Machine Learning Applications for Business

Individual Assignment 1

Submitted by
Ankit Dwivedi (M22MS093)
MBA’24

Course Instructor
Dr. Bhargab Chattopadhyay

About the dataset

Problem

Bob has started his own mobile company. He wants to give tough fight to big companies like Apple, Samsung etc.

He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market, you cannot simply assume things. To solve this problem, he collects sales data of mobile phones of various companies.

Bob wants to find out some relation between features of a mobile phone (ex: RAM, Internal Memory etc) and its selling price. But he is not so good at Machine Learning.

In this problem you do not have to predict actual price but a price range indicating how high the price is. Our goal is to identify a relationship between different features of a mobile phone, such as RAM and internal memory, and its selling price. Our objective is not to predict the actual price of a mobile phone, but to determine a price range that indicates how high the price is likely to be. We hope that with the help of a machine learning model, we can better estimate the pricing of our mobile phones and compete effectively with other players in the market.

About Dataset

The dataset consists of 2000 rows and 21 columns.

Data information.

1.battery_power**-** phone battery capacity is the amount of electricity that a fully charged battery can deliver to a stand-alone device before it is completely discharged. Simply put, this indicator can give a rough idea of how long the phone will work on its own before it is completely discharged.

2.blue **-** the presence of bluetooth.Bluetooth is a short-range wireless technology standard that is used to exchange data between fixed and mobile devices over short distances using UHF radio waves in the ISM bands from 2.402 to 2.48 GHz and build personal area networks (PANs). It is mainly used as an alternative to wired connections, to share files between nearby portable devices, and to connect mobile phones and music.

3.clock_speed-speed at which microprocessor executes instructions.Clock speed is the number of operations that the processor performs per second. The higher it is, the more processor performance. The number of processor cores and cache size are also important. Now even the cheapest dual-core processors come with a frequency of 3.5 GHz - this is the level of a multimedia or gaming computer of the middle class. If this indicator is higher, the possibility of overclocking the processor and the number of cores also increase.

4.dual_sim-has dual sim support or not.The term Dual Sim in a phone or smartphone means support for two SIM cards, one of which you can use, for example, for personal calls, and the second for work. Many modern smartphones support two SIM cards.

5.fc-front camera mega pixels. The front camera is a camera that looks like a small eye, which is located on the front of the phone, in the same place where the sensors are installed (that is, at the top). Other manufacturers did not pay due attention to the characteristics of the front camera in the smartphone, as they hardly interested people. The front camera was used exclusively for making video calls.

6.four_g-has 4G or not. 4G is a generation of mobile communications with increased requirements. It is customary to refer to the fourth generation as promising technologies that allow data transmission at a speed of up to 100 Mbps to mobile (with high mobility) and up to 1 Gbps to fixed subscribers (with low mobility).

7.int_memory-Internal Memory in Gigabytes. Internal Storage is a data storage on a smartphone where important data is found: the operating system (OS), installed applications, photos, videos, documents and other files.

8.m_dep-mobile Depth in cm.

9.mobile_wt-Weight of mobile phone.

10.n_cores-Number of cores of processor.The total number of cores in a single processor in an Android smartphone is typically eight (most iPhone upgrades have six). "The number of nuclear strikes on smartphone performance." big.LITTLE, in turn, stands for simply: there are cores that are more productive (large) and less productive (small).

11.pc-Primary Camera mega pixels. The number of megapixels of a camera sensor describes the image resolution that can be captured with this camera. For example, cameras with a 12 megapixel sensor can take photos with a resolution of 4200x2800 pixels, an 8 megapixel camera allows you to take pictures with a resolution of 3264x2468 pixels.

12.px_height-Pixel Resolution Height.

13.px_width-Pixel Resolution Width.

14.ram-Random Access Memory in Megabytes.Random Access Memory (RAM) is the link between the processor and the playback system because it contains temporary information necessary for running applications to run.

15.sc_h-Screen Height of mobile in cm.

16.sc_w-Screen Width of mobile in cm.

17.talk_time-longest time that a single battery charge.

18.three_g-has 3G or not.Mobile communication of the third generation is built on the basis of packet data transmission. Networks of the third generation 3G operate on the border of decimeter and connected to the network. They allow you to organize videotelephony, watch movies and individual content on your mobile phone.

19.touch_screen-has touchscreen or not. A touchscreen, in fact, is a touch glass that works according to a simple scheme: touching the observer allows you to realize any functions or symptoms of exposure.

20.wifi-has wifi or not. Wi-Fi is a wireless networking technology that allows devices such as computers (laptops and desktops), mobile devices (smartphones and wearables), and other equipment (printers and camcorders) to access the Internet.

21.price_range This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).

Variables

Response Variable:

price _range – The range of mobile phone prices and is categorized into 0,1,2,3.

Predictor Variables:

1. battery_power - Total energy a battery can store in one time measured in mAh
2. blue - Has Bluetooth or not
3. clock_speed - speed at which microprocessor executes instructions
4. dual_sim - Has dual sim support or not
5. fc - Front Camera mega pixels
6. four_g - Has 4G or not
7. int_memory - Internal Memory in Gigabytes
8. m_dep - Mobile Depth in cm
9. mobile_wt - Weight of mobile phone
10. n_cores - Number of cores of processor
11. pc - Primary Camera mega pixels
12. px_height - Pixel Resolution Height
13. px_width - Pixel Resolution Width
14. ram - Random Access Memory in Megabytes
15. sc_h - Screen Height of mobile in cm
16. sc_w - Screen Width of mobile in cm
17. talk_time - longest time that a single battery charge will last when you are
18. three_g – Has 3G or not
19. touch_screen - Has touch screen or not
20. wifi – Has wifi or not

STEP 1 : IMPORT LIBRARIES & LOADING DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

```
from google.colab import files
Files=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 train.csv to train (1).csv
Saving test.csv to test (1).csv
```

+ Code + Text

```
[154] train = pd.read_csv('train.csv')
      test = pd.read_csv('test.csv')
      train
```

STEP 2 : Checking null, Dropping Duplicate values and EDA

```
## check null
train.isnull().sum()
```

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

```
## drop_duplicates
Data1 = train.drop_duplicates()
```

```
df = pd.DataFrame(train)
statistics = df.describe()
# Mode is not included in df.describe(), so we calculate it separately
mode = df.mode().iloc[0]
# Add 'mode' to the statistics DataFrame
statistics.loc['mode'] = mode
print(statistics)
```

```
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000
mode      618.000000     0.0000     0.500000     1.000000     0.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000  ...
mean      0.521500    32.046500     0.501750   140.249000     4.520500  ...
std       0.499662    18.145715     0.288416    35.399655     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...
mode      1.000000    27.000000     0.100000   182.000000     4.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000
mode    347.000000   874.000000  1229.000000    17.000000     1.000000

         talk_time      three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
75%      16.000000     1.000000      1.000000     1.000000     2.250000
max      20.000000     1.000000      1.000000     1.000000     3.000000
mode      7.000000     1.000000      1.000000     1.000000     0.000000
```

```python
#EDA ON DATASET
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   object
dtypes: float64(2), int64(18), object(1)
memory usage: 328.2+ KB
```

