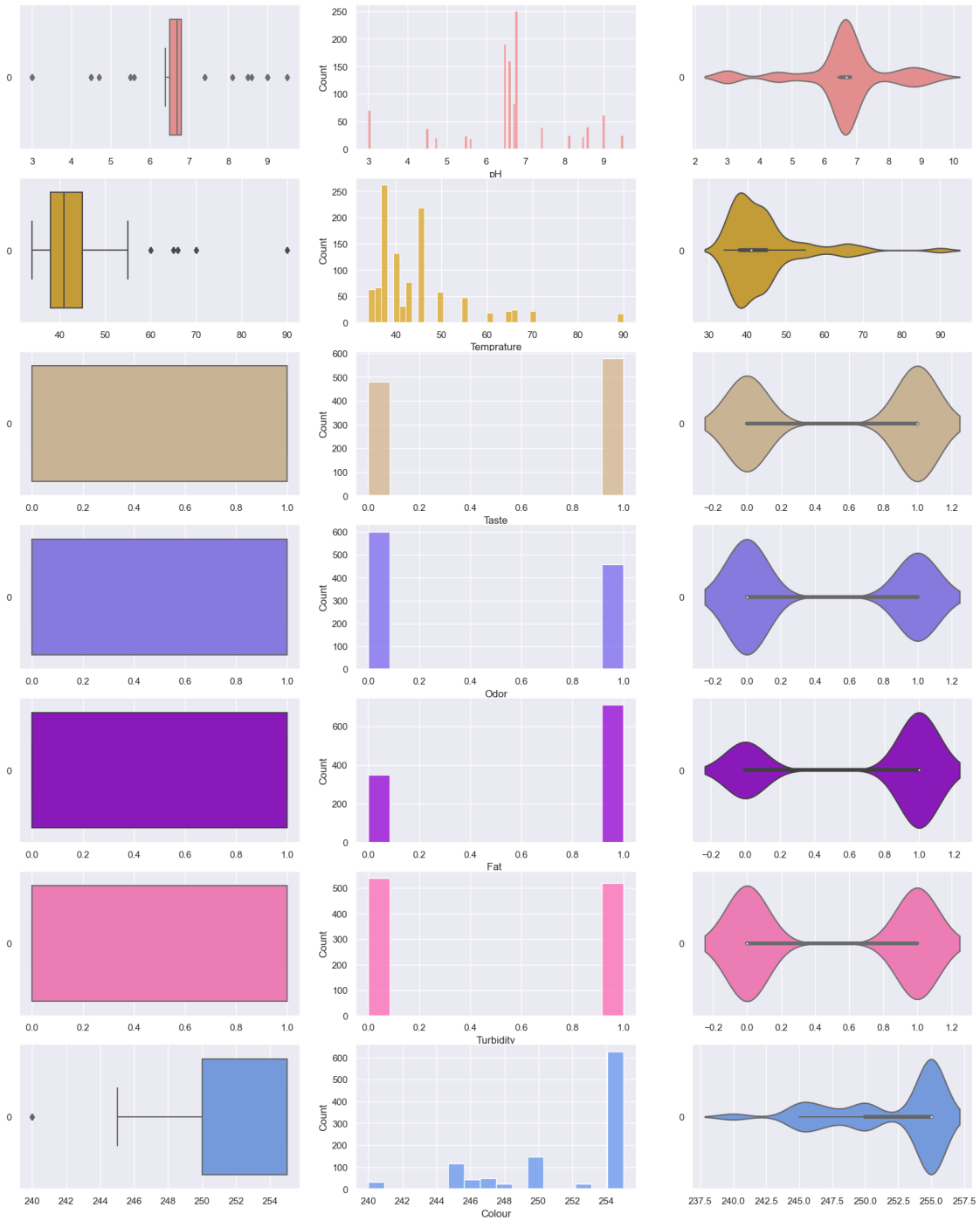
[19]:
```
milkData2
```

[19]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour
0	6.6	35	1	0	1	0	254
1	6.6	36	0	1	0	1	253
2	8.5	70	1	1	1	1	246
3	9.5	34	1	1	0	1	255
4	6.6	37	0	0	0	0	255
…	…	…	…	…	…	…	…
1054	6.7	45	1	1	0	0	247
1055	6.7	38	1	0	1	0	255
1056	3.0	40	1	1	1	1	255
1057	6.8	43	1	0	1	0	250
1058	8.6	55	0	1	1	1	255

```
[1059 rows x 7 columns]
```

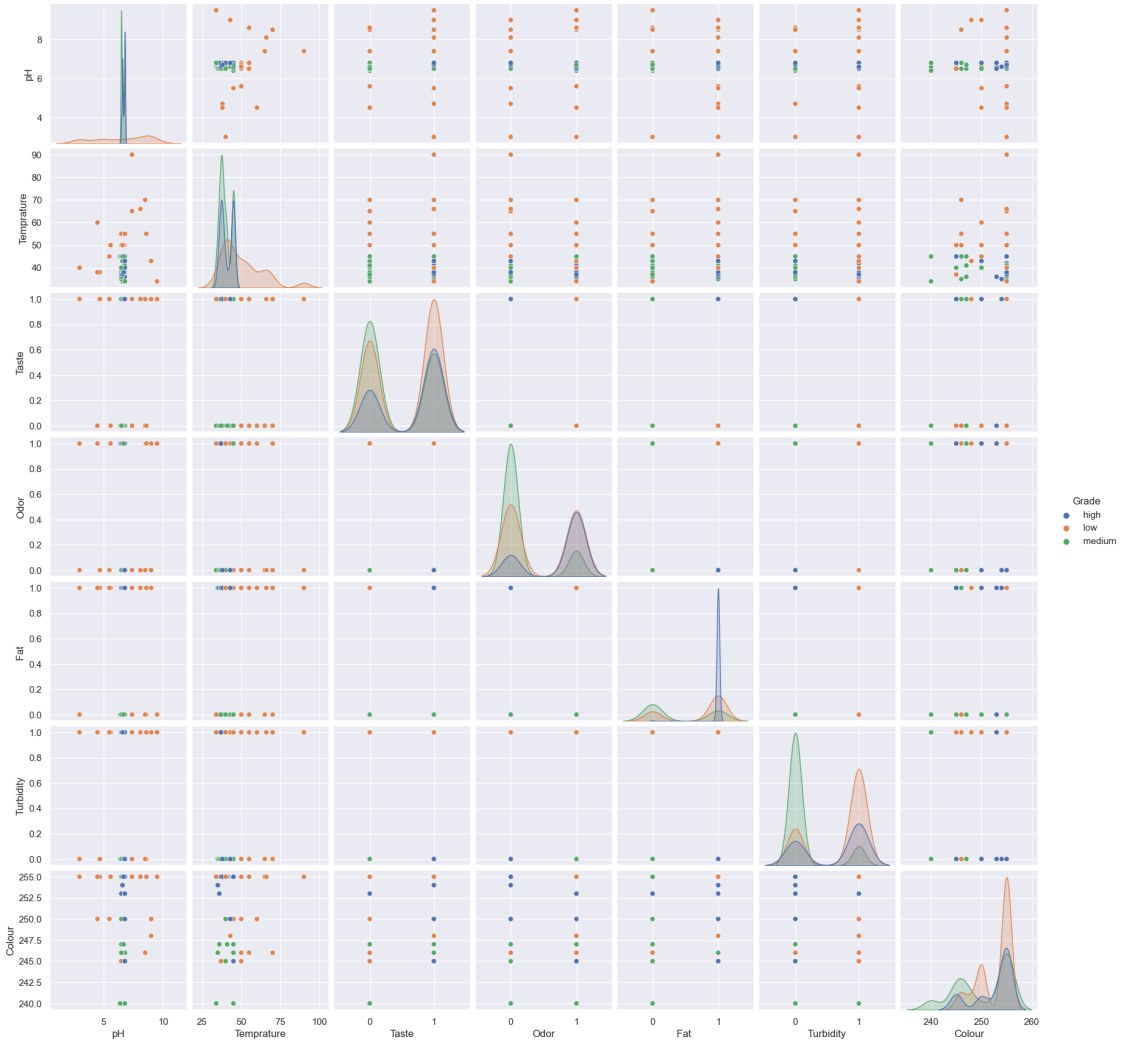
[20]:
```
f, axes = plt.subplots(7, 3, figsize=(20, 25))
colors = ["lightcoral", "goldenrod", "tan", "mediumslateblue", "darkviolet",␣
 ↪"hotpink", "cornflowerblue", "dodgerblue", "royalblue", "mediumaquamarine",␣
 ↪"teal", "firebrick"]

count = 0
for var in milkData2:
    sb.boxplot(data=milkData2[var], orient = "h", color = colors[count], ax =␣
 ↪axes[count,0])
    sb.histplot(data=milkData2[var], color = colors[count], ax = axes[count,1])
    sb.violinplot(data=milkData2[var], orient = "h", color = colors[count], ax␣
 ↪= axes[count,2])
    count += 1
```



