Mobile Price Classification:

Abstract:

This project is centered on creating a machine learning model that can categorize phone prices into different price-strata according to its specifications. Using an extensive database of mobile phone attributes, we used different classification algorithms such as Random Forest, K-Nearest Neighbours (KNN), Support Vector Machine (SVM), Gaussian Naive Bayes and XGBoost. Data were processed after pre-processing such as data type conversion, normalization, and outlier detection. Hyperparameter tuning was performed on the best models (KNN and XGBoost) to increase model performance. The end models could accomplish a relatively high accuracy, indicating that it is possible to predict mobile price ranges based on technical specifications.

Introduction:

Dynamic tech advancements of mobile phones have made it difficult for both end-users and manufacturers to correlate a device's specifications VS the pricing analysis of the markets. It is in this context, that this project attempts to explore a machine learning based classification system that would classify mobile phones into their price range based on hardware features with high accuracy. Such a facility can be important 30 for market analysis, competitive pricing, and consumer advice. The paper explains the methodology, starting with data acquisition and preprocessing, through model selection, training and testing to conclusions and future work.

Dataset Description:

The dataset I have used in this project consists with 20 features describing different aspects of the mobile phones and price_range is the target variable. Those features are: battery_power, blue (Bluetooth), clock_speed, dual_sim, fc (front camera megapixels), four_g, int_memory (internal memory), m_dep (mobile depth), mobile_wt (mobile weight), n_cores (number of processor cores), pc (primary camera megapixels), px_height (Pixel Resolution Height), px_width (Pixel Resolution Width), ram (Random Access Memory), sc_h (Screen Height), sc_w (Screen Width), talk_time, three_g, touch_screen, and wifi. The price_range looks like a factor, and probably represents Low, Medium, and High. All hundred user's records per particular service are pooled and becomes a single record, therefore 20 records in total are created.

Technologies and Libraries Used:

The project was coded in Python with several relevant libraries for data pre-processing, machine learning and model validation.

1. Pandas: This was used to load, manipulate and inspect the data.
2. NumPy: Used for numerical calculations, and array handling.
3. Scikit-learn - used for machine learning, included for example in:
4. Data splitting (train_test_split)
5. Preprocessing (MinMaxScaler, StandardScaler, LabelEncoder)
6. Model implementations (RandomForestClassifier, KNeighborsClassifier, SVC, GaussianNB, GridSearchCV)
7. Evaluation metrics (accuracy_score, classification_report)
8. XGBoost: The top choice for gradient boosting, as it is well-documented and very intuitive for boosting.
9. Joblib: For saving and loading the trained models.

Data Preprocessing:

Preprocessing of data was an important process to enhance the content and applicability of data for machine learning models.

- Data Type Conversion: Some integer columns (blue, dual_sim, four_g, three_g, touch_screen, wifi) portraying boolean features were converted to boolean type for clearer display and possible model explanation.
- Normalization: Numerical variables were scaled with MinMaxScaler to have a common range (0-1). This is to avoid giving too much weight to features with high variances and improving the performance of distance-based algorithms such as KNN and SVM.
- Outlier Removal: Outliers were detected using the Interquartile Range (IQR) technique with a cut-off value of 1.5 for the numerical columns. This step is intended to increase the model robustness by down-weighting the effect of extreme values. Outliers were removed and a clean dataset was obtained with 0 rows removed as a result of outlier removal process.

Model Architecture:

In this work, a few supervised machine learning classification techniques were investigated to know the better algorithm which can be used for prediction of mobile's price ranges.

1. Random Forest Classifier: A bagging classifier from an ensemble learning method, and is a decision tree where a forest is grown during training and the class and class mode or class mean prediction to be used if the Random Forest's predictions are the class modes or class means the Average.
2. K-Nearest Neighbors (KNN) Classifier: A non-parametric, lazy learning algorithm that assigns classification according to the majority class of its 'k' nearest neighbors in the feature space.
3. SVM Classifier: Support Vector Machine (SVM) is a very powerful algorithm which draws an optimal hyper-plane between different data points to categorise them into different classes; the margin between which is maximized.
4. Gaussian Naive Bayes: A probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features.
5. XGBoost Classifier - A Scalable and Flexible algorithm for Gradient Boosting. It also uses machine learning algorithms within the Gradient Boosting domain.