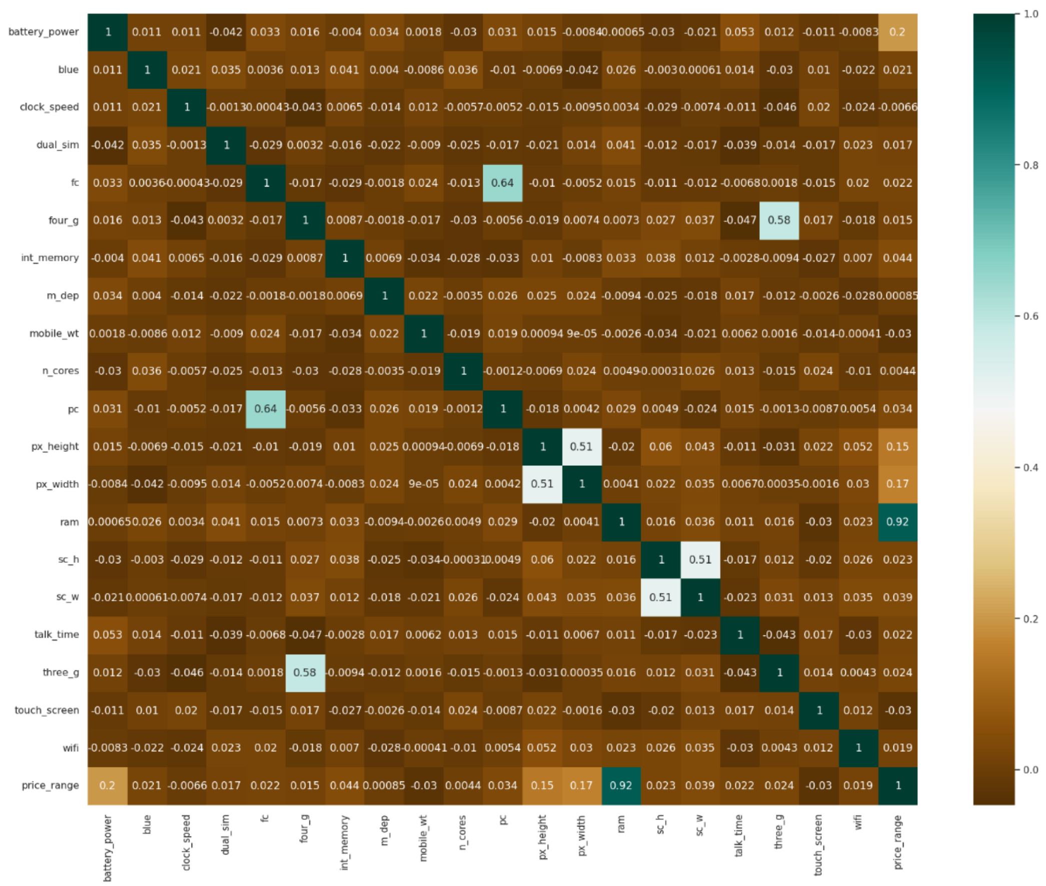
```
train.loc[(train['price_range'] ==0), 'price_range'] = 'Low Cost'
train.loc[(train['price_range'] ==1), 'price_range'] = 'Medium Cost'
train.loc[(train['price_range'] ==2), 'price_range'] = 'High Cost'
train.loc[(train['price_range'] ==3), 'price_range'] = 'Very High Cost'

train.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	1	Medium Cost
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	0	High Cost
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	0	High Cost
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	0	High Cost