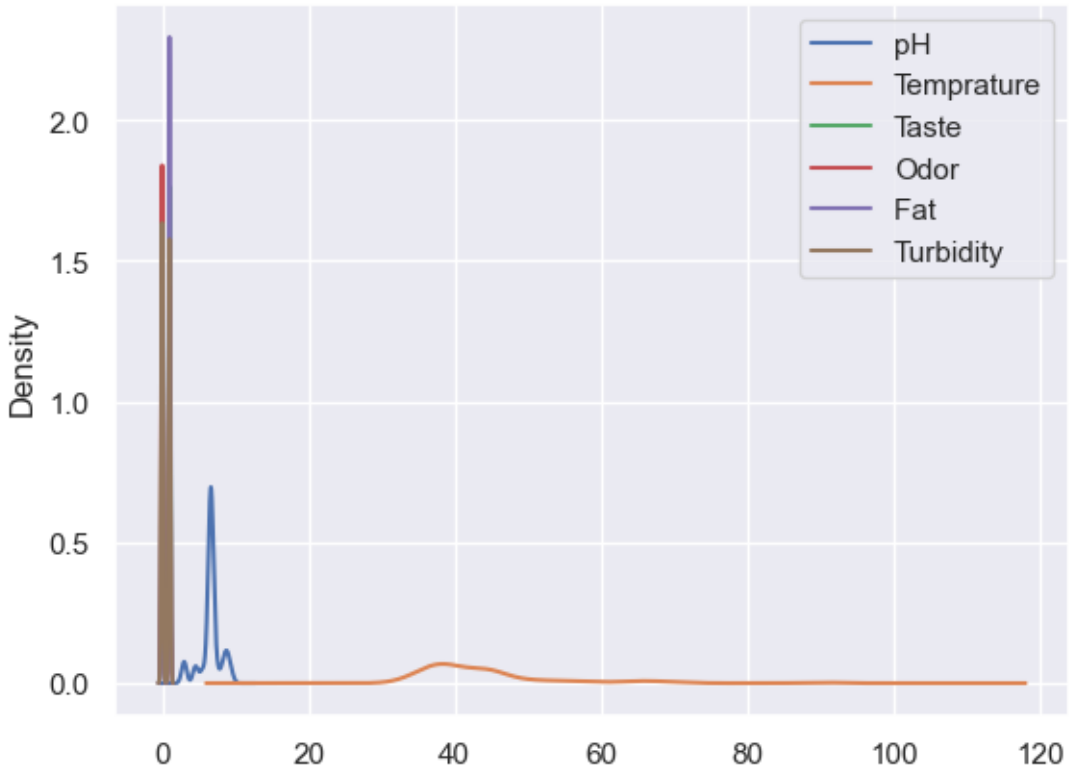
[21]:
```
sb.pairplot(milkData,vars=milkData.columns[:-1],hue='Grade')
```

```
C:\Users\harsh\OneDrive\Documents\anaconda\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[21]: <seaborn.axisgrid.PairGrid at 0x24904625f50>



[22]:
```
cols = milkData.columns[:6]
densityplot = milkData[cols].plot(kind='density')
```



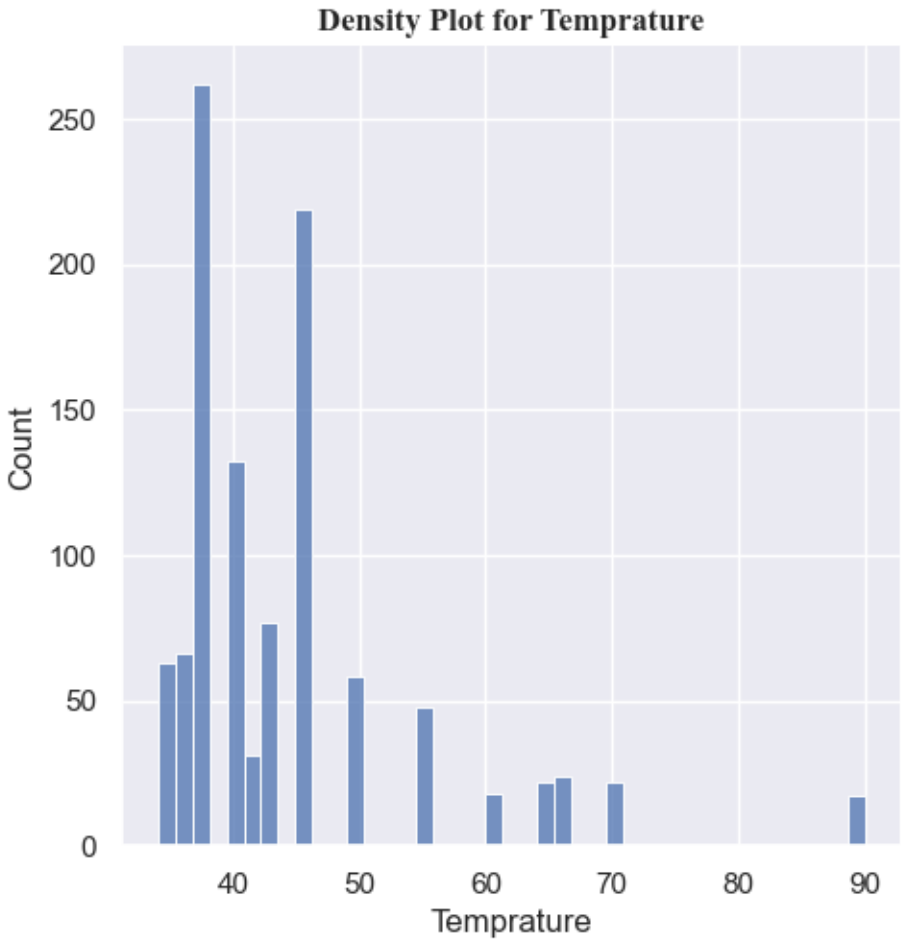
[23]:
```
print('The lowest temperature in the Milk is '+ str(milkData['Temprature'].
  ↪min()))
print('The highest temperature in the Milk is '+ str(milkData['Temprature'].
  ↪max()))
print('The average temperature in the Milk is '+␣
  ↪str(round(milkData['Temprature'].mean(),3)))
#min, max, average for PH
print('The lowest pH in the Milk is '+ str(milkData['pH'].min()))
print('The highest pH in the Milk is '+ str(milkData['pH'].max()))
print('The average pH in the Milk is '+ str(round(milkData['pH'].mean(),3)))
#min,max,averagr for colour
print('The lowest Colour in the Milk is '+ str(milkData['Colour'].min()))
print('The highest Colour in the Milk is '+ str(milkData['Colour'].max()))
print('The average Colour in the Milk is '+ str(round(milkData['Colour'].
  ↪mean(),3)))
```

```
The lowest temperature in the Milk is 34
The highest temperature in the Milk is 90
The average temperature in the Milk is 44.227
The lowest pH in the Milk is 3.0
The highest pH in the Milk is 9.5
The average pH in the Milk is 6.63
```

```
The lowest Colour in the Milk is 240
The highest Colour in the Milk is 255
The average Colour in the Milk is 251.84
```

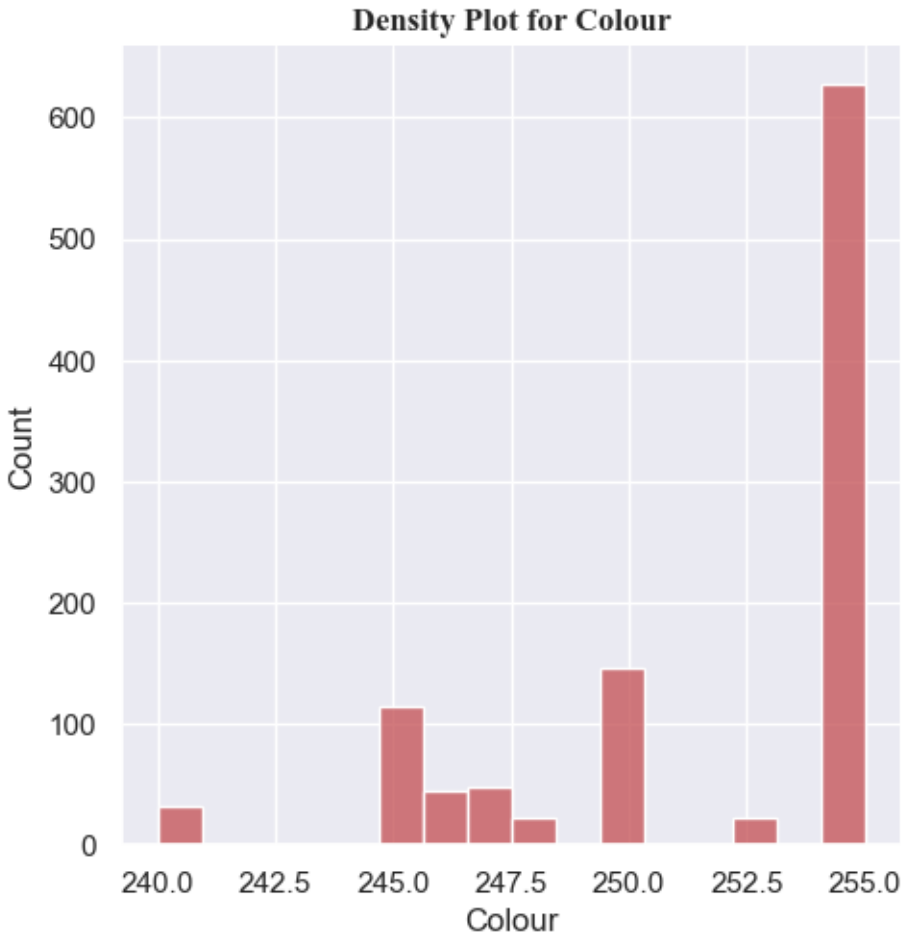
[24]:
```
sb.displot(milkData['Temprature'], color = 'b')
plt.title("Density Plot for Temprature",fontname="Times New Roman", size=12␣
  ↪,fontweight="bold" )
plt.show()
```

```
C:\Users\harsh\OneDrive\Documents\anaconda\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



