-----------------------------GitHub REPO link---------------------------------

[https://github.com/CAIS380-ML-S24/hw03-sshaw (https://github.com/CAIS380-ML-S24/hw03-sshaw)](https://github.com/CAIS380-ML-S24/hw03-sshaw)

In [3]:

```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
summarize)
```

In [4]:

```
from ISLP import confusion_table
from ISLP.models import contrast
from sklearn.discriminant_analysis import \
(LinearDiscriminantAnalysis as LDA,
QuadraticDiscriminantAnalysis as QDA)
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [7]:

```
camera = pd.read_csv('camera_dataset.csv')
camera.head
```

Out[7]:

```
<bound method NDFrame.head of                       Model  Release date
Max resolution  Low resolution  \
0            Agfa ePhoto 1280          1997          1024.0            640.
0
1            Agfa ePhoto 1680          1998          1280.0            640.
0
2            Agfa ePhoto CL18          2000           640.0              0.
0
3            Agfa ePhoto CL30          1999          1152.0            640.
0
4      Agfa ePhoto CL30 Clik!          1999          1152.0            640.
0
...                       ...           ...             ...
...
1033          Toshiba PDR-M65          2001          2048.0           1024.
0
1034          Toshiba PDR-M70          2000          2048.0           1024.
0
1035          Toshiba PDR-M71          2001          2048.0           1024.
0
1036          Toshiba PDR-M81          2001          2400.0           1200.
0
1037          Toshiba PDR-T10          2002          1600.0            800.
0

      Effective pixels  Zoom wide (W)  Zoom tele (T)  Normal focus range
\
0                  0.0           38.0          114.0                70.0
1                  1.0           38.0          114.0                50.0
2                  0.0           45.0           45.0                 0.0
3                  0.0           35.0           35.0                 0.0
4                  0.0           43.0           43.0                50.0
...                ...            ...            ...                 ...
1033               3.0           38.0          114.0                10.0
1034               3.0           35.0          105.0                80.0
1035               3.0           35.0           98.0                80.0
1036               3.0           35.0           98.0                80.0
1037               1.0           38.0           38.0                40.0

      Macro focus range  Storage included  Weight (inc. batteries)  \
0                  40.0               4.0                    420.0
1                   0.0               4.0                    420.0
2                   0.0               2.0                      0.0
3                   0.0               4.0                      0.0
4                   0.0              40.0                    300.0
...                 ...               ...                      ...
1033               10.0               8.0                    320.0
1034                9.0              16.0                    390.0
1035               10.0               8.0                    340.0
1036               10.0              16.0                    340.0
1037               20.0               8.0                    180.0

      Dimensions  Price
0           95.0  179.0
1          158.0  179.0
2            0.0  179.0
3            0.0  269.0
```

```
4          128.0  1299.0
...          ...     ...
1033       120.0    62.0
1034       116.0    62.0
1035       107.0    62.0
1036       107.0    62.0
1037        86.0   129.0

[1038 rows x 13 columns]>
```

In [8]:
```
camera.columns
```

Out[8]:
```
Index(['Model', 'Release date', 'Max resolution', 'Low resolution',
       'Effective pixels', 'Zoom wide (W)', 'Zoom tele (T)',
       'Normal focus range', 'Macro focus range', 'Storage included',
       'Weight (inc. batteries)', 'Dimensions', 'Price'],
      dtype='object')
```

adding Prange

In [9]:
```
camera['pRange'] = pd.cut(camera['Price'],
                             bins=[-float('inf'), 150, 399, float('inf')],
                             labels=['low', 'medium', 'high'])
print(camera['pRange'])
```

```
0       medium
1       medium
2       medium
3       medium
4         high
         ...
1033       low
1034       low
1035       low
1036       low
1037       low
Name: pRange, Length: 1038, dtype: category
Categories (3, object): ['low' < 'medium' < 'high']
```

In [10]:
```
camera['Release date'] = pd.to_numeric(camera['Release date'], errors='coe
```

In [11]:
```
#Training data
train_df = camera[(camera['Release date'] >= 1994) & (camera['Release date
#Testing data
test_df = camera[(camera['Release date'] >= 2005) & (camera['Release date'
```

In [12]:
```
from sklearn.preprocessing import LabelEncoder
```

In [13]:
```
y_train = LabelEncoder().fit_transform(train_df['pRange'])
y_test = LabelEncoder().fit_transform(test_df['pRange'])
```

In [14]:
```
X_train = train_df.drop(columns=['Model', 'Price', 'Release date', 'pRange
X_test = test_df.drop(columns=['Model', 'Price', 'Release date', 'pRange']
```

In [15]:
```
 lda = LDA(store_covariance=True)
```

In [16]:
```
lda.fit(X_train,y_train)
```

Out[16]:
```
LinearDiscriminantAnalysis