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	0	Medium Cost

5 rows × 21 columns

STEP 3 : Data Visualisation

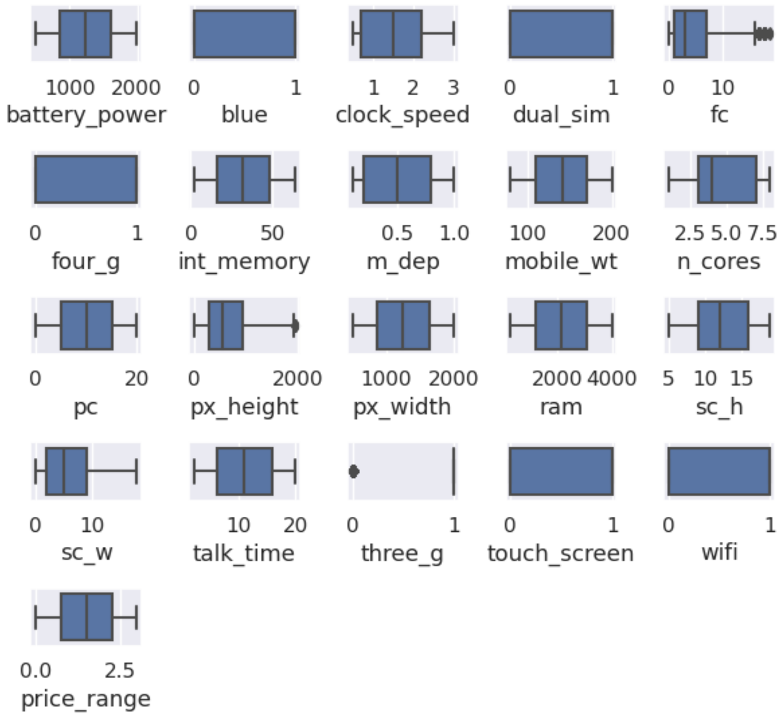


These values have the highest correlation to price:

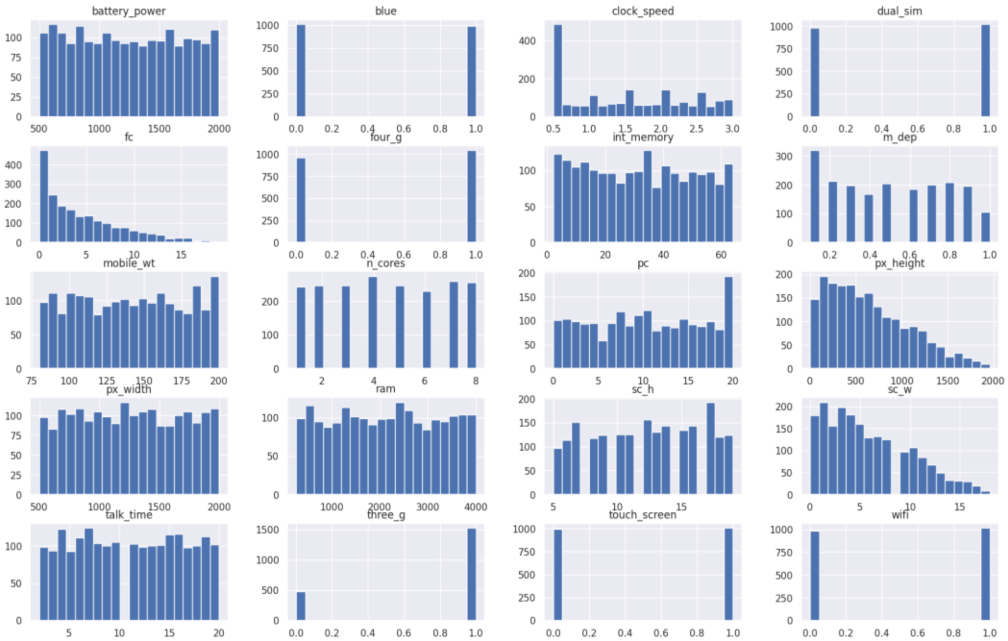
- ram

- battery power
- screen dimensions
- internal memory

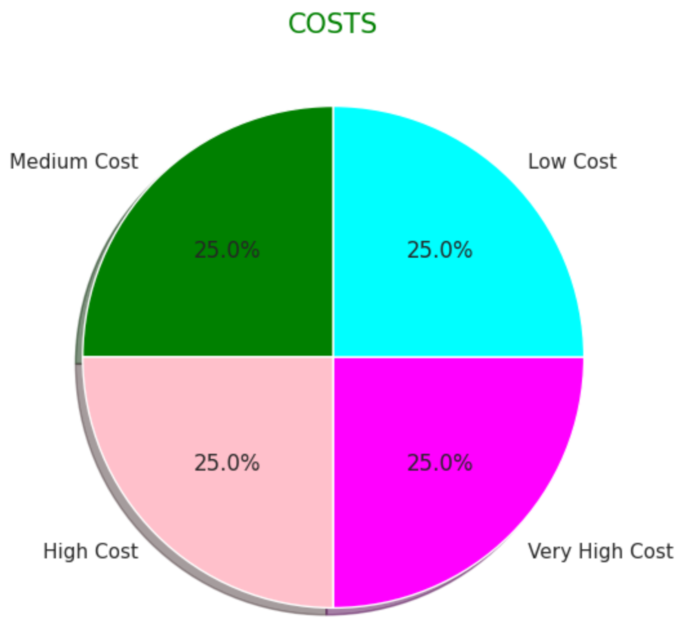
BOX PLOTS



Histograms

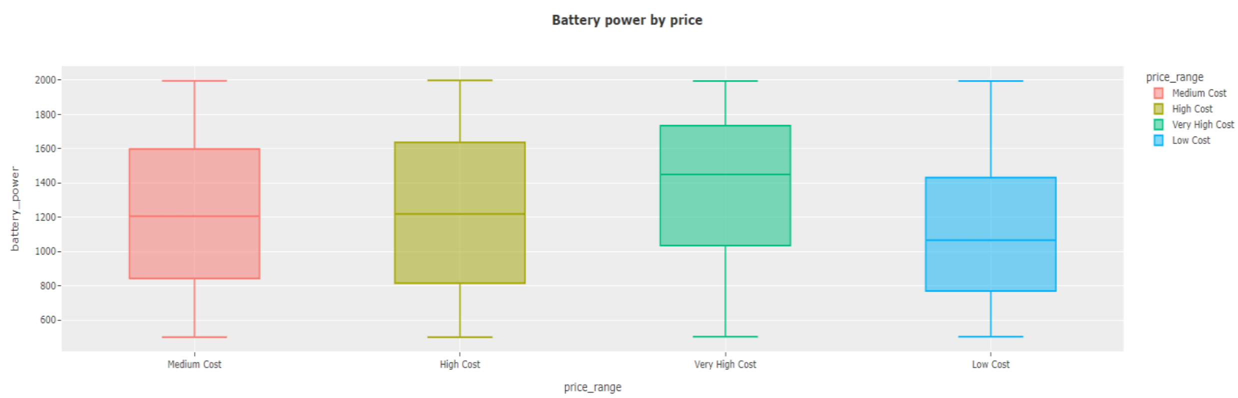


Cost Distribution of Mobile Phones

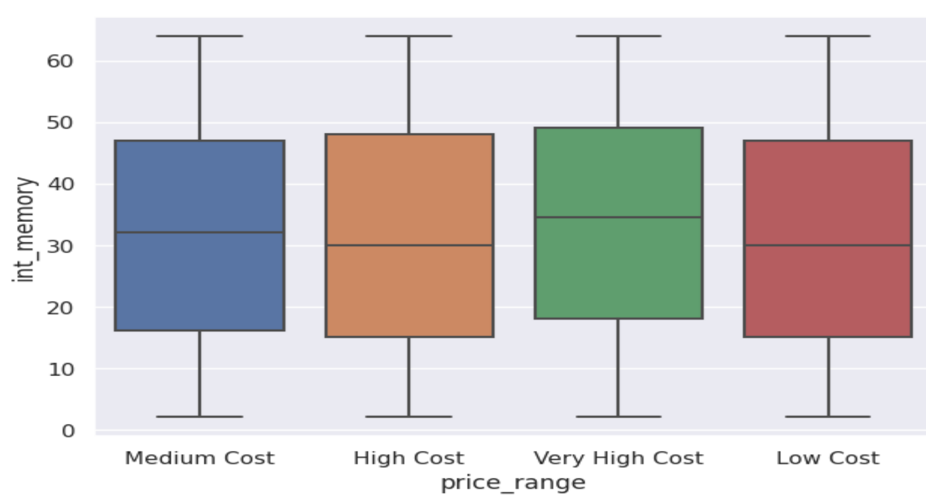


Effect of Ram Capacity on Price

Battery Power by Price



Internal Memory Vs Price Range



#RESULT