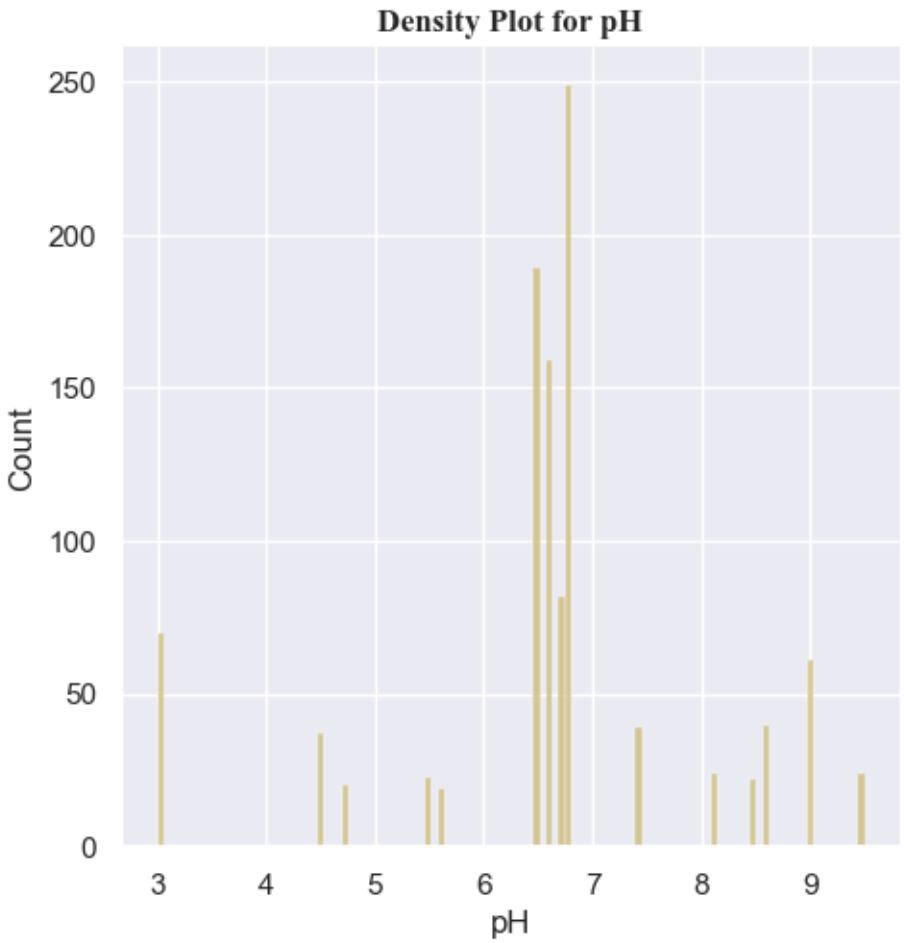
[25]:
```
sb.displot(milkData['Colour'], color = 'r')
plt.title("Density Plot for Colour",fontname="Times New Roman", size=12␣
  ↪,fontweight="bold" )
plt.show()
```

```
C:\Users\harsh\OneDrive\Documents\anaconda\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



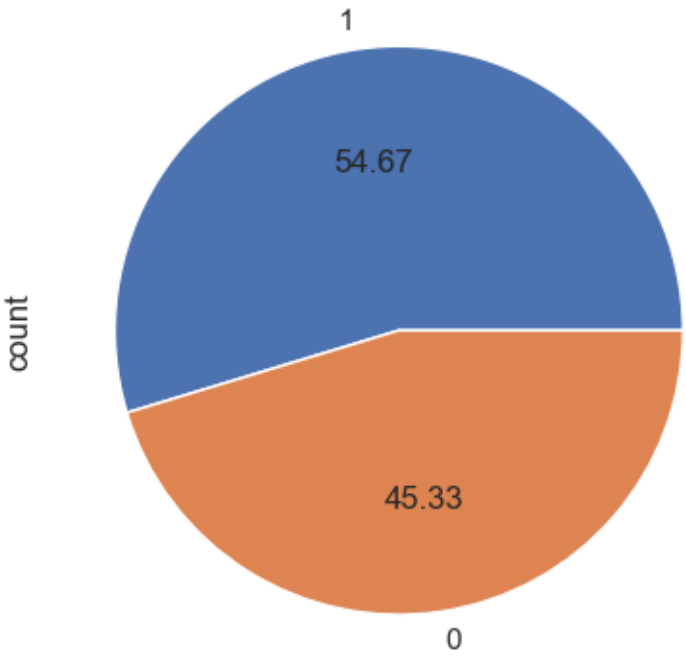
[26]:
```
sb.displot(milkData['pH'], color = 'y')
plt.title("Density Plot for pH",fontname="Times New Roman", size=12␣
 ↪,fontweight="bold" )
plt.show()
```

```
C:\Users\harsh\OneDrive\Documents\anaconda\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

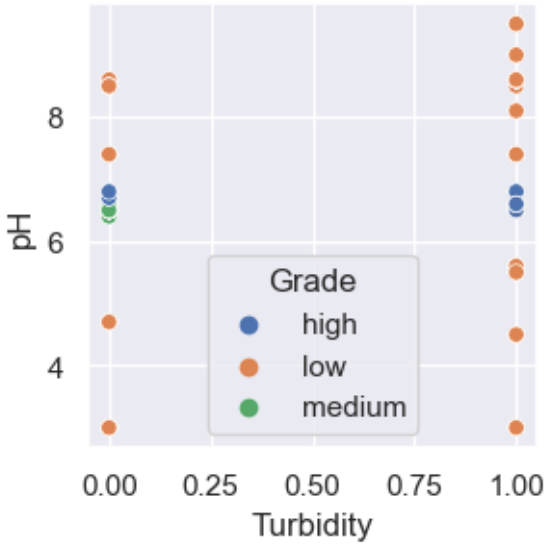


[27]:
```
milkData['Taste'].value_counts().plot(kind='pie',autopct='%.2f')
```

[27]: <Axes: ylabel='count'>



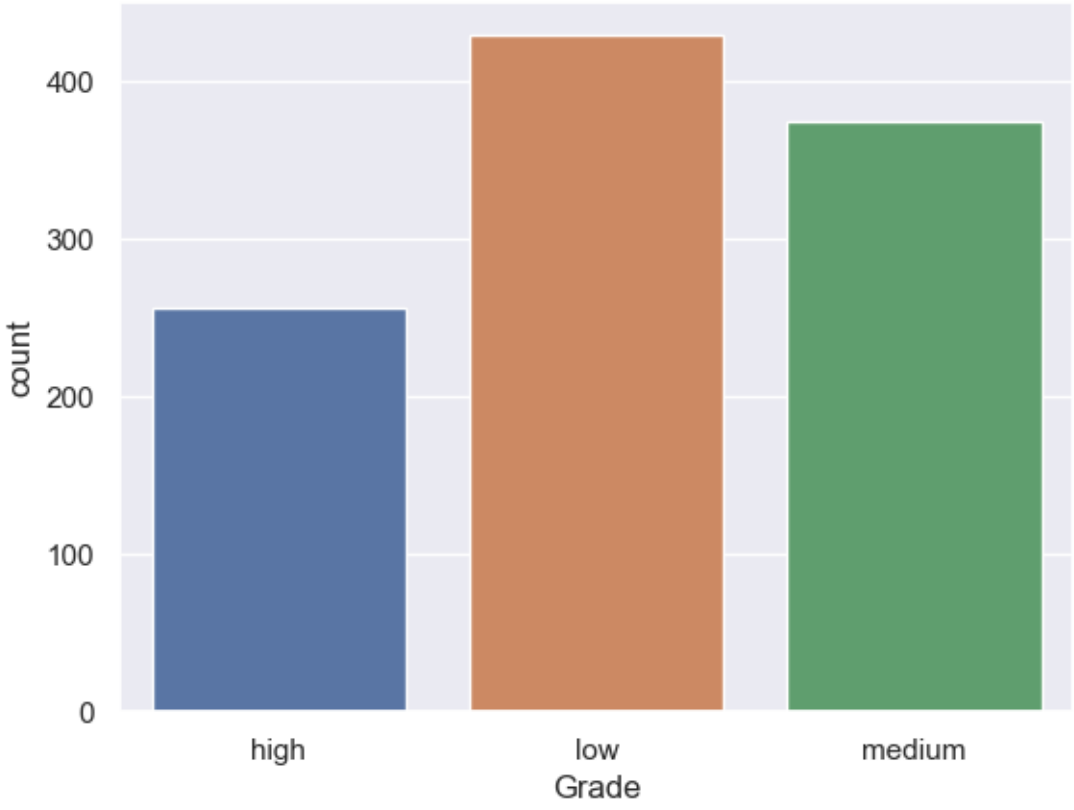
[28]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(3,3))
sns.scatterplot(x='Turbidity', y='pH', hue='Grade', data=milkData)
plt.show()
```



[29]:
```
milkData.Grade.value_counts()
```

[29]:
```
Grade
low       429
medium    374
high      256
Name: count, dtype: int64
```

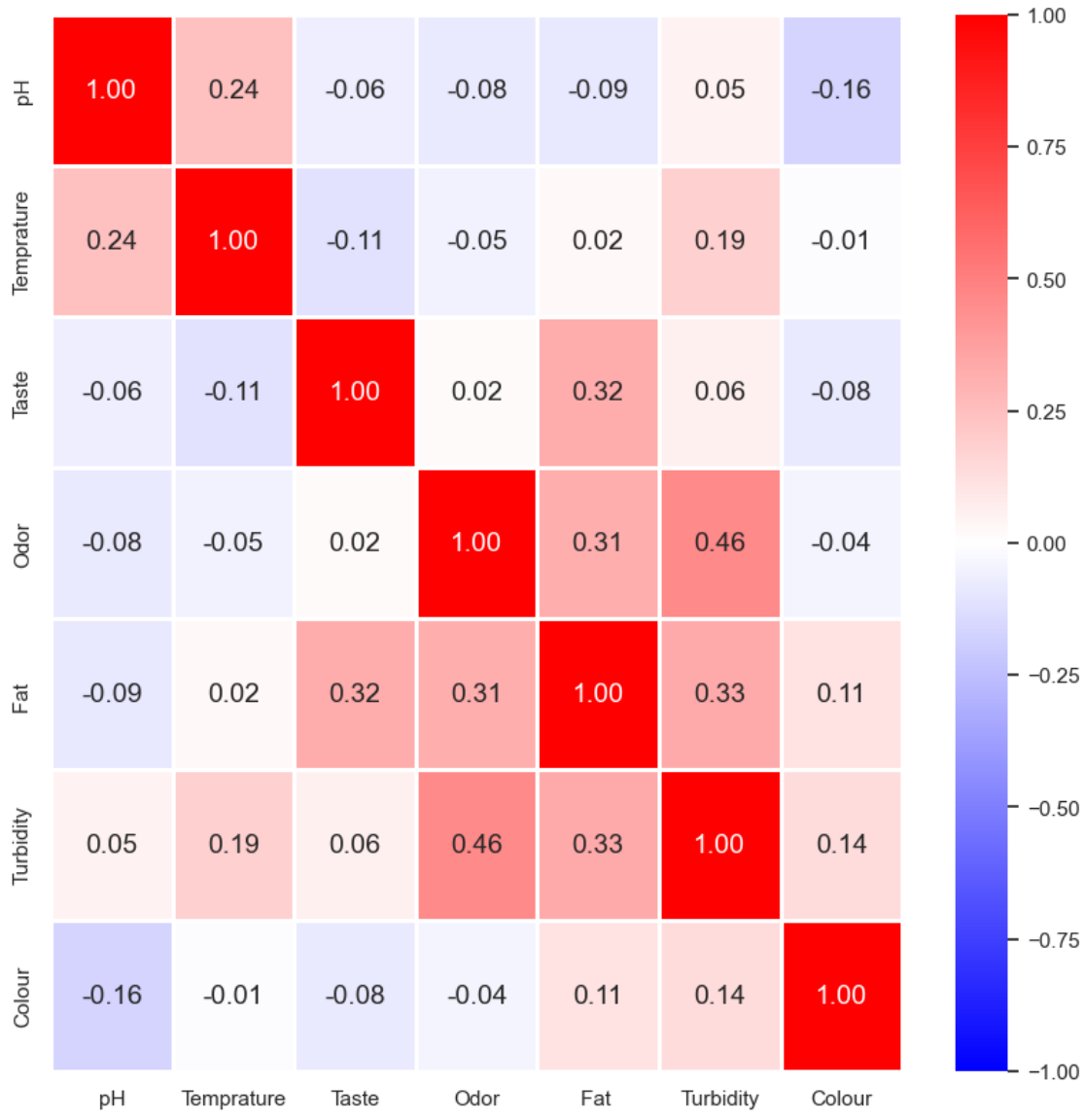
[30]:
```
sb.countplot(x='Grade', data=milkData)
plt.show()
```



[31]:
```
# Exclude non-numeric columns
numeric_data = milkData.select_dtypes(include=['float64', 'int64'])