Training Configuration:

The data is then performed using the train_test_split method with a test size of 20% and a random-style_seed of 42 to maintain consistency across models. Features were normalized for SVM, Gaussian Naive Bayes, and XGBoost to 0 mean and unit variance using StandardScaler prior to training, as this is common to enhance such models.

Hyperparameter tuning was not performed for KNN and XGBoost (best initial models) besides for both we did use GridSearchCV (rule of thumb, 2-fold more than the number of factors evaluated) with 5-fold cross-validation and scoring metric as accuracy.

- KNN Hyperparameters: n_neighbors (3, 5, 7, 9, 11, 13, 15), weights (uniform, distance), metric (euclidean, manhattan, minkowski).
- XGBoost Hyperparameters: n_estimators (100, 200), max_depth (3, 5, 7), learning_rate (0.01, 0.1, 0.2), subsample (0.7, 0.8, 0.9), colsample_bytree (0.7, 0.8, 0.9), gamma (0, 0.1, 0.2), reg_alpha (0, 0.01, 0.1) and reg_lambda (1, 0.1, 0.01).

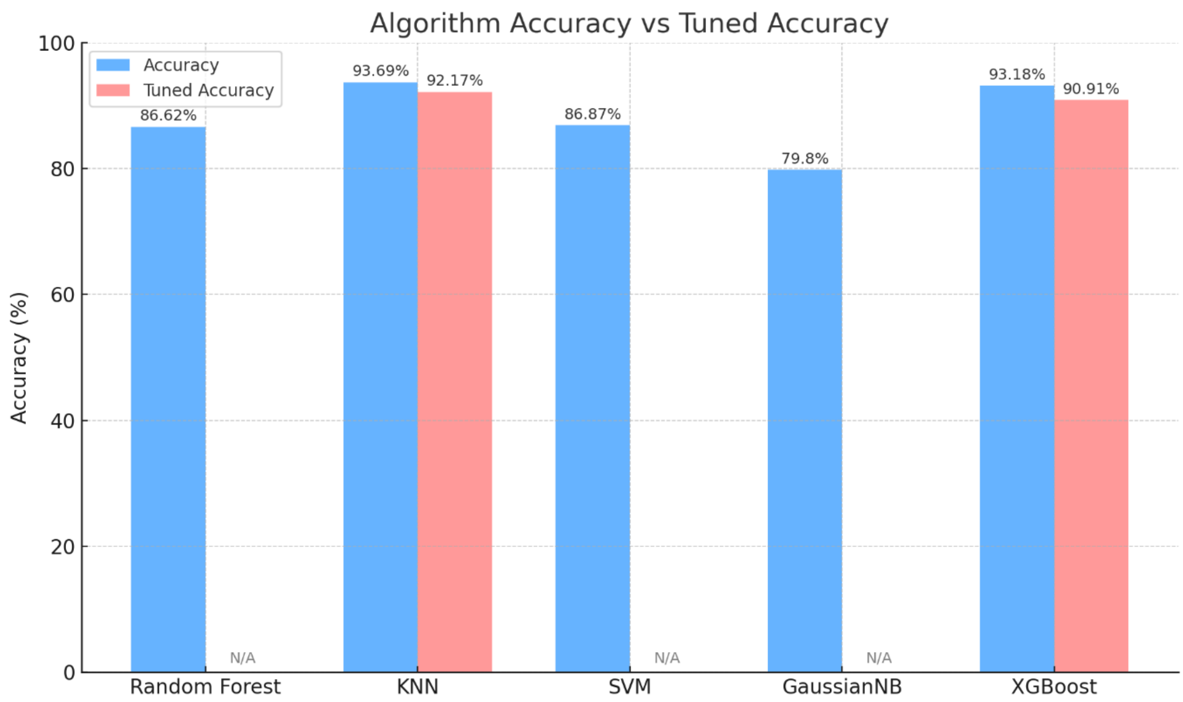
Results and Evaluation:

The models were evaluated based on accuracy and a detailed classification report (precision, recall, f1-score).