```
#As the battery power increases, we can say that the price increases.
#The higher the RAM capacity, the higher the price.
#The price goes up when the phone has 3G.
#The percentages of cheap, medium, expensive, very expensive phones in the dataset are equal.
```

STEP 4 : Data Preprocessing & Outlier Removal

Biasness

```
count = data['price_range'].value_counts()
count
```

```
1    500
2    500
3    500
0    500
Name: price_range, dtype: int64
```

All the classes are equal across the data set. So, there is no bias found in the dataset.

Multi-Collinearity

	feature	VIF
0	battery_power	8.076717
1	blue	1.981927
2	clock_speed	4.260479
3	dual_sim	2.015006
4	fc	3.413529
5	four_g	3.194321
6	int_memory	3.961239
7	m_dep	3.911115
8	mobile_wt	12.972548
9	n_cores	4.646070
10	pc	6.228797
11	px_height	4.262680
12	px_width	11.766282
13	ram	4.688608
14	sc_h	11.510780
15	sc_w	3.720867
16	talk_time	4.859144
17	three_g	6.191783
18	touch_screen	1.989078
19	wifi	2.021012

The Variance Inflation Factor values help in identifying the features that are highly correlated with other features. High VIF values (typically greater than 10) indicate high multicollinearity and may suggest that those features should be considered for removal or further analysis in your regression model to avoid multicollinearity-related issues. Lower VIF values indicate lower multicollinearity.

So, from the dataset, mobile_wt, px_width, sc_h has a high VIF value indicating high multi collinearity and battery_power, pc, three_g also has significant multi collinearity.

STEP 5: Training , Testing , Splitting & Model Fitting

```
[177] Data1.columns

      Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
             'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
             'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
             'touch_screen', 'wifi', 'price_range'],
            dtype='object')

[178] Data1=Data1.reset_index()

[179] import numpy as np
      Q1 = np.percentile(train["talk_time"], 25,
                         interpolation = 'midpoint')

      Q3 = np.percentile(train["talk_time"], 75,
                         interpolation = 'midpoint')
      IQR = Q3 - Q1
      UpperLimit=Q3 + 1.5*IQR
      LowerLimit=Q1 - 1.5*IQR

      OutlierList=[]
      for i in range (train["talk_time"].shape[0]):
        if train["talk_time"][i]>=UpperLimit:
          OutlierList.append(i)
        elif train["talk_time"][i]<=LowerLimit:
          OutlierList.append(i)
      print(OutlierList)

      []

[181] len(OutlierList)

      0

[182] Data1=Data1.drop(OutlierList)

[183] # Define the dependent variable and predictors for training set
      Y1=train['price_range']

[231] X1=train[['battery_power', 'blue', 'clock_speed', 'dual_sim',
              'fc', 'four_g', 'int_memory', 'm_dep', 'n_cores', 'pc',
              'px_height', 'ram', 'sc_w', 'talk_time', 'three_g',
              'touch_screen', 'wifi']]

      from statsmodels.stats.outliers_influence import variance_inflation_factor
      p = train[['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
              'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
              'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
              'touch_screen', 'wifi']]
      # VIF dataframe
      vif_data = pd.DataFrame()
      vif_data["feature"] = p.columns

      # calculating VIF for each feature
      vif_data["VIF"] = [variance_inflation_factor(p.values, i) for i in range(len(p.columns))]
```

Step6 : Model Evaluation

```
#MULTINOMIA

from sklearn.naive_bayes import MultinomialNB
CLF_MN=MultinomialNB()
CLF_MN=CLF_MN.fit(X_train,Y_train)
Y_pred=CLF_MN.predict(X_test)
from sklearn.metrics import classification_report
Report=classification_report(Y_test,Y_pred)
print(Report)

# Calculate the accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
                precision    recall  f1-score   support

     High Cost       0.38      0.36      0.37        92
      Low Cost       0.78      0.76      0.77       105
   Medium Cost       0.41      0.42      0.42        91
Very High Cost       0.57      0.61      0.59       112

      accuracy                           0.55       400
     macro avg       0.54      0.54      0.54       400
  weighted avg       0.55      0.55      0.55       400

Accuracy: 54.75%
```

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='lbfgs',multi_class='multinomial')
clf.fit(X_train, Y_train)

clf.score(X_test, Y_test)
```

```
0.6325
```

```
from sklearn.metrics import accuracy_score

# Predict the classes on the test set
Y_pred = clf.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 63.25%
```

2. Support Vector Classifier

```
# import support vector classifier
# "Support Vector Classifier"
from sklearn.svm import SVC
CLF_svc = SVC()

# fitting x samples and y classes
CLF_svc=CLF_svc.fit(X_train,Y_train)
Y_pred=CLF_svc.predict(X_test)
from sklearn.metrics import classification_report
Report=classification_report(Y_test,Y_pred)
print(Report)

# Calculate the accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
                precision    recall  f1-score   support

     High Cost       0.94      0.95      0.94        92
      Low Cost       0.99      0.98      0.99       105
   Medium Cost       0.95      0.99      0.97        91
Very High Cost       0.98      0.95      0.96       112

      accuracy                           0.96       400
     macro avg       0.96      0.97      0.96       400
  weighted avg       0.97      0.96      0.97       400

Accuracy: 96.50%
```

3. Random Forest Classifier

```
[237] from sklearn.ensemble import RandomForestClassifier
CLF_RandFr =  RandomForestClassifier()
CLF_RandFr=CLF_RandFr.fit(X_train,Y_train)
Y_pred=CLF_RandFr.predict(X_test)
from sklearn.metrics import classification_report
Report=classification_report(Y_test,Y_pred)
print(Report)

# Calculate the accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
                precision    recall  f1-score   support

     High Cost       0.81      0.85      0.83        92
      Low Cost       0.94      0.95      0.95       105
   Medium Cost       0.88      0.87      0.87        91
Very High Cost       0.93      0.89      0.91       112

      accuracy                           0.89       400
     macro avg       0.89      0.89      0.89       400
  weighted avg       0.89      0.89      0.89       400

Accuracy: 89.25%
```

4. Bagging Classifier

```
[ ] from sklearn.ensemble import BaggingClassifier

    # Create a Decision Tree classifier
    rf = RandomForestClassifier()
    # Create a BaggingClassifier
    model = BaggingClassifier(base_estimator=rf, n_estimators=10)

    # Fit the model on the training data
    classifiers = model.fit(X_train, Y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)
    from sklearn.metrics import classification_report
    Report=classification_report(Y_test,Y_pred)
    print(Report)

    # Calculate the accuracy
    accuracy = accuracy_score(Y_test, Y_pred)
    print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
                precision    recall  f1-score   support

     High Cost       0.81      0.85      0.83        92
      Low Cost       0.94      0.95      0.95       105
   Medium Cost       0.88      0.87      0.87        91
Very High Cost       0.93      0.89      0.91       112

      accuracy                           0.89       400
     macro avg       0.89      0.89      0.89       400
  weighted avg       0.89      0.89      0.89       400

Accuracy: 89.25%
```

5. Decision Tree Classifier

```
# Calculate accuracy
accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", accuracy)

from sklearn.metrics import classification_report
Report=classification_report(Y_test,Y_pred)
print(Report)
```

```
Accuracy: 0.8775
                precision    recall  f1-score   support

     High Cost       0.81      0.85      0.83        92
      Low Cost       0.94      0.95      0.95       105
   Medium Cost       0.88      0.87      0.87        91
Very High Cost       0.93      0.89      0.91       112

      accuracy                           0.89       400
     macro avg       0.89      0.89      0.89       400
  weighted avg       0.89      0.89      0.89       400
```