# Display the correlation matrix
correlation_matrix = numeric_data.corr()
print(correlation_matrix)
```

```
# Plot the heatmap
fig, axes = plt.subplots(1, 1, figsize=(10, 10))
heatmap = sb.heatmap(correlation_matrix, vmin=-1, vmax=1, linewidths=1,
                     annot=True, fmt=".2f", annot_kws={"size": 14}, cmap="bwr")

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

```
                  pH  Temprature     Taste      Odor       Fat   Turbidity  \
pH          1.000000    0.244684 -0.064053 -0.081331 -0.093429   0.048384
Temprature  0.244684    1.000000 -0.109792 -0.048870  0.024073   0.185106
Taste      -0.064053   -0.109792  1.000000  0.017582  0.324149   0.055755
Odor       -0.081331   -0.048870  0.017582  1.000000  0.314505   0.457935
Fat        -0.093429    0.024073  0.324149  0.314505  1.000000   0.329264
Turbidity   0.048384    0.185106  0.055755  0.457935  0.329264   1.000000
Colour     -0.164565   -0.008511 -0.082654 -0.039361  0.114151   0.136436

               Colour
pH         -0.164565
Temprature -0.008511
Taste      -0.082654
Odor       -0.039361
Fat         0.114151
Turbidity   0.136436
Colour      1.000000
```



[32]:
```
pd.DataFrame(milkData.groupby("Odor")["Turbidity"].mean())
```

[32]:

Odor	Turbidity
0	0.291181
1	0.753275

[33]:
```
pd.DataFrame(milkData.groupby('Fat ')['Taste'].mean())
```

[33]:

Fat	Taste
0	0.316092

```
1     0.659634
```

[34]:
```
pd.DataFrame(milkData.groupby('pH')['Temprature'].mean())
```

[34]:
```
      Temprature
pH
3.0   40.000000
4.5   48.702703
4.7   38.000000
5.5   45.000000
5.6   50.000000
6.4   45.000000
6.5   37.746032
6.6   41.125786
6.7   42.951220
6.8   42.273092
7.4   75.897436
8.1   66.000000
8.5   70.000000
8.6   55.000000
9.0   43.000000
9.5   34.000000
```

[35]:
```
#transform Grade in numerical numbers using label encoder
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
milkData['Grade']= label_encoder.fit_transform(milkData['Grade'])
```

[36]:
```
milkData.head()
```

[36]:

	pH	Temprature	Taste	Odor	Fat	Turbidity	Colour	Grade
0	6.6	35	1	0	1	0	254	0
1	6.6	36	0	1	0	1	253	0
2	8.5	70	1	1	1	1	246	1
3	9.5	34	1	1	0	1	255	1
4	6.6	37	0	0	0	0	255	2

[37]:
```
from sklearn.linear_model import LogisticRegression #logistic regression
from sklearn.ensemble import RandomForestClassifier #Random Forest
from sklearn.neighbors import KNeighborsClassifier #KNN
from sklearn.tree import DecisionTreeClassifier #Decision Tree
from sklearn.model_selection import train_test_split #training and testing data␣
 ↪split
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
```

[38]:
```
#splinting into train and test
x= milkData.drop(['Grade'],axis=1)
y= milkData['Grade']

from sklearn.preprocessing import StandardScaler
PredictorScaler=StandardScaler()

PredictorScalerFit=PredictorScaler.fit(x)

x=PredictorScalerFit.transform(x)
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.
 ↪3,random_state=42)
```

[39]:
```
print("X Train : ", X_train.shape)
print("X Test  : ", X_test.shape)
print("Y Train : ", y_train.shape)
print("Y Test  : ", y_test.shape)
```

```
X Train :  (741, 7)
X Test  :  (318, 7)
Y Train :  (741,)
Y Test  :  (318,)
```

[40]:
```
# Import necessary libraries
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# Create a StandardScaler instance
scaler = StandardScaler()

# Fit the scaler to the training data
scaler.fit(X_train)

# Transform the training and test sets
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a logistic regression model on the scaled data
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
```

[40]:
```
LogisticRegression()
```

[41]:
```
#This is for the test dataset
model = LogisticRegression()
model.fit(X_train_scaled,y_train)
X_test_predict_lr =model.predict(X_test)
```

```
X_test_predict_lr
```

[41]:
```
array([1, 1, 1, 1, 2, 1, 1, 0, 0, 0, 1, 2, 0, 2, 1, 1, 1, 2, 2, 2, 0, 2,
       0, 2, 2, 0, 2, 0, 1, 1, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2, 1, 1, 0, 0,
       0, 0, 0, 0, 2, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 2, 2, 1, 1, 0, 0, 0,
       2, 1, 1, 2, 1, 1, 0, 2, 1, 1, 1, 2, 1, 1, 2, 2, 0, 0, 2, 2, 0, 1,
       2, 0, 1, 1, 0, 0, 1, 1, 1, 2, 1, 0, 1, 1, 1, 1, 2, 2, 1, 1, 2, 0,
       1, 2, 2, 1, 0, 1, 2, 1, 2, 0, 2, 1, 1, 1, 2, 1, 2, 2, 0, 2, 1, 2,
       1, 2, 1, 1, 2, 1, 2, 2, 2, 0, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 0, 1,
       2, 0, 1, 0, 1, 0, 2, 2, 2, 1, 2, 0, 1, 1, 2, 1, 0, 0, 1, 1, 0, 1,
       2, 2, 2, 2, 1, 2, 0, 1, 0, 1, 2, 0, 2, 1, 1, 2, 0, 1, 0, 1, 1, 1,
       2, 2, 0, 2, 0, 0, 2, 1, 2, 0, 2, 2, 0, 0, 2, 2, 0, 1, 1, 0, 2, 1,
       2, 2, 0, 0, 1, 0, 1, 0, 2, 0, 0, 1, 2, 0, 0, 1, 2, 2, 2, 1, 1, 0,
       1, 1, 1, 0, 1, 2, 1, 1, 1, 0, 2, 0, 2, 2, 1, 2, 1, 0, 2, 0, 0, 0,
       2, 1, 0, 0, 2, 0, 2, 2, 1, 0, 2, 1, 1, 1, 2, 0, 2, 0, 1, 0, 1, 1,
       1, 2, 0, 2, 1, 1, 2, 0, 2, 1, 0, 0, 2, 1, 2, 0, 1, 2, 1, 1, 1, 0,
       2, 2, 2, 0, 2, 2, 2, 1, 1, 0])
```

[42]:
```
#This is for the test dataset
model = LogisticRegression()
model.fit(X_train_scaled,y_train)
X_test_predict_lr =model.predict(X_test)
print('The accuracy for Logistic Regression model is (Test Dataset) ', metrics.
 ↪accuracy_score(X_test_predict_lr,y_test))
#This is for the train dataset
model = LogisticRegression()
model.fit(X_test_scaled,y_test)
X_train_predict_lr =model.predict(X_train)
print('The accuracy for Logistic Regression model is (Train Dataset) ',metrics.
 ↪accuracy_score(X_train_predict_lr,y_train))
```

```
The accuracy for Logistic Regression model is (Test Dataset)  0.8522012578616353
The accuracy for Logistic Regression model is (Train Dataset)
0.8259109311740891
```

