

Resource: <https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification>

1.Download a dataset from Kaggle.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
train = pd.read_csv("/kaggle/input/datasets/iabhishekofficial/mobile-price-classification/train.csv")
train
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	256
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	260
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	260
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	270
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	60
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	200
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	300
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	80
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	39

2000 rows × 21 columns

2.Handle missing values appropriately to prepare the datasets for applying machine learning models.

```
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
```

```
X = train.drop("price_range", axis = 1).values
y = train["price_range"].values
```

3. Apply machine learning models with a grid search optimizer to the cleaned datasets and evaluate the models' performance.

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

```
model = RandomForestClassifier(random_state = 42)

# Grid Search parameters
parameter_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 15, 30, 45],
    'min_samples_split': [3, 6, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialized Grid Search
grid_search = GridSearchCV(estimator = model,
                           param_grid = parameter_grid,
                           cv = 5,
                           scoring = 'accuracy',
                           n_jobs = -1)

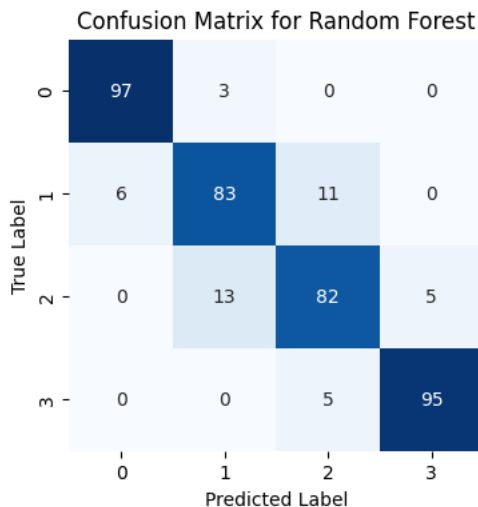
# Fit the model
grid_search.fit(X_train, y_train)

# Find the best estimator from GridSearch
best_parameter = grid_search.best_estimator_

# Predict the result
y_pred = best_parameter.predict(X_test)

# The evaluation
result = confusion_matrix(y_test, y_pred)
```

```
# Plot Confusion Matrix
plt.figure(figsize=(4, 4))
sns.heatmap(result, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels = np.unique(y_test), yticklabels = np.unique(y_test))
plt.title(f'Confusion Matrix for Random Forest')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



4. Use two feature selection methods to identify the most important features from the datasets.

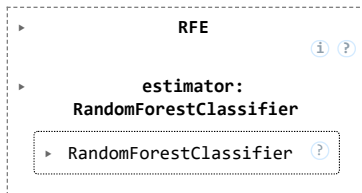
A - Recursive Feature Elimination

```
X = train.drop(['price_range'], axis=1)
y = train['price_range']
```

```
model = RandomForestClassifier(n_estimators = 100, random_state = 42)

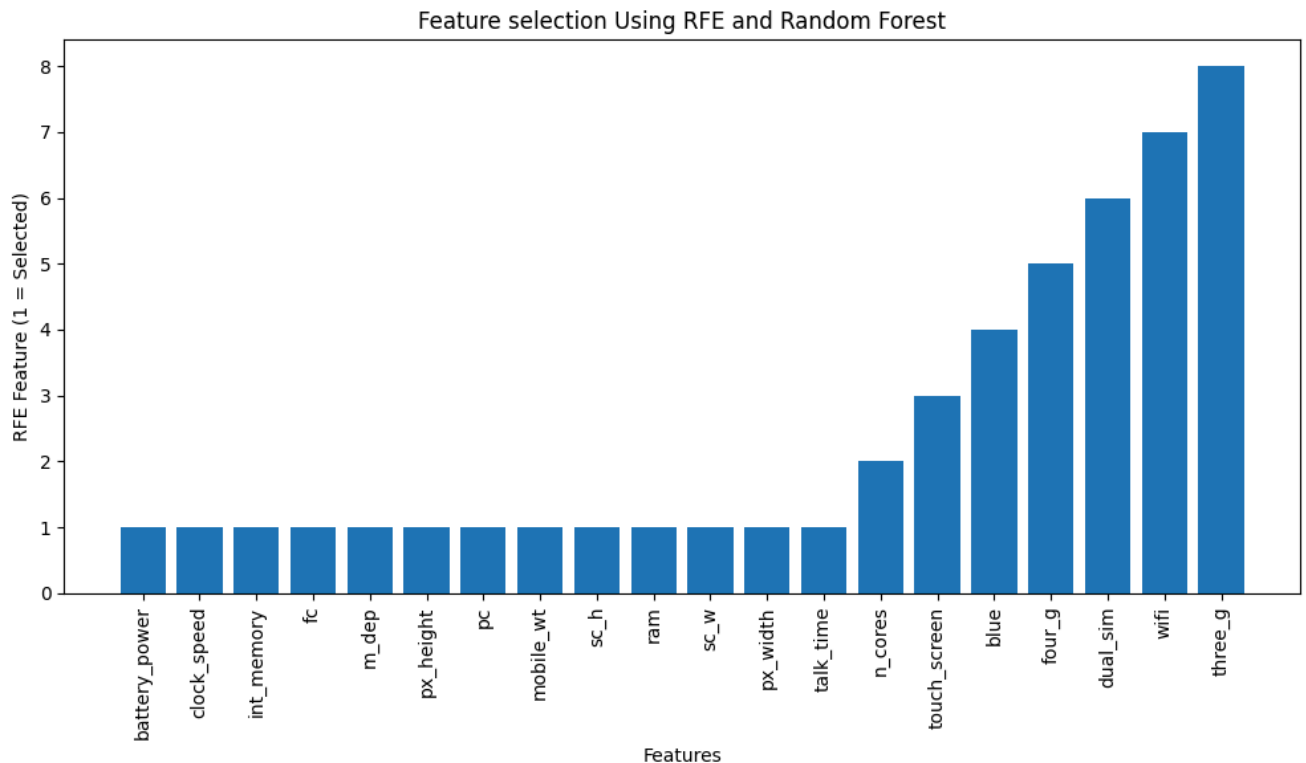
model_rfe = RFE(estimator = model, n_features_to_select = 13)

model_rfe.fit(X, y)
```



```
rank = pd.DataFrame({
    "Feature": X.columns,
    "RFE_Feature": model_rfe.ranking_
}).sort_values('RFE_Feature')
```

```
plt.figure(figsize=(10, 6))
plt.bar(rank['Feature'], rank['RFE_Feature'])
plt.xlabel("Features")
plt.ylabel("RFE Feature (1 = Selected)")
plt.title("Feature selection Using RFE and Random Forest")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



B - Embedded Methods (Feature Selection Happens During Training)

```
X = train.drop(['price_range'], axis=1)
y = train['price_range']
```

```
model = RandomForestClassifier(random_state = 42)
# Fit the model
model.fit(X, y)