So, SVC classifier is having the best accuracy and below is the price_range predictions for test.csv data

```
array([2, 3, 3, 3, 1, 3, 3, 1, 3, 0, 3, 3, 0, 0, 2, 0, 2, 1, 3, 2, 0, 3,
       1, 1, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 0, 1, 3, 1, 2, 1, 1, 2, 0, 0,
       0, 1, 0, 3, 1, 2, 1, 0, 2, 0, 3, 1, 3, 1, 1, 3, 3, 3, 0, 2, 0, 1,
       2, 3, 1, 2, 1, 2, 2, 3, 3, 0, 2, 0, 2, 3, 0, 3, 3, 0, 3, 0, 3, 1,
       3, 0, 1, 1, 2, 0, 2, 2, 0, 2, 1, 2, 1, 0, 0, 3, 0, 2, 0, 1, 2, 3,
       3, 2, 1, 3, 3, 3, 3, 1, 3, 0, 0, 3, 2, 1, 2, 0, 3, 3, 3, 1, 0, 2,
       2, 1, 3, 0, 1, 0, 3, 2, 1, 3, 1, 3, 2, 3, 3, 3, 2, 3, 2, 3, 1, 0,
       3, 2, 3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 1, 0, 3, 0, 0, 0, 2, 0, 0, 1,
       0, 0, 1, 2, 1, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 0, 1, 3, 1, 0, 2, 2,
       2, 3, 1, 2, 2, 3, 3, 2, 1, 1, 0, 0, 1, 2, 0, 2, 3, 3, 0, 2, 0, 3,
       2, 3, 3, 1, 0, 1, 0, 3, 0, 1, 0, 2, 2, 1, 3, 1, 2, 0, 3, 1, 2, 0,
       0, 2, 1, 3, 3, 3, 1, 1, 3, 0, 0, 2, 3, 3, 1, 3, 1, 0, 3, 2, 1, 2,
       3, 3, 3, 1, 0, 1, 2, 2, 1, 1, 3, 2, 0, 3, 0, 0, 3, 0, 0, 3, 2, 3,
       3, 2, 1, 3, 3, 2, 3, 1, 2, 1, 2, 0, 2, 3, 1, 0, 0, 3, 0, 3, 0, 1,
       2, 0, 2, 3, 1, 3, 2, 2, 0, 2, 0, 0, 0, 1, 3, 2, 0, 0, 0, 3, 2, 0,
       2, 3, 1, 2, 2, 2, 3, 1, 3, 3, 2, 2, 3, 3, 3, 0, 3, 0, 3, 1, 2, 1,
       3, 3, 0, 1, 0, 3, 1, 3, 1, 3, 0, 0, 0, 0, 2, 0, 0, 2, 2, 1, 2, 2,
       2, 0, 1, 0, 0, 3, 3, 0, 3, 1, 2, 2, 1, 2, 3, 1, 1, 2, 2, 1, 2, 0,
       1, 1, 0, 3, 2, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 2, 2, 3, 1, 3, 0, 3,
       1, 3, 0, 1, 1, 1, 2, 0, 3, 2, 3, 3, 1, 3, 2, 2, 1, 3, 2, 0, 1, 2,
       1, 1, 0, 0, 0, 1, 2, 1, 0, 3, 2, 0, 2, 2, 0, 0, 3, 1, 1, 0, 3, 3,
       3, 0, 3, 0, 2, 3, 2, 3, 0, 2, 0, 2, 3, 0, 1, 1, 0, 0, 1, 1, 1, 3,
       3, 3, 2, 3, 1, 2, 2, 2, 3, 3, 2, 1, 2, 1, 2, 2, 1, 0, 2, 2, 0, 0,
       0, 3, 1, 0, 2, 2, 2, 0, 3, 0, 2, 2, 1, 3, 0, 2, 3, 0, 1, 1, 3, 3,
       2, 1, 1, 3, 2, 0, 2, 1, 3, 0, 3, 3, 1, 2, 2, 1, 3, 0, 1, 2, 3, 1,
       3, 2, 3, 1, 0, 1, 0, 3, 1, 0, 3, 2, 3, 2, 0, 3, 3, 3, 2, 3, 3, 1,
       2, 0, 2, 2, 3, 0, 0, 1, 1, 2, 2, 1, 0, 0, 2, 2, 3, 1, 0, 2, 1, 3,
       3, 0, 1, 3, 1, 2, 1, 1, 0, 0, 2, 1, 0, 1, 2, 2, 1, 0, 2, 2, 1, 0,
       3, 0, 0, 3, 2, 0, 0, 1, 0, 0, 3, 0, 3, 1, 3, 1, 1, 3, 3, 0, 2, 1,
       3, 2, 2, 2, 0, 3, 0, 2, 0, 2, 0, 0, 1, 1, 1, 2, 2, 3, 1, 3, 2, 2,
       1, 3, 2, 0, 2, 2, 0, 3, 3, 0, 2, 1, 1, 2, 0, 3, 2, 0, 3, 2, 3, 0,
       0, 3, 0, 2, 2, 3, 2, 2, 2, 2, 1, 1, 3, 0, 1, 1, 2, 2, 1, 0, 0, 1,
       0, 0, 3, 1, 1, 2, 0, 1, 0, 1, 3, 0, 3, 2, 3, 0, 0, 1, 2, 2, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 3, 3, 1, 3, 1, 2, 3, 0, 1, 0, 2, 2, 0, 3, 1,
       0, 3, 1, 1, 0, 3, 3, 3, 2, 3, 0, 3, 2, 0, 0, 0, 2, 3, 2, 0, 2, 1,
       2, 0, 0, 3, 2, 1, 3, 1, 2, 1, 1, 1, 3, 1, 1, 1, 2, 0, 0, 2, 1, 0,
       2, 0, 0, 1, 0, 2, 3, 3, 3, 0, 1, 2, 1, 1, 0, 0, 2, 1, 0, 2, 0, 3,
       2, 2, 1, 2, 0, 2, 1, 3, 0, 0, 3, 2, 3, 0, 0, 2, 3, 3, 1, 2, 2, 1,
       0, 0, 2, 3, 0, 3, 0, 0, 0, 2, 2, 1, 2, 0, 3, 3, 1, 2, 3, 3, 0, 1,
       1, 2, 1, 2, 2, 0, 1, 3, 1, 1, 3, 1, 2, 3, 2, 1, 1, 1, 3, 3, 0, 2,
       3, 0, 2, 3, 2, 2, 1, 3, 2, 0, 1, 2, 0, 2, 1, 1, 2, 3, 2, 1, 2, 1,
       1, 1, 3, 1, 0, 1, 2, 3, 1, 0, 0, 2, 2, 2, 3, 0, 3, 2, 2, 1, 3, 0,