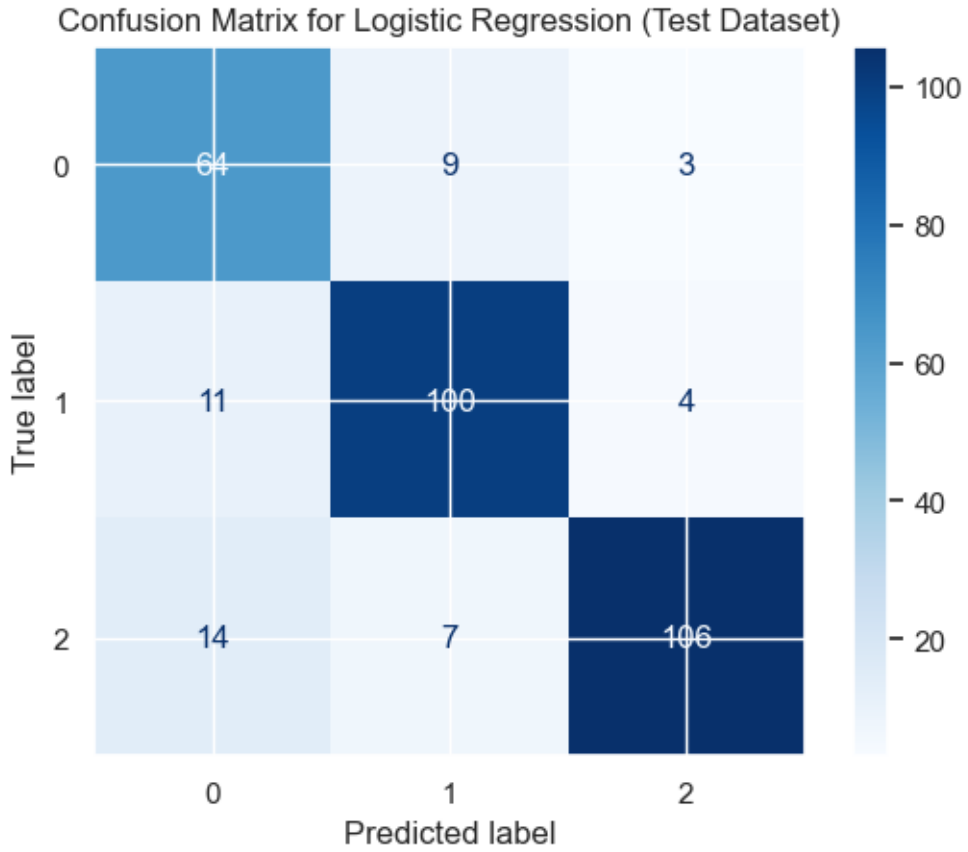
[43]:
```
from sklearn.metrics import classification_report

# Assuming 'model_lr' is the Logistic Regression model and 'model_rfc' is the␣
 ↪Random Forest model
# (Ensure that these models have been trained before running this code)

# Split the data into training and testing sets
x = milkData.drop(['Grade'], axis=1)
y = milkData['Grade']

# Standardize the features
PredictorScaler = StandardScaler()
```

```
PredictorScalerFit = PredictorScaler.fit(x)
x = PredictorScalerFit.transform(x)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=42)

# Train the Logistic Regression model
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)



# Predictions on the testing set
y_pred_lr = model_lr.predict(X_test)

# Logistic Regression classification report
print("Classification Report for Logistic Regression (Test Dataset):")
print(classification_report(y_test, y_pred_lr))
```

```
Classification Report for Logistic Regression (Test Dataset):
              precision    recall  f1-score   support

           0       0.72      0.84      0.78        76
           1       0.86      0.87      0.87       115
           2       0.94      0.83      0.88       127

    accuracy                           0.85       318
   macro avg       0.84      0.85      0.84       318
weighted avg       0.86      0.85      0.85       318
```

[44]:
```
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming 'y_test' and 'y_pred_lr' are the true labels and predicted labels,␣
 ↪respectively

# Create confusion matrix
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)

# Display confusion matrix using ConfusionMatrixDisplay
disp_lr = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_lr,␣
 ↪display_labels=model_lr.classes_)
disp_lr.plot(cmap='Blues', values_format='d')
```

```
plt.title('Confusion Matrix for Logistic Regression (Test Dataset)')
plt.show()
```

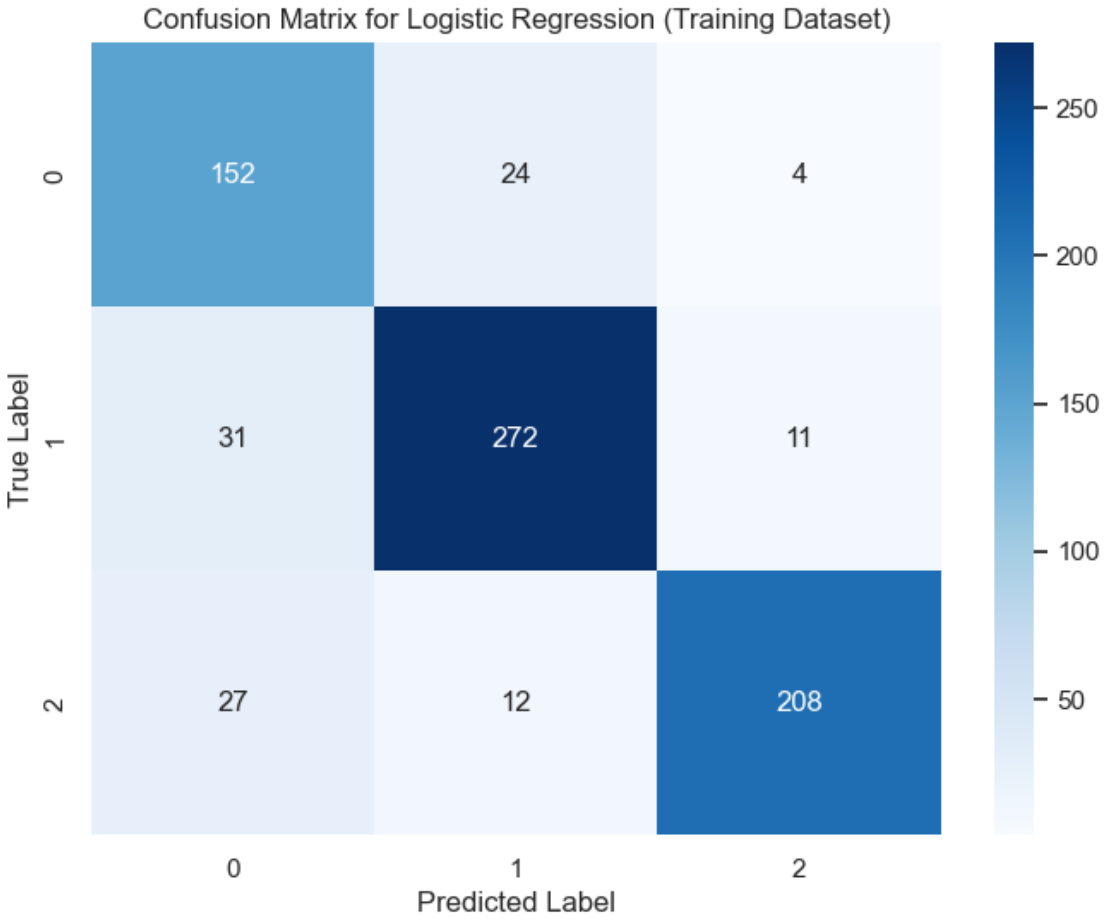


[45]:
```
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Assuming 'model_lr' is the Logistic Regression model
# Ensure that 'model_lr' has been trained before running this code

# Split the data into training and testing sets
x = milkData.drop(['Grade'], axis=1)
y = milkData['Grade']

# Standardize the features
PredictorScaler = StandardScaler()
PredictorScalerFit = PredictorScaler.fit(x)
x = PredictorScalerFit.transform(x)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=42)

# Train the Logistic Regression model
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)

# Predictions on the training set
y_pred_lr_train = model_lr.predict(X_train)

# Logistic Regression classification report for training dataset
print("Classification Report for Logistic Regression (Training Dataset):")
print(classification_report(y_train, y_pred_lr_train))
```

```
Classification Report for Logistic Regression (Training Dataset):
              precision    recall  f1-score   support

           0       0.72      0.84      0.78       180
           1       0.88      0.87      0.87       314
           2       0.93      0.84      0.89       247

    accuracy                           0.85       741
   macro avg       0.85      0.85      0.85       741
weighted avg       0.86      0.85      0.85       741
```

[46]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the confusion matrix
conf_matrix_train = confusion_matrix(y_train, y_pred_lr_train)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_train, annot=True, fmt='d', cmap='Blues',
            xticklabels=model_lr.classes_, yticklabels=model_lr.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Logistic Regression (Training Dataset)')
plt.show()
```



[47]:
```
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
# Make predictions on the test dataset
X_test_predict_rfc = model.predict(X_test)
X_test_predict_rfc
```

[47]:
```
array([0, 1, 1, 2, 2, 1, 1, 0, 1, 2, 1, 2, 0, 2, 1, 1, 0, 2, 2, 2, 1, 2,
       0, 2, 2, 2, 0, 0, 1, 1, 2, 0, 0, 2, 0, 2, 2, 0, 2, 2, 1, 1, 1, 2,
       0, 1, 0, 0, 2, 0, 2, 1, 1, 2, 2, 1, 0, 1, 1, 2, 2, 1, 0, 0, 0, 1,
       2, 2, 1, 2, 1, 1, 0, 2, 1, 1, 1, 2, 1, 2, 2, 2, 0, 0, 2, 2, 2, 1,
       2, 0, 1, 1, 0, 2, 1, 2, 0, 0, 1, 0, 1, 1, 1, 1, 2, 2, 1, 1, 2, 0,
       0, 2, 2, 1, 0, 1, 2, 1, 1, 0, 2, 1, 1, 1, 2, 0, 2, 2, 0, 2, 1, 2,
       1, 2, 1, 1, 2, 1, 2, 2, 2, 0, 1, 2, 1, 1, 0, 1, 1, 2, 2, 2, 2, 1,
       2, 1, 1, 0, 1, 0, 2, 2, 2, 1, 2, 0, 1, 1, 2, 1, 2, 0, 1, 1, 0, 1,
       2, 2, 2, 2, 1, 2, 0, 1, 0, 1, 2, 0, 2, 1, 1, 2, 2, 1, 0, 1, 1, 0,
       2, 2, 2, 2, 0, 0, 2, 1, 2, 2, 2, 2, 0, 0, 2, 2, 0, 1, 1, 0, 2, 1,
       2, 2, 0, 0, 1, 0, 1, 1, 2, 0, 0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 0,
       0, 1, 1, 0, 1, 2, 1, 1, 1, 0, 2, 0, 2, 2, 1, 2, 1, 1, 2, 0, 0, 0,
```