LinearDiscriminantAnalysis(store_covariance=True)
```

In [17]:
```
X_train.fillna(X_train.mean(), inplace=True)
X_test.fillna(X_test.mean(), inplace=True)
```

In [18]:
```
y_pred_lda = lda.predict(X_test)
```

In [19]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [20]:
```
conf_matrix_lda = confusion_matrix(y_test, y_pred_lda)
overall_accuracy_lda = accuracy_score(y_test, y_pred_lda)
overall_error_rate_lda = 1 - overall_accuracy_lda

conf_matrix_lda, overall_error_rate_lda
```

Out[20]:
```
(array([[ 49,   2,  35],
        [ 28,   9, 124],
        [ 20,  15, 177]], dtype=int64),
 0.4880174291938998)
```

In [ ]:

In [ ]:

------LDA---------

- **Low Price Range Error Percentage**: (2 + 35) / 86 43.02%
- **Medium Price Range Error Percentage**: (28 + 124) / 161 94.41%
- **High Price Range Error Percentage**: (20 + 15) / 212 16.51%

Overall Error Rate: 48.8%

The error seems to be particularly bad within the medium price range, with an error percentage of 94.41%. This indicates that the model struggles significantly with correctly classifying cameras in the medium price range, misclassifying them as either low or high in the vast majority of cases. The high price range sees the least error, suggesting the model is relatively more successful at identifying cameras in this category

In [21]:
```
qda = QDA(store_covariance=True)
qda.fit(X_train, y_train)
```

Out[21]:
```
QuadraticDiscriminantAnalysis

QuadraticDiscriminantAnalysis(store_covariance=True)
```

In [22]:
```
qda_pred = qda.predict(X_test)
```

In [23]:
```
conf_matrix_qda = confusion_matrix(y_test, qda_pred)
overall_accuracy_qda = accuracy_score(y_test, qda_pred)
overall_error_rate_qda = 1 - overall_accuracy_qda

conf_matrix_qda, overall_error_rate_qda
```

Out[23]:
```
(array([[50, 19, 17],
        [19, 63, 79],
        [27, 94, 91]], dtype=int64),
 0.5555555555555556)
```

----------QDA------------

- **Low Price Range Error Percentage**: (19+17) / (50+19+17) 41.86%
- **Medium Price Range Error Percentage**: (19+79) / (19+63+79) 60.87%
- **High Price Range Error Percentage**: (27+94) / (27+94+91) 57.08%

OVERALL - 55.6%

The QDA model does a bit better than the LDA model, especially when predicting cameras in the medium price range. While the LDA model had a lot of trouble with these medium-priced cameras, the QDA model still struggles but not as much. Both models find it hard to accurately classify cameras by their price ranges, showing that choosing the right model is important, but so are other things like picking the best features and having good data.

In [24]:
```
NB = GaussianNB()
NB.fit(X_train, y_train)
```

Out[24]:
```
GaussianNB

GaussianNB()
```

In [25]:
```
nb_pred = NB.predict(X_test)
```

In [26]:
```
conf_matrix_nb = confusion_matrix(y_test, nb_pred)
overall_accuracy_nb = accuracy_score(y_test, nb_pred)
overall_error_rate_nb = 1 - overall_accuracy_nb

conf_matrix_nb, overall_error_rate_nb
```

Out[26]:
```
(array([[ 67,   2,  17],
        [ 58,   2, 101],
        [ 93,   1, 118]], dtype=int64),
 0.5925925925925926)
```

- **Low Price Range Error Percentage**:(2 + 17) / 86 22.06%
- **Medium Price Range Error Percentage**: (58 + 101) / 161 98.76%
- **High Price Range Error Percentage**:(93 + 1) / 212 44.34%

OVERALL - 59.26%

In the Naive Bayes model, the error is most noticeable in the medium price range, similar to what we observed with the LDA and QDA models. However, the Naive Bayes model demonstrates a particularly high error rate in this category, suggesting it also struggles significantly with medium-priced cameras. Compared to the LDA and QDA models, the pattern of difficulty with the medium price range persists across all models, indicating a consistent challenge in accurately classifying cameras in this price bracket. This shows that regardless of the model used, predicting the medium price range accurately remains a tough problem, highlighting potential issues with the features used for modeling or the inherent complexity of the data in this specific price range.

For the camera dataset, Naive Bayes might not be ideal because it assumes features like megapixels and zoom don't affect each other, which is likely not true here. LDA could be better if camera features across different price ranges behave similarly, but if these features vary significantly by price range, QDA might be the best since it allows for such differences. However, if our camera features vary widely by price range, theoretically, QDA should have been the most accurate. The real measure of success, though, depends on how well each model's assumptions align with our specific dataset and whether the model that theoretically fits the best actually delivered the best performance based on our data.

In [ ]:
```

```