       1, 3, 1, 2, 1, 1, 3, 2, 0, 3, 0, 2, 3, 0, 2, 1, 3, 3, 1, 0, 2, 3,
       1, 0, 2, 1, 2, 1, 3, 0, 2, 2, 0, 2, 3, 2, 3, 0, 2, 1, 1, 2, 2, 3,
       3, 0, 2, 1, 2, 1, 3, 0, 1, 3, 0, 2, 0, 0, 3, 3, 2, 0, 0, 0, 0, 3,
       2, 3, 3, 0, 0, 2, 1, 0, 2, 2])
```

```
array(['Very High Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
       'Medium Cost', 'Very High Cost', 'Very High Cost', 'Medium Cost',
       'Very High Cost', 'Low Cost', 'Very High Cost', 'Very High Cost',
       'Low Cost', 'Low Cost', 'High Cost', 'Low Cost', 'High Cost',
       'Medium Cost', 'Very High Cost', 'High Cost', 'Medium Cost',
       'Very High Cost', 'Medium Cost', 'Medium Cost', 'Very High Cost',
       'Low Cost', 'High Cost', 'Low Cost', 'Very High Cost', 'Low Cost',
       'High Cost', 'Low Cost', 'Very High Cost', 'Low Cost',
       'Medium Cost', 'Medium Cost', 'Very High Cost', 'Medium Cost',
       'High Cost', 'Medium Cost', 'Medium Cost', 'High Cost', 'Low Cost',
       'Low Cost', 'Low Cost', 'Medium Cost', 'Low Cost',
       'Very High Cost', 'Medium Cost', 'High Cost', 'Medium Cost',
       'Low Cost', 'Very High Cost', 'Low Cost', 'Very High Cost',
       'Medium Cost', 'Very High Cost', 'Medium Cost', 'Medium Cost',
       'Very High Cost', 'Very High Cost', 'Very High Cost', 'Low Cost',
       'Medium Cost', 'Medium Cost', 'Medium Cost', 'Medium Cost',
       'Very High Cost', 'Medium Cost', 'High Cost', 'Medium Cost',
       'High Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
       'Low Cost', 'High Cost', 'Low Cost', 'High Cost', 'Very High Cost',
       'Low Cost', 'Very High Cost', 'Very High Cost', 'Low Cost',
       'Very High Cost', 'Low Cost', 'Very High Cost', 'Medium Cost',
       'Very High Cost', 'Low Cost', 'Medium Cost', 'High Cost',
       'High Cost', 'Medium Cost', 'High Cost', 'High Cost', 'Low Cost',
       'High Cost', 'Medium Cost', 'High Cost', 'Medium Cost', 'Low Cost',
       'Low Cost', 'Very High Cost', 'Low Cost', 'High Cost', 'Low Cost',
       'Medium Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
       'Very High Cost', 'Medium Cost', 'Very High Cost',
       'Very High Cost', 'Very High Cost', 'Very High Cost', 'High Cost',
       'Very High Cost', 'Low Cost', 'Low Cost', 'Very High Cost',
       'High Cost', 'Medium Cost', 'High Cost', 'Low Cost',
       'Very High Cost', 'High Cost', 'Very High Cost', 'Medium Cost',
       'Low Cost', 'High Cost', 'High Cost', 'Medium Cost',
       'Very High Cost', 'Medium Cost', 'Medium Cost', 'Low Cost',
       'Very High Cost', 'High Cost', 'Medium Cost', 'High Cost',
       'Medium Cost', 'High Cost', 'High Cost', 'Very High Cost',
       'Very High Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
       'High Cost', 'Very High Cost', 'Medium Cost', 'Low Cost',
       'Very High Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
       'Very High Cost', 'Very High Cost', 'High Cost', 'High Cost',
       'Very High Cost', 'Very High Cost', 'Very High Cost',
       'Very High Cost', 'Medium Cost', 'Low Cost', 'Very High Cost',
       'Low Cost', 'Low Cost', 'Low Cost', 'High Cost', 'Medium Cost',
       'Low Cost', 'Medium Cost', 'Low Cost', 'Low Cost', 'Medium Cost',
       'High Cost', 'Medium Cost', 'Low Cost', 'Low Cost', 'Medium Cost',
```