```
       2, 1, 0, 0, 2, 2, 2, 2, 1, 0, 2, 1, 1, 1, 2, 0, 2, 0, 2, 0, 1, 1,
       1, 2, 0, 2, 1, 1, 2, 0, 2, 2, 0, 0, 2, 1, 2, 1, 0, 2, 1, 1, 1, 0,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 0])
```

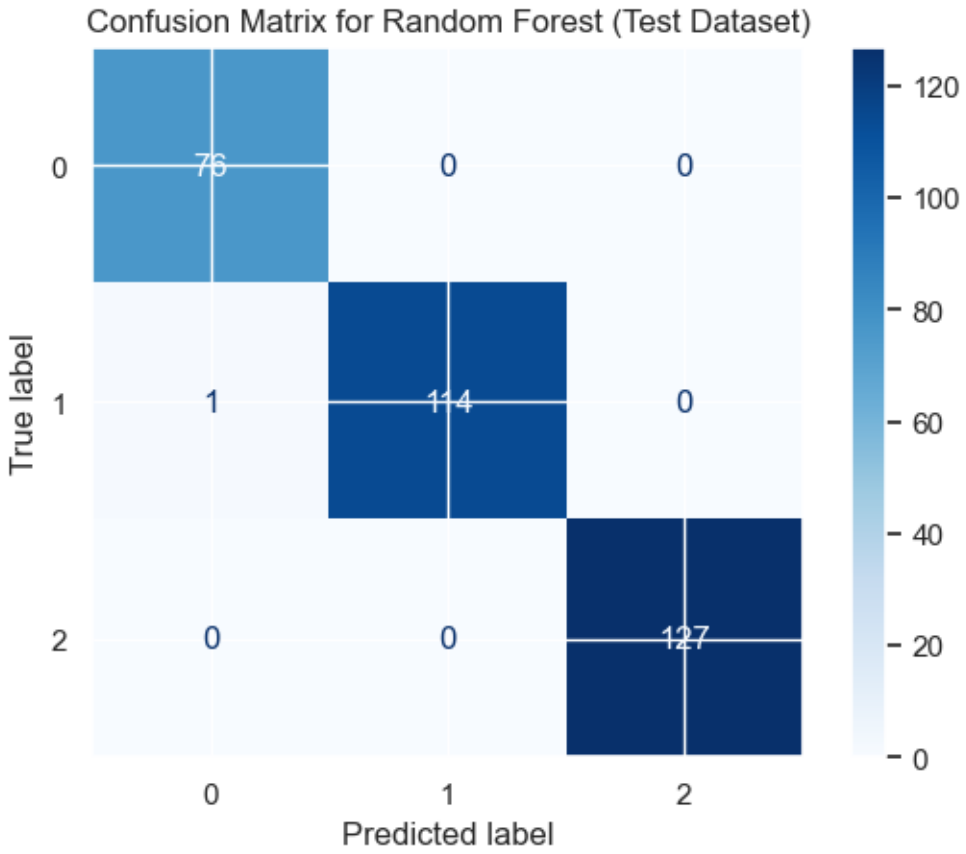
[48]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
X_test_predict_rfc = model.predict(X_test)
print('The accuracy of the Random Forests model is (Test Dataset)',␣
 ↪accuracy_score(X_test_predict_rfc, y_test))
model_train = RandomForestClassifier(n_estimators=100)
model_train.fit(X_test, y_test)
X_train_predict_rfc = model_train.predict(X_train)
print('The accuracy of the Random Forests model is (Train Dataset)',␣
 ↪accuracy_score(X_train_predict_rfc, y_train))
```

```
The accuracy of the Random Forests model is (Test Dataset) 0.9968553459119497
The accuracy of the Random Forests model is (Train Dataset) 0.9986504723346828
```

[49]:
```
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

# Predict on the test set
y_test_predict_rfc = model.predict(X_test)



# Create confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_predict_rfc)

# Print Classification Report
print('Classification Report for Random Forest (Test Dataset):\n',␣
 ↪classification_report(y_test, y_test_predict_rfc))
```

```
Classification Report for Random Forest (Test Dataset):
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        76
           1       1.00      0.99      1.00       115
           2       1.00      1.00      1.00       127

    accuracy                           1.00       318
   macro avg       1.00      1.00      1.00       318
weighted avg       1.00      1.00      1.00       318
```

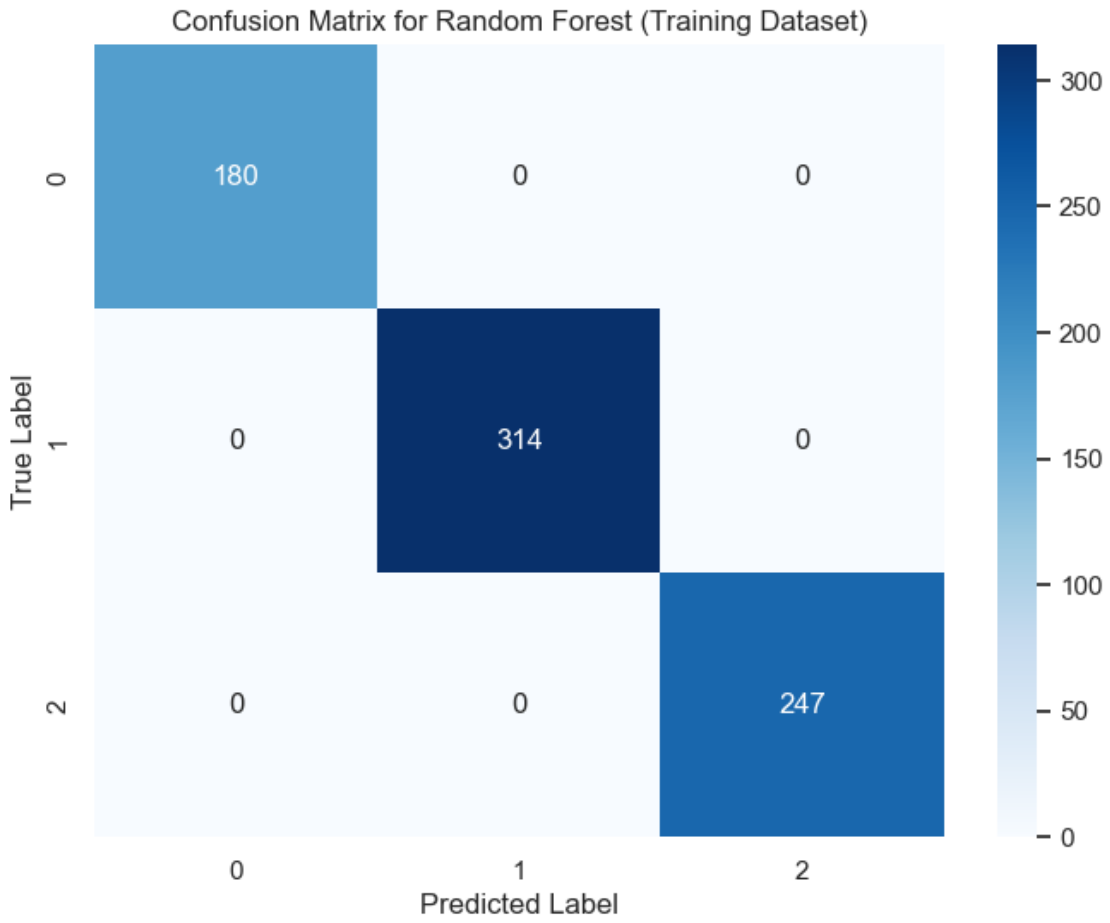
[50]:
```
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming 'y_test' and 'y_test_predict_rfc' are the true labels and predicted␣
 ↪labels, respectively

# Create confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_predict_rfc)

# Display confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,␣
 ↪display_labels=model.classes_)
disp.plot(cmap='Blues', values_format='d')

plt.title('Confusion Matrix for Random Forest (Test Dataset)')
plt.show()
```



[51]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Predict on the training set
y_train_predict_rfc = model.predict(X_train)

# Create confusion matrix for training dataset
conf_matrix_train = confusion_matrix(y_train, y_train_predict_rfc)

# Print Classification Report for training dataset
print('Classification Report for Random Forest (Training Dataset):\n',␣
 ↪classification_report(y_train, y_train_predict_rfc))
```

```
Classification Report for Random Forest (Training Dataset):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       180
           1       1.00      1.00      1.00       314
           2       1.00      1.00      1.00       247

    accuracy                           1.00       741
   macro avg       1.00      1.00      1.00       741
weighted avg       1.00      1.00      1.00       741
```

[52]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the confusion matrix for the training dataset
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_train, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y_train), yticklabels=np.unique(y_train))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Random Forest (Training Dataset)')
plt.show()
```



[53]:
```
model = DecisionTreeClassifier()
model.fit(X_train,y_train)
X_test_predict_dtc = model.predict(X_test)
print('The accuracy for Decision Tree Classifier model is (Test Dataset) ',␣
 ↪metrics.accuracy_score(X_test_predict_dtc,y_test))
model = RandomForestClassifier(n_estimators=100)
model.fit(X_test,y_test)
X_train_predict =model.predict(X_train)
print('The accuracy for Decision Tree Classifier model is  (Train Dataset)␣
 ↪',metrics.accuracy_score(X_train_predict,y_train))
```

```
The accuracy for Decision Tree Classifier model is (Test Dataset)
0.9937106918238994
The accuracy for Decision Tree Classifier model is  (Train Dataset)
0.9986504723346828
```

[54]:
```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn import tree
import matplotlib.pyplot as plt

# Train a decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Visualize the decision tree
plt.figure(figsize=(20, 20))
tree.plot_tree(clf, filled=True)
plt.show()
```