Algorithm	Accuracy	Tuned Accuracy
Random Forest	86.62%	N/A
KNN	93.69%	92.17%
SVM	86.87%	N/A
GaussianNB	79.80%	N/A
XGBoost	93.18%	90.91%

At the beginning, KNN achieved the highest accuracy of 0.9369, XGBoost maintained the second highest accuracy of 0.9318. After hyperparameter tuning, KNN model reached 0.9217 of accuracy {'metric': 'manhattan', 'n_neighbors': 13, 'weights': 'uniform'}. The optimized XGBoost classifier had accuracy 0.9091 with the best parameters colsample_bytree: 0.9, gamma: 0.1, learning_rate: 0.2, max_depth: 3, n_estimators: 200, reg_alpha: 0, reg_lambda: 1, subsample: 0.7. Tuning sometimes affected accuracy slightly but the KNN/XGBoost were consistently the best-performing algorithms across the trials.

Results Graph:



Challenges Faced:

One of the first difficulties was deciding on the best algorithms to use in a multi-class classification problem with numerical and boolean features. It was trial and error trying different models and learning the strengths and weaknesses of each was paramount. Another difficulty was about managing the data scaling and outlier detection in a way to maximize the model performance. The outlier removal process found no significant outliers present, but it was important to have verified that this step had been carried out correctly. Last but not least, to achieve full exploitation of hyperparameter space of complex models like XGBoost, we had to sensibly set up GridSearchCV so as to not overload the computer.

Future Scope:

There are a number of lines that could be pursued to improve this project.

- Feature Engineering: Deriving some calculated features from existing ones (e.g., pixel density from px_height and px_width, or estimates of battery life) might double-check our model.
- Advanced Hyperparameter Optimization: Using more sophisticated tuning methods such as Randomized Search CV or Bayesian Optimization may be able to better navigate the hyperparameter space and find a better fit.
- Deep Learning Models: Exploring the possibility of using neural networks such as deep learning architectures would be useful for complex data patterns.
- Model Deploying: If the trained model can be deployed as webservice or API to predict price range of new specifications will be an awesome feature and provide the real-time predictions for the price range of mobile phone.
- Interpretability: Investigating the methods to interpret the models' predictions, especially for tree-based models, one may reveal which features are more determinant in order to classify the 3 classes of prices.

Conclusions:

This work convincingly implemented (i.e., developed and evaluated) machine learning models for classifying mobile phone prices from the heady information about its features. After proper data preprocessing of the expressed transcripts referring to data type transformation, normalization and outlier detections, the dataset was designed for robust model building. They Compared Different algorithms and found the best result with XGBoost and K-nearest Neighbors both reported good accuracy pervasively. These models were further tuned for hyperparameters, even though the initial performance was already competitive. The project serves as an example of how machine learning algorithms can be applied to solve real-life classification tasks and serves as a strong core for future improvements for predicting mobile price.

```
import pandas as pd
import numpy as np
import joblib

df = pd.read_csv(r"E:\Projects\mobile_phone_pricing\Mobile Phone
Pricing\dataset.csv")

df.head()
```

```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory
m_dep  \
0            842     0          2.2         0   1       0           7
0.6
1           1021     1          0.5         1   0       1          53
0.7
2            563     1          0.5         1   2       1          41
0.9
3            615     1          2.5         0   0       0          10
0.8
4           1821     1          1.2         0  13       1          44
0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w
talk_time  \
0        188        2  ...         20       756  2549     9     7
19
1        136        3  ...        905      1988  2631    17     3
7
2        145        5  ...       1263      1716  2603    11     2
9
3        131        6  ...       1216      1786  2769    16     8
11
4        141        2  ...       1208      1212  1411     8     2
15

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1

[5 rows x 21 columns]
```

```
df.shape
```

```
(2000, 21)
```

```
df.describe
```

```
<bound method NDFrame.describe of       battery_power  blue
clock_speed  dual_sim  fc  four_g  int_memory  \
0               842     0          2.2         0   1       0
7
1              1021     1          0.5         1   0       1
53
2               563     1          0.5         1   2       1
41
3               615     1          2.5         0   0       0
10
4              1821     1          1.2         0  13       1
44
...             ...   ...          ...       ...  ..     ...          .
..
1995            794     1          0.5         1   0       1
2
1996           1965     1          2.6         1   0       0
39
1997           1911     0          0.9         1   1       1
36
1998           1512     0          0.9         0   4       1
46
1999            510     1          2.0         1   5       1
45

      m_dep  mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h
sc_w  \
0       0.6        188        2  ...         20       756  2549     9
7
1       0.7        136        3  ...        905      1988  2631    17
3
2       0.9        145        5  ...       1263      1716  2603    11
2
3       0.8        131        6  ...       1216      1786  2769    16
8
4       0.6        141        2  ...       1208      1212  1411     8
2
...     ...        ...      ...  ...        ...       ...   ...   ...
...
1995    0.8        106        6  ...       1222      1890   668    13
4
1996    0.2        187        4  ...        915      1965  2032    11
10
1997    0.7        108        8  ...        868      1632  3057     9
1
1998    0.1        145        5  ...        336       670   869    18
10
1999    0.9        168        6  ...        483       754  3919    19
4
```

```
      talk_time  three_g  touch_screen  wifi  price_range
0            19        0             0     1            1
1             7        1             1     0            2
2             9        1             1     0            2
3            11        1             0     0            2
4            15        1             1     0            1
...         ...      ...           ...   ...          ...
1995         19        1             1     0            0
1996         16        1             1     1            2
1997          5        1             1     0            3
1998         19        1             1     1            0
1999          2        1             1     1            3

[2000 rows x 21 columns]>
```

```
df.dtypes
```

```
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

Converting data types into more suitable data types

```
df['dual_sim'] = df['dual_sim'].astype(bool)

df['four_g'] = df['four_g'].astype(bool)

df['three_g'] = df['three_g'].astype(bool)

df['touch_screen'] = df['touch_screen'].astype(bool)
```

```
df['wifi'] = df['wifi'].astype(bool)

df['blue'] = df['blue'].astype(bool)

df.dtypes
```

```
battery_power      int64
blue                bool
clock_speed      float64
dual_sim            bool
fc                 int64
four_g              bool
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g             bool
touch_screen        bool
wifi                bool
price_range        int64
dtype: object
```

```
df.head()
```

```
   battery_power   blue  clock_speed  dual_sim  fc  four_g  int_memory
m_dep  \
0            842  False          2.2     False   1   False           7
0.6
1           1021   True          0.5      True   0    True          53
0.7
2            563   True          0.5      True   2    True          41
0.9
3            615   True          2.5     False   0   False          10
0.8
4           1821   True          1.2     False  13    True          44
0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w
talk_time  \
0        188        2  ...         20       756  2549     9     7
19
1        136        3  ...        905      1988  2631    17     3
7
```

```
2        145         5  ...        1263       1716  2603     11      2
9
3        131         6  ...        1216       1786  2769     16      8
11
4        141         2  ...        1208       1212  1411      8      2
15

   three_g  touch_screen   wifi  price_range
0    False         False   True            1
1     True          True  False            2
2     True          True  False            2
3     True         False  False            2
4     True          True  False            1

[5 rows x 21 columns]
```

Looking for null values and outliers

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

#Normalization
from sklearn.preprocessing import MinMaxScaler, StandardScaler

def normalize_dataframe(df, columns=None, scaler_type='minmax'):
    df_scaled = df.copy()
    numerical_cols =
df_scaled.select_dtypes(include=np.number).columns.tolist()

    if columns is None:
        columns_to_scale = numerical_cols
    else:
        columns_to_scale = [col for col in columns if col in
numerical_cols]

    if not columns_to_scale:
        print("Warning: No numerical columns found or specified for
scaling. Returning original DataFrame.")
        return df_scaled

    scaler = None
    if scaler_type == 'minmax':
        scaler = MinMaxScaler()
    elif scaler_type == 'standard':
        scaler = StandardScaler()
    else:
        print(f"Error: Invalid scaler_type: {scaler_type}. Choose
'minmax' or 'standard'. Returning original DataFrame.")
        raise ValueError("Invalid scaler_type. Choose 'minmax' or
'standard'.")
```

```
    df_scaled[columns_to_scale] =
scaler.fit_transform(df_scaled[columns_to_scale])
    print(f"Features scaled using {scaler_type.capitalize()}Scaler for
columns: {columns_to_scale}")
    return df_scaled

normalize_dataframe(df)
```

```
Features scaled using MinmaxScaler for columns: ['battery_power',
'clock_speed', 'fc', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores',
'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time',
'price_range']
```

```
      battery_power   blue  clock_speed  dual_sim        fc  four_g  \
0          0.227789  False         0.68     False  0.052632   False
1          0.347361   True         0.00      True  0.000000    True
2          0.041416   True         0.00      True  0.105263    True
3          0.076152   True         0.80     False  0.000000   False
4          0.881764   True         0.28     False  0.684211    True
...             ...    ...          ...       ...       ...     ...
1995       0.195725   True         0.00      True  0.000000    True
1996       0.977956   True         0.84      True  0.000000   False
1997       0.941884  False         0.16      True  0.052632    True
1998       0.675351  False         0.16     False  0.210526    True
1999       0.006012   True         0.60      True  0.263158    True

      int_memory     m_dep  mobile_wt   n_cores  ...  px_height
px_width  \
0       0.080645  0.555556   0.900000  0.142857  ...   0.010204
0.170895
1       0.822581  0.666667   0.466667  0.285714  ...   0.461735
0.993324
2       0.629032  0.888889   0.541667  0.571429  ...   0.644388
0.811749
3       0.129032  0.777778   0.425000  0.714286  ...   0.620408
0.858478
4       0.677419  0.555556   0.508333  0.142857  ...   0.616327
0.475300
...          ...       ...        ...       ...  ...        ...
...
1995    0.000000  0.777778   0.216667  0.714286  ...   0.623469
0.927904
1996    0.596774  0.111111   0.891667  0.428571  ...   0.466837
0.977971
1997    0.548387  0.666667   0.233333  1.000000  ...   0.442857
0.755674
1998    0.709677  0.000000   0.541667  0.571429  ...   0.171429
0.113485
1999    0.693548  0.888889   0.733333  0.714286  ...   0.246429
0.169559
```

```
           ram      sc_h      sc_w  talk_time  three_g  touch_screen
wifi  \
0     0.612774  0.285714  0.388889   0.944444    False         False
True
1     0.634687  0.857143  0.166667   0.277778     True          True
False
2     0.627205  0.428571  0.111111   0.388889     True          True
False
3     0.671566  0.785714  0.444444   0.500000     True         False
False
4     0.308658  0.214286  0.111111   0.722222     True          True
False
...        ...       ...       ...        ...      ...           ...
...
1995  0.110102  0.571429  0.222222   0.944444     True          True
False
1996  0.474613  0.428571  0.555556   0.777778     True          True
True
1997  0.748530  0.285714  0.055556   0.166667     True          True
False
1998  0.163816  0.928571  0.555556   0.944444     True          True
True
1999  0.978888  1.000000  0.222222   0.000000     True          True
True

      price_range
0        0.333333
1        0.666667
2        0.666667
3        0.666667
4        0.333333
...           ...
1995     0.000000
1996     0.666667
1997     1.000000
1998     0.000000
1999     1.000000

[2000 rows x 21 columns]
```

```
#Outlier removal
def remove_outliers_iqr(df, columns=None, threshold=1.5):
    df_cleaned = df.copy()
    numerical_cols =
df_cleaned.select_dtypes(include=np.number).columns.tolist()

    if columns is None:
        columns_to_check = numerical_cols
    else:
```

```
        columns_to_check = [col for col in columns if col in
numerical_cols]

    if not columns_to_check:
        print("Warning: No numerical columns found or specified for
outlier removal. Returning original DataFrame.")
        return df_cleaned

    initial_rows = len(df_cleaned)
    rows_to_drop = []

    for col in columns_to_check:
        Q1 = df_cleaned[col].quantile(0.25)
        Q3 = df_cleaned[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - threshold * IQR
        upper_bound = Q3 + threshold * IQR

        outlier_indices = df_cleaned[(df_cleaned[col] < lower_bound) |
(df_cleaned[col] > upper_bound)].index
        rows_to_drop.extend(outlier_indices)

        print(f"Outlier check applied to column '{col}'.")

    rows_to_drop = list(set(rows_to_drop))
    df_cleaned.drop(rows_to_drop, inplace=True)

    removed_rows = initial_rows - len(df_cleaned)
    print(f"Total rows removed due to outliers: {removed_rows}")

    return df_cleaned

df = remove_outliers_iqr(df)

df.shape
```

```
Outlier check applied to column 'battery_power'.
Outlier check applied to column 'clock_speed'.
Outlier check applied to column 'fc'.
Outlier check applied to column 'int_memory'.
Outlier check applied to column 'm_dep'.
Outlier check applied to column 'mobile_wt'.
Outlier check applied to column 'n_cores'.
Outlier check applied to column 'pc'.
Outlier check applied to column 'px_height'.
Outlier check applied to column 'px_width'.
Outlier check applied to column 'ram'.
Outlier check applied to column 'sc_h'.
Outlier check applied to column 'sc_w'.
Outlier check applied to column 'talk_time'.
```

```
Outlier check applied to column 'price_range'.
Total rows removed due to outliers: 0

(1980, 21)
```

Testing the dataset with several different algorithms

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

X = df.drop(columns=["price_range"])
y = df["price_range"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Random Forest Classifier Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Random Forest Classifier Evaluation:
Accuracy: 0.8662
Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.96      0.94       107
           1       0.82      0.79      0.81        97
           2       0.76      0.79      0.78        86
           3       0.95      0.90      0.92       106

    accuracy                           0.87       396
   macro avg       0.86      0.86      0.86       396
weighted avg       0.87      0.87      0.87       396
```

```
#KNN
from sklearn.neighbors import KNeighborsClassifier

X = df.drop(columns=["price_range"])
y = df["price_range"]
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("K-Nearest Neighbors Classifier Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
K-Nearest Neighbors Classifier Evaluation:
Accuracy: 0.9369
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.99      0.96       107
           1       0.92      0.91      0.91        97
           2       0.93      0.87      0.90        86
           3       0.96      0.96      0.96       106

    accuracy                           0.94       396
   macro avg       0.94      0.93      0.93       396
weighted avg       0.94      0.94      0.94       396
```

```python
#SVM
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

X = df.drop(columns=["price_range"])
y = df["price_range"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SVC(random_state=42)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
```

```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Support Vector Machine Classifier Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Support Vector Machine Classifier Evaluation:
Accuracy: 0.8687
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.93      0.94       107
           1       0.80      0.86      0.83        97
           2       0.77      0.77      0.77        86
           3       0.94      0.90      0.92       106

    accuracy                           0.87       396
   macro avg       0.86      0.86      0.86       396
weighted avg       0.87      0.87      0.87       396
```

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler

X = df.drop(columns=["price_range"])
y = df["price_range"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = GaussianNB()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Gaussian Naive Bayes Classifier Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Gaussian Naive Bayes Classifier Evaluation:
Accuracy: 0.7980
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.93      0.93       107
           1       0.69      0.72      0.71        97
           2       0.60      0.67      0.64        86
           3       0.96      0.83      0.89       106

    accuracy                           0.80       396
   macro avg       0.80      0.79      0.79       396
weighted avg       0.81      0.80      0.80       396
```

```python
#XGBoost
import xgboost as xgb
from sklearn.preprocessing import StandardScaler

X = df.drop(columns=["price_range"])
y = df["price_range"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = xgb.XGBClassifier(use_label_encoder=False,
eval_metric='mlogloss', random_state=42)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("XGBoost Classifier Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
C:\Users\tanis\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\xgboost\training.py:183: UserWarning:
[13:36:04] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\
learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
XGBoost Classifier Evaluation:
Accuracy: 0.9318
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       107
           1       0.90      0.91      0.90        97
           2       0.90      0.87      0.89        86
           3       0.97      0.95      0.96       106

    accuracy                           0.93       396
   macro avg       0.93      0.93      0.93       396
weighted avg       0.93      0.93      0.93       396
```

Hence we can say that the best algorithms will be KNN and XGBoost , now we will proceed ahead with them using hyperparameter tuning

```python
#KNN Hyperparameter tuned
from sklearn.model_selection import  GridSearchCV

X = df.drop(columns=["price_range"])
y = df["price_range"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

knn = KNeighborsClassifier()

grid_search = GridSearchCV(estimator=knn, param_grid=param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1,
verbose=1)

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f"Best Hyperparameters: {best_params}")

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```

```
report = classification_report(y_test, y_pred)

print("\nK-Nearest Neighbors Classifier Evaluation (Tuned):")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
Best Hyperparameters: {'metric': 'manhattan', 'n_neighbors': 13,
'weights': 'uniform'}

K-Nearest Neighbors Classifier Evaluation (Tuned):
Accuracy: 0.9217
Classification Report:
               precision    recall  f1-score   support

           0       0.91      1.00      0.96       107
           1       0.90      0.88      0.89        97
           2       0.88      0.87      0.88        86
           3       0.98      0.92      0.95       106

    accuracy                           0.92       396
   macro avg       0.92      0.92      0.92       396
weighted avg       0.92      0.92      0.92       396
```

```
#XGBoost Hyperparameter Tuned
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder

X = df.drop(columns=["price_range"])
y = df["price_range"]

le = LabelEncoder()
y_encoded = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.01, 0.1],
```

```
    'reg_lambda': [1, 0.1, 0.01]
}

xgb_model = xgb.XGBClassifier(
    objective='multi:softprob',
    eval_metric='mlogloss',
    random_state=42
)

grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train_scaled, y_train)

best_params = grid_search.best_params_
print(f"Best Hyperparameters found by GridSearchCV: {best_params}")

best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=[str(c)
for c in le.classes_])

print("\nXGBoost Classifier Evaluation (Tuned):")
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", report)
```

```
Fitting 5 folds for each of 4374 candidates, totalling 21870 fits
Best Hyperparameters found by GridSearchCV: {'colsample_bytree': 0.9,
'gamma': 0.1, 'learning_rate': 0.2, 'max_depth': 3, 'n_estimators':
200, 'reg_alpha': 0, 'reg_lambda': 1, 'subsample': 0.7}

XGBoost Classifier Evaluation (Tuned):
Accuracy: 0.9091
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.98      0.96        99
           1       0.88      0.88      0.88        99
           2       0.86      0.87      0.86        99
           3       0.97      0.91      0.94        99

    accuracy                           0.91       396
```

```
   macro avg       0.91      0.91      0.91       396
weighted avg       0.91      0.91      0.91       396
```

We can choose the algorithm we want to use but KNN and XGboost will surely be better algorithms for the dataset compared to few pther algorithms