'Medium Cost', 'High Cost', 'High Cost', 'Medium Cost', 'Low Cost',
'Low Cost', 'Low Cost', 'Medium Cost', 'Low Cost',
'Very High Cost', 'Medium Cost', 'Low Cost', 'High Cost',
'High Cost', 'High Cost', 'Very High Cost', 'Medium Cost',
'Medium Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
'High Cost', 'High Cost', 'Medium Cost', 'Low Cost', 'Low Cost',
'Medium Cost', 'High Cost', 'Low Cost', 'High Cost',
'Very High Cost', 'Very High Cost', 'Low Cost', 'High Cost',
'Low Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
'Very High Cost', 'Medium Cost', 'Low Cost', 'Medium Cost',
'Low Cost', 'Very High Cost', 'Low Cost', 'Medium Cost',
'Low Cost', 'High Cost', 'High Cost', 'Medium Cost', 'High Cost',
'Low Cost', 'Very High Cost', 'Low Cost', 'Very High Cost',
'Medium Cost', 'High Cost', 'Low Cost', 'Low Cost', 'High Cost',
'Medium Cost', 'Very High Cost', 'Very High Cost',
'Very High Cost', 'Medium Cost', 'Medium Cost', 'Very High Cost',
'Low Cost', 'Low Cost', 'High Cost', 'Very High Cost',
'Very High Cost', 'Medium Cost', 'Very High Cost', 'Medium Cost',
'Medium Cost', 'Very High Cost', 'High Cost', 'Medium Cost',
'High Cost', 'Very High Cost', 'Very High Cost', 'Very High Cost',
'Medium Cost', 'Low Cost', 'Medium Cost', 'High Cost',
'Very High Cost', 'Medium Cost', 'Medium Cost', 'Very High Cost',
'High Cost', 'Low Cost', 'Very High Cost', 'Low Cost', 'Low Cost',
'High Cost', 'Low Cost', 'Low Cost', 'Very High Cost', 'High Cost',
'Very High Cost', 'Very High Cost', 'High Cost', 'Medium Cost',
'Very High Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
'Medium Cost', 'High Cost', 'Medium Cost', 'High Cost', 'Low Cost',
'High Cost', 'Very High Cost', 'Medium Cost', 'Low Cost',
'Low Cost', 'Very High Cost', 'Low Cost', 'Very High Cost',
'Low Cost', 'Medium Cost', 'High Cost', 'Low Cost', 'High Cost',
'Very High Cost', 'Medium Cost', 'Very High Cost', 'High Cost',
'High Cost', 'Medium Cost', 'High Cost', 'Low Cost', 'Low Cost',
'Low Cost', 'Medium Cost', 'Very High Cost', 'High Cost',
'Low Cost', 'Low Cost', 'Low Cost', 'Very High Cost', 'High Cost',
'Low Cost', 'High Cost', 'Very High Cost', 'Medium Cost',
'High Cost', 'High Cost', 'High Cost', 'Very High Cost',
'Medium Cost', 'Very High Cost', 'Very High Cost', 'High Cost',
'High Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
'Low Cost', 'Very High Cost', 'Low Cost', 'Very High Cost',
'Medium Cost', 'Very High Cost', 'Medium Cost', 'Very High Cost',
'Very High Cost', 'Low Cost', 'Medium Cost', 'Low Cost',
'Very High Cost', 'Medium Cost', 'Very High Cost', 'High Cost',
'Very High Cost', 'Low Cost', 'Low Cost', 'Low Cost', 'Low Cost',
'High Cost', 'Low Cost', 'Low Cost', 'High Cost', 'High Cost',
'Medium Cost', 'High Cost', 'High Cost', 'High Cost', 'Low Cost',
'Medium Cost', 'Low Cost', 'Low Cost', 'Very High Cost',
'High Cost', 'Low Cost', 'Very High Cost', 'Medium Cost',
'High Cost', 'High Cost', 'Medium Cost', 'High Cost',
'Very High Cost', 'Medium Cost', 'Medium Cost', 'High Cost',
'High Cost', 'Medium Cost', 'High Cost', 'Low Cost', 'Medium Cost',
'Medium Cost', 'Low Cost', 'Very High Cost', 'High Cost',
'Low Cost', 'Low Cost', 'Medium Cost', 'Low Cost', 'Low Cost',
'Medium Cost', 'Medium Cost', 'Low Cost', 'Low Cost', 'Low Cost',
'High Cost', 'High Cost', 'Very High Cost', 'High Cost',
'Very High Cost', 'Low Cost', 'Very High Cost', 'Low Cost',
'Very High Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'Medium Cost', 'High Cost', 'Low Cost', 'Very High Cost',
'High Cost', 'Very High Cost', 'Very High Cost', 'Medium Cost',
'Very High Cost', 'Medium Cost', 'Very High Cost', 'Medium Cost',
'Very High Cost', 'High Cost', 'Low Cost', 'Medium Cost',
'High Cost', 'Medium Cost', 'Medium Cost', 'Low Cost', 'Low Cost',
'Low Cost', 'Medium Cost', 'High Cost', 'Medium Cost', 'Low Cost',
'Very High Cost', 'High Cost', 'Low Cost', 'High Cost',
'High Cost', 'Low Cost', 'Low Cost', 'Very High Cost',
'Medium Cost', 'High Cost', 'Low Cost', 'Very High Cost',
'High Cost', 'Very High Cost', 'Low Cost', 'Very High Cost',
'Low Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
'Very High Cost', 'Low Cost', 'High Cost', 'Low Cost', 'High Cost',
'Very High Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'Low Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'Medium Cost', 'Very High Cost', 'Very High Cost',
'Very High Cost', 'High Cost', 'Very High Cost', 'Medium Cost',
'High Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
'Very High Cost', 'High Cost', 'Low Cost', 'High Cost',
'Medium Cost', 'High Cost', 'High Cost', 'Medium Cost', 'Low Cost',
'High Cost', 'High Cost', 'Low Cost', 'Low Cost', 'Low Cost',
'Very High Cost', 'Medium Cost', 'Low Cost', 'High Cost',
'High Cost', 'High Cost', 'Low Cost', 'Very High Cost', 'Low Cost',
'High Cost', 'High Cost', 'Medium Cost', 'Very High Cost',
'Low Cost', 'High Cost', 'Very High Cost', 'Low Cost',
'Medium Cost', 'Medium Cost', 'Very High Cost', 'Very High Cost',
'Medium Cost', 'Medium Cost', 'High Cost', 'Very High Cost',
'High Cost', 'Low Cost', 'High Cost', 'Medium Cost', 'High Cost',
'Low Cost', 'Very High Cost', 'Very High Cost', 'Medium Cost',
'High Cost', 'High Cost', 'High Cost', 'Very High Cost',
'Low Cost', 'Medium Cost', 'High Cost', 'Very High Cost',
'Medium Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
'Medium Cost', 'Medium Cost', 'Medium Cost', 'Low Cost',
'Very High Cost', 'Medium Cost', 'Low Cost', 'Very High Cost',
'High Cost', 'Very High Cost', 'High Cost', 'Low Cost',
'Very High Cost', 'Very High Cost', 'Very High Cost', 'High Cost',
'Very High Cost', 'Very High Cost', 'Medium Cost', 'High Cost',
'Low Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
'Medium Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'High Cost', 'High Cost', 'High Cost', 'Low Cost', 'Low Cost',
'High Cost', 'High Cost', 'Very High Cost', 'High Cost',
'Low Cost', 'High Cost', 'Medium Cost', 'Very High Cost',
'Very High Cost', 'Low Cost', 'Medium Cost', 'Very High Cost',
'Low Cost', 'High Cost', 'Medium Cost', 'Medium Cost', 'Low Cost',