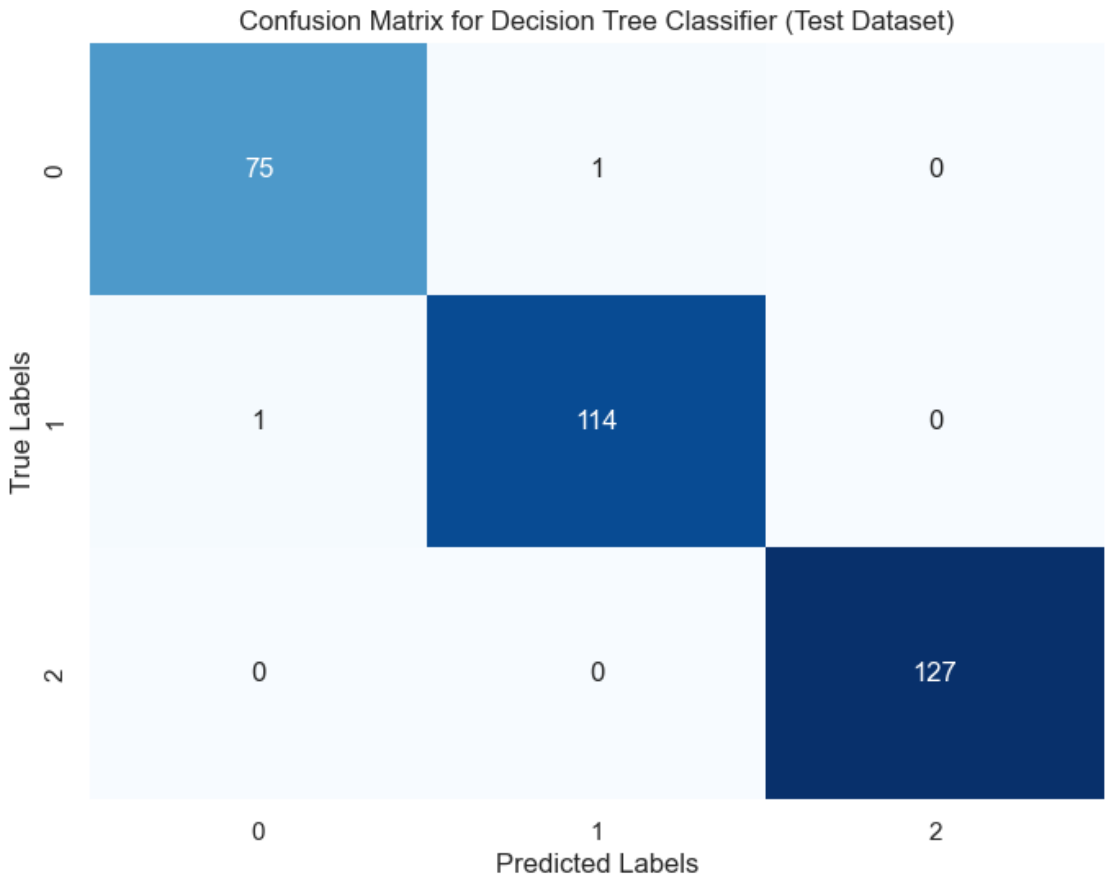
[ ]:

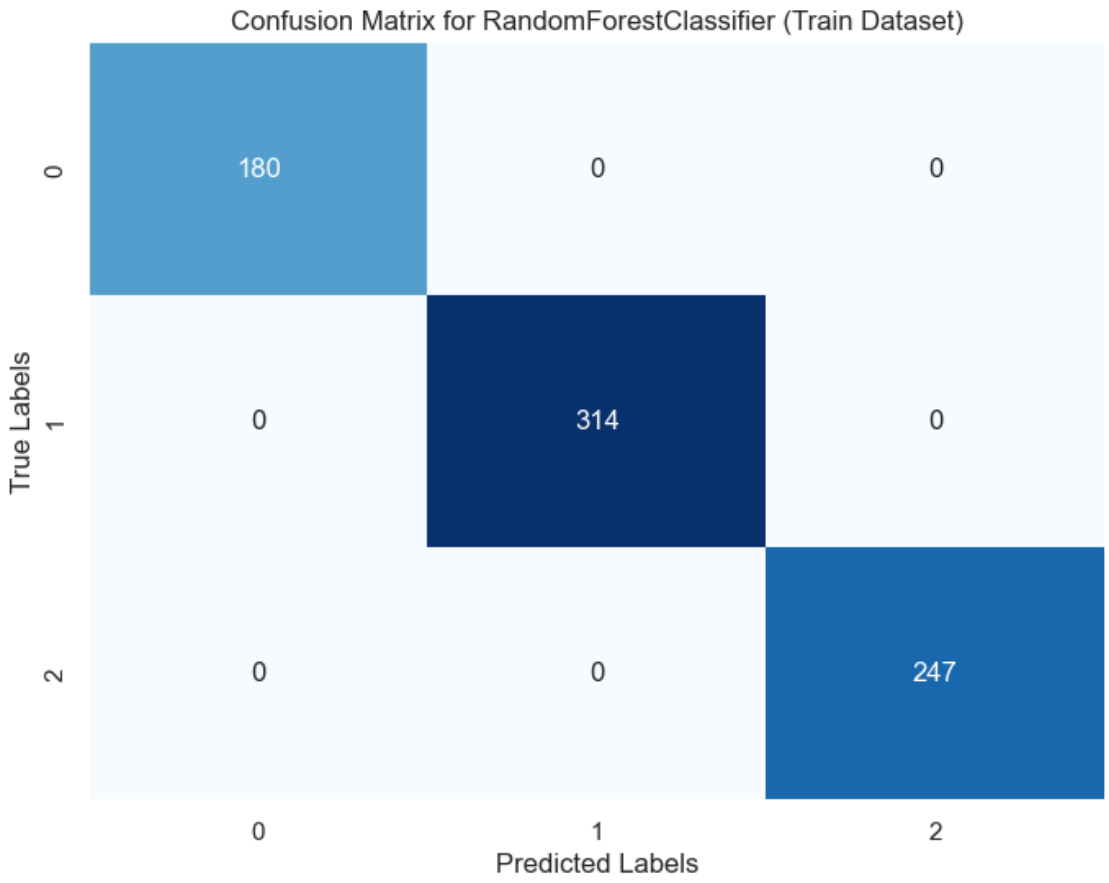
[56]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

# For Decision Tree Classifier
model_dtc = DecisionTreeClassifier()
model_dtc.fit(X_train, y_train)
X_test_predict_dtc = model_dtc.predict(X_test)
print('The accuracy for Decision Tree Classifier model is (Test Dataset): ',␣
 ↪metrics.accuracy_score(X_test_predict_dtc, y_test))

# Classification report for test dataset
from sklearn.metrics import classification_report
print("Classification Report for Test Dataset:")
print(classification_report(y_test, X_test_predict_dtc))

# For RandomForestClassifier
model_rf = RandomForestClassifier(n_estimators=100)
model_rf.fit(X_train, y_train)
X_train_predict_rf = model_rf.predict(X_train)
print('The accuracy for RandomForestClassifier model is (Train Dataset): ',␣
 ↪metrics.accuracy_score(X_train_predict_rf, y_train))

# Classification report for train dataset
print("Classification Report for Train Dataset:")
print(classification_report(y_train, X_train_predict_rf))
```

```
The accuracy for Decision Tree Classifier model is (Test Dataset):
0.9937106918238994
Classification Report for Test Dataset:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        76
           1       0.99      0.99      0.99       115
           2       1.00      1.00      1.00       127

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318

The accuracy for RandomForestClassifier model is (Train Dataset):  1.0
Classification Report for Train Dataset:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       180
           1       1.00      1.00      1.00       314
           2       1.00      1.00      1.00       247
```

```
    accuracy                           1.00       741
   macro avg       1.00      1.00      1.00       741
weighted avg       1.00      1.00      1.00       741
```

[57]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# For Decision Tree Classifier
model_dtc = DecisionTreeClassifier()
model_dtc.fit(X_train, y_train)
X_test_predict_dtc = model_dtc.predict(X_test)

# Confusion matrix for test dataset
conf_matrix_test = confusion_matrix(y_test, X_test_predict_dtc)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=model_dtc.classes_, yticklabels=model_dtc.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Decision Tree Classifier (Test Dataset)')
plt.show()

# For RandomForestClassifier
model_rf = RandomForestClassifier(n_estimators=100)
model_rf.fit(X_train, y_train)
X_train_predict_rf = model_rf.predict(X_train)

# Confusion matrix for train dataset
conf_matrix_train = confusion_matrix(y_train, X_train_predict_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_train, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=model_rf.classes_, yticklabels=model_rf.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for RandomForestClassifier (Train Dataset)')
plt.show()
```


Confusion Matrix for Decision Tree Classifier (Test Dataset)
True Labels
Predicted Labels
0
1
2
75
1
0
1
114
0
0
0
127



[ ]:

[58]:
```
model = KNeighborsClassifier()
model.fit(X_train,y_train)
X_test_predict_knn = model.predict(X_test)
print('The accuracy for KNN classifier model is (Test Dataset)', metrics.
 ↪accuracy_score(X_test_predict_knn,y_test))
model = KNeighborsClassifier()
model.fit(X_test,y_test)
X_train_predict_knn =model.predict(X_train)
print('The accuracy for KNN classifier model is (Train Dataset) ',metrics.
 ↪accuracy_score(X_train_predict_knn,y_train))
```

```
The accuracy for KNN classifier model is (Test Dataset) 0.9937106918238994
The accuracy for KNN classifier model is (Train Dataset)  0.9568151147098516
```