'Low Cost', 'High Cost', 'Medium Cost', 'Low Cost', 'Medium Cost',
'Medium Cost', 'High Cost', 'High Cost', 'Low Cost', 'High Cost',
'High Cost', 'Medium Cost', 'Low Cost', 'Very High Cost',
'Low Cost', 'Low Cost', 'Very High Cost', 'High Cost', 'Low Cost',
'Low Cost', 'Low Cost', 'Low Cost', 'Low Cost', 'Very High Cost',
'Low Cost', 'Very High Cost', 'Medium Cost', 'Very High Cost',
'High Cost', 'Medium Cost', 'Very High Cost', 'Very High Cost',
'Low Cost', 'Medium Cost', 'Low Cost', 'Very High Cost',
'High Cost', 'High Cost', 'High Cost', 'Low Cost',
'Very High Cost', 'Low Cost', 'High Cost', 'Low Cost', 'High Cost',
'Low Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'Medium Cost', 'High Cost', 'Medium Cost', 'Very High Cost',
'Medium Cost', 'Very High Cost', 'High Cost', 'High Cost',
'Medium Cost', 'Very High Cost', 'High Cost', 'Low Cost',
'Medium Cost', 'High Cost', 'Low Cost', 'Very High Cost',
'Very High Cost', 'Low Cost', 'High Cost', 'Medium Cost',
'Medium Cost', 'High Cost', 'Low Cost', 'Very High Cost',
'High Cost', 'Low Cost', 'Very High Cost', 'High Cost',
'Very High Cost', 'Low Cost', 'Low Cost', 'Very High Cost',
'Low Cost', 'High Cost', 'High Cost', 'Very High Cost',
'High Cost', 'High Cost', 'High Cost', 'High Cost', 'Medium Cost',
'High Cost', 'Very High Cost', 'Low Cost', 'Medium Cost',
'Low Cost', 'Medium Cost', 'High Cost', 'Medium Cost', 'Low Cost',
'Low Cost', 'Medium Cost', 'Low Cost', 'Low Cost',
'Very High Cost', 'Low Cost', 'Medium Cost', 'High Cost',
'Low Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'Very High Cost', 'Low Cost', 'Very High Cost', 'High Cost',
'Very High Cost', 'Low Cost', 'Low Cost', 'Medium Cost',
'High Cost', 'High Cost', 'Medium Cost', 'Low Cost', 'Medium Cost',
'Medium Cost', 'Low Cost', 'Medium Cost', 'Medium Cost',
'Low Cost', 'Low Cost', 'Very High Cost', 'Very High Cost',
'Low Cost', 'Very High Cost', 'Medium Cost', 'Medium Cost',
'Very High Cost', 'Low Cost', 'Medium Cost', 'Low Cost',
'High Cost', 'High Cost', 'Low Cost', 'Very High Cost',
'Medium Cost', 'Low Cost', 'Very High Cost', 'Low Cost',
'Medium Cost', 'Low Cost', 'Very High Cost', 'Very High Cost',
'Very High Cost', 'High Cost', 'Very High Cost', 'Low Cost',
'Very High Cost', 'High Cost', 'Low Cost', 'Low Cost', 'Low Cost',
'High Cost', 'Very High Cost', 'High Cost', 'Low Cost',
'High Cost', 'Medium Cost', 'High Cost', 'Low Cost', 'Low Cost',
'Very High Cost', 'High Cost', 'Low Cost', 'Very High Cost',

```
       'Medium Cost', 'High Cost', 'Medium Cost', 'Medium Cost',
       'Medium Cost', 'Very High Cost', 'Medium Cost', 'Medium Cost',
       'Medium Cost', 'High Cost', 'Medium Cost', 'Low Cost', 'High Cost',
       'High Cost', 'Low Cost', 'High Cost', 'Low Cost', 'Low Cost',
       'Low Cost', 'Low Cost', 'High Cost', 'Very High Cost',
       'Very High Cost', 'Very High Cost', 'Low Cost', 'Medium Cost',
       'High Cost', 'Medium Cost', 'Medium Cost', 'Low Cost', 'Low Cost',
       'High Cost', 'Medium Cost', 'Low Cost', 'High Cost', 'Low Cost',
       'Very High Cost', 'High Cost', 'High Cost', 'Medium Cost',
       'High Cost', 'Low Cost', 'High Cost', 'Medium Cost',
       'Very High Cost', 'Low Cost', 'Low Cost', 'Very High Cost',
       'High Cost', 'Very High Cost', 'Low Cost', 'Low Cost', 'High Cost',
       'Very High Cost', 'Very High Cost', 'Medium Cost', 'High Cost',
       'High Cost', 'Medium Cost', 'Low Cost', 'Low Cost', 'High Cost',
       'Very High Cost', 'Low Cost', 'Very High Cost', 'Low Cost',
       'Low Cost', 'Low Cost', 'High Cost', 'High Cost', 'Medium Cost',
       'High Cost', 'Low Cost', 'Very High Cost', 'High Cost',
       'Medium Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
       'Low Cost', 'Medium Cost', 'Medium Cost', 'High Cost',
       'Medium Cost', 'High Cost', 'High Cost', 'Low Cost', 'Medium Cost',
       'Very High Cost', 'Medium Cost', 'Medium Cost', 'Very High Cost',
       'Medium Cost', 'High Cost', 'Very High Cost', 'High Cost',
       'Medium Cost', 'Medium Cost', 'Medium Cost', 'Very High Cost',
       'Very High Cost', 'Low Cost', 'High Cost', 'Very High Cost',
       'Low Cost', 'High Cost', 'Very High Cost', 'Very High Cost',
       'High Cost', 'High Cost', 'Very High Cost', 'High Cost',
       'Low Cost', 'Medium Cost', 'High Cost', 'Low Cost', 'High Cost',
       'Medium Cost', 'Medium Cost', 'High Cost', 'High Cost',
       'High Cost', 'Medium Cost', 'High Cost', 'Medium Cost', 'Low Cost',
       'Medium Cost', 'Very High Cost', 'Medium Cost', 'Low Cost',
       'Medium Cost', 'High Cost', 'Very High Cost', 'Medium Cost',
       'Low Cost', 'Low Cost', 'Very High Cost', 'High Cost', 'High Cost',
       'Very High Cost', 'Low Cost', 'Very High Cost', 'Very High Cost',
       'High Cost', 'Medium Cost', 'Very High Cost', 'Low Cost',
       'Medium Cost', 'Very High Cost', 'Medium Cost', 'Medium Cost',
       'Medium Cost', 'Medium Cost', 'Very High Cost', 'High Cost',
       'Low Cost', 'Very High Cost', 'Low Cost', 'High Cost',
       'Very High Cost', 'Low Cost', 'Very High Cost', 'High Cost',
       'Very High Cost', 'Very High Cost', 'Medium Cost', 'Low Cost',
       'High Cost', 'Very High Cost', 'Medium Cost', 'Low Cost',
       'High Cost', 'Medium Cost', 'High Cost', 'Medium Cost',
       'High Cost', 'Low Cost', 'High Cost', 'High Cost', 'Low Cost',
       'High Cost', 'Very High Cost', 'High Cost', 'Very High Cost',
       'Low Cost', 'High Cost', 'Medium Cost', 'Medium Cost', 'High Cost',
       'High Cost', 'Very High Cost', 'Very High Cost', 'Low Cost',
       'High Cost', 'Medium Cost', 'High Cost', 'Medium Cost',
       'Very High Cost', 'Medium Cost', 'Medium Cost', 'Very High Cost',
       'Low Cost', 'Medium Cost', 'Low Cost', 'Low Cost',
       'Very High Cost', 'Very High Cost', 'High Cost', 'Low Cost',
       'Low Cost', 'Low Cost', 'Low Cost', 'Very High Cost', 'High Cost',
       'Very High Cost', 'Very High Cost', 'Low Cost', 'Low Cost',
       'High Cost', 'Medium Cost', 'Low Cost', 'High Cost', 'High Cost'],
      dtype=object)
```
CodeText