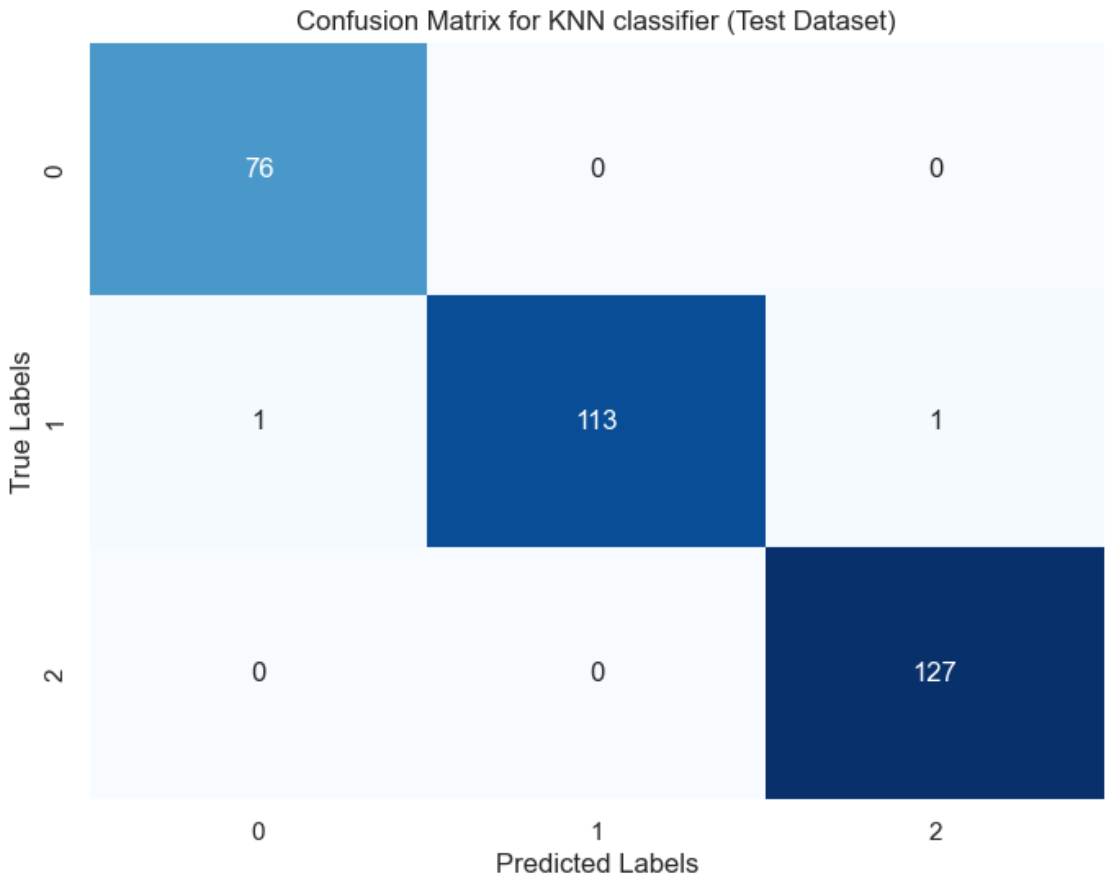
[59]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
```

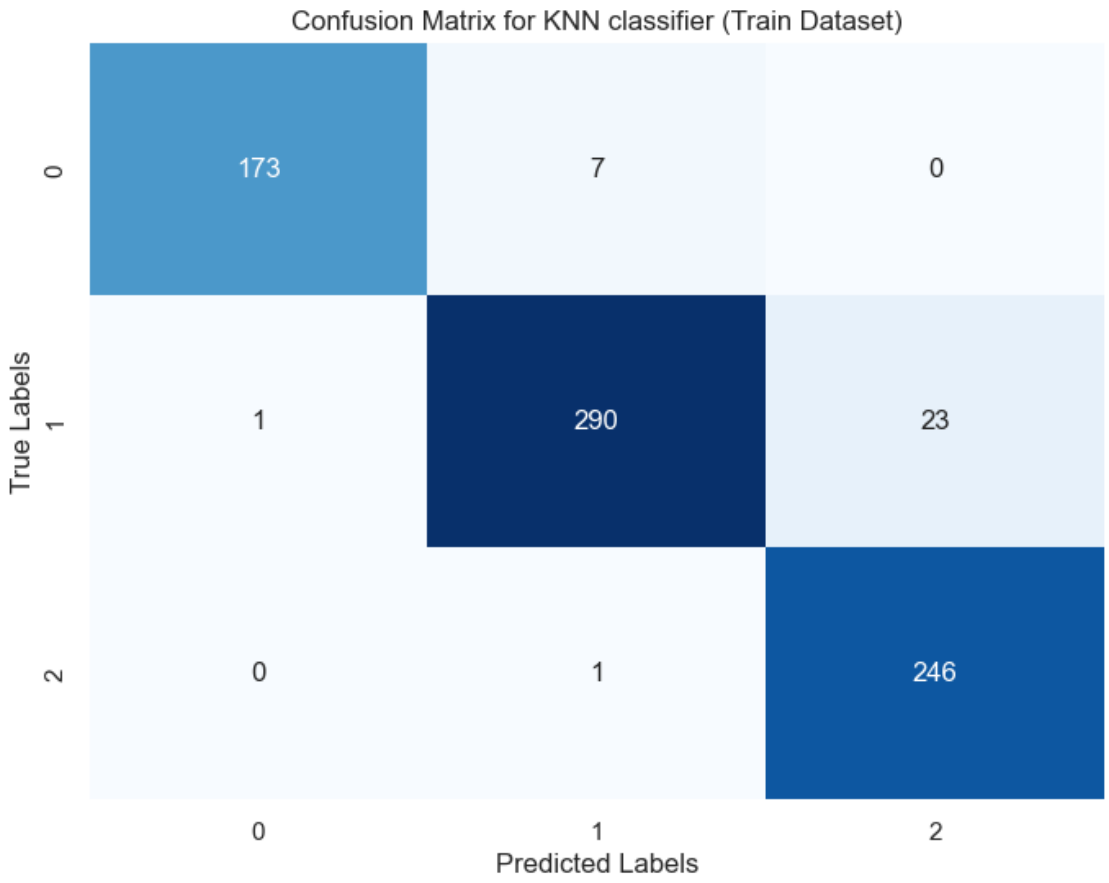
```
# For KNN classifier
model_knn = KNeighborsClassifier()
model_knn.fit(X_train, y_train)
X_test_predict_knn = model_knn.predict(X_test)

# Classification report for test dataset
print("Classification Report for Test Dataset:")
print(classification_report(y_test, X_test_predict_knn))

# For the train dataset
model_knn_train = KNeighborsClassifier()
model_knn_train.fit(X_test, y_test)
X_train_predict_knn = model_knn_train.predict(X_train)

# Classification report for train dataset
print("Classification Report for Train Dataset:")
print(classification_report(y_train, X_train_predict_knn))
```

```
Classification Report for Test Dataset:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        76
           1       1.00      0.98      0.99       115
           2       0.99      1.00      1.00       127

    accuracy                           0.99       318
   macro avg       0.99      0.99      0.99       318
weighted avg       0.99      0.99      0.99       318

Classification Report for Train Dataset:
              precision    recall  f1-score   support

           0       0.99      0.96      0.98       180
           1       0.97      0.92      0.95       314
           2       0.91      1.00      0.95       247

    accuracy                           0.96       741
   macro avg       0.96      0.96      0.96       741
weighted avg       0.96      0.96      0.96       741
```

[60]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# For KNN classifier
```

```
model_knn = KNeighborsClassifier()
model_knn.fit(X_train, y_train)
X_test_predict_knn = model_knn.predict(X_test)

# Confusion matrix for test dataset
conf_matrix_test = confusion_matrix(y_test, X_test_predict_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_test, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=model_knn.classes_, yticklabels=model_knn.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for KNN classifier (Test Dataset)')
plt.show()

# For the train dataset
model_knn_train = KNeighborsClassifier()
model_knn_train.fit(X_test, y_test)
X_train_predict_knn = model_knn_train.predict(X_train)

# Confusion matrix for train dataset
conf_matrix_train = confusion_matrix(y_train, X_train_predict_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_train, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=model_knn_train.classes_, yticklabels=model_knn_train.
 ↪classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for KNN classifier (Train Dataset)')
plt.show()
```

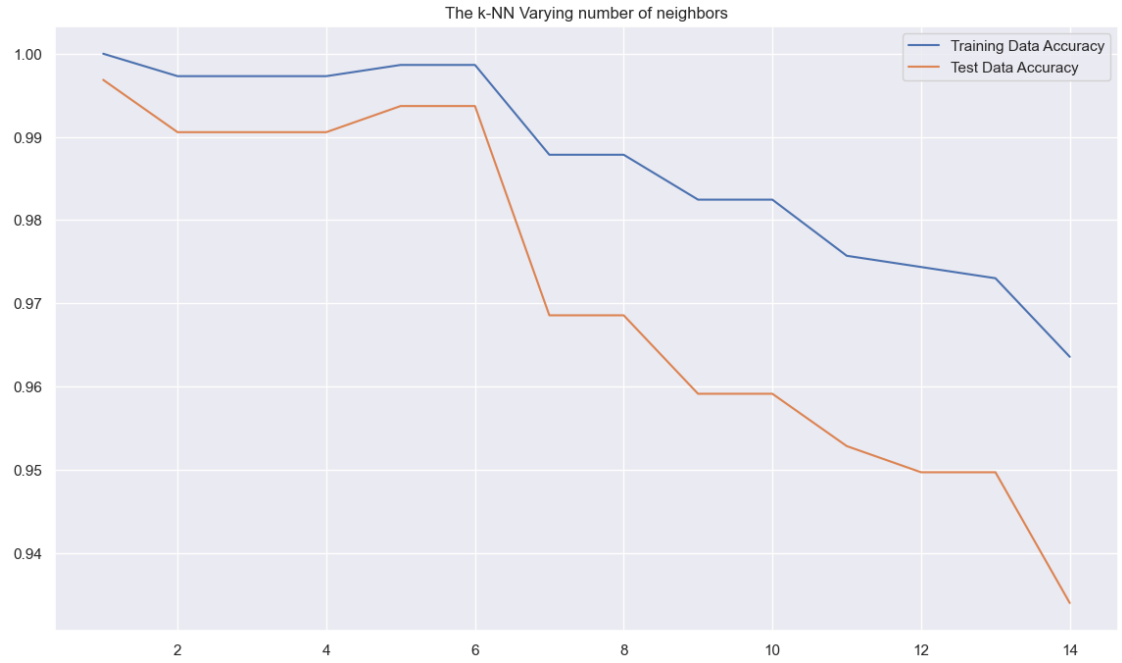

Confusion Matrix for KNN classifier (Test Dataset)
True Labels
Predicted Labels
0
1
2
76
0
0
1
113
1
0
0
127
0
1
2



[61]:
```
test_acc=[]
train_acc=[]

for i in range(1,15):
  knn = KNeighborsClassifier(i) #setting up a knn classifier
  knn.fit(X_train,y_train) #fitting the model
  # computing the accuracy for both the trainig and the test data
  train_acc.append(knn.score(X_train,y_train))
  test_acc.append(knn.score(X_test,y_test))
```

[62]:
```
import matplotlib.pyplot as plt
import seaborn as sb

plt.figure(figsize=(14, 8))
plt.title('The k-NN Varying number of neighbors')

# Assuming train_acc and test_acc are lists or arrays containing your data
sb.lineplot(x=range(1, 15), y=train_acc, label='Training Data Accuracy')
sb.lineplot(x=range(1, 15), y=test_acc, label='Test Data Accuracy')
```

```
plt.show()
```


The k-NN Varying number of neighbors
Training Data Accuracy
Test Data Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
2
4
6
8
10
12
14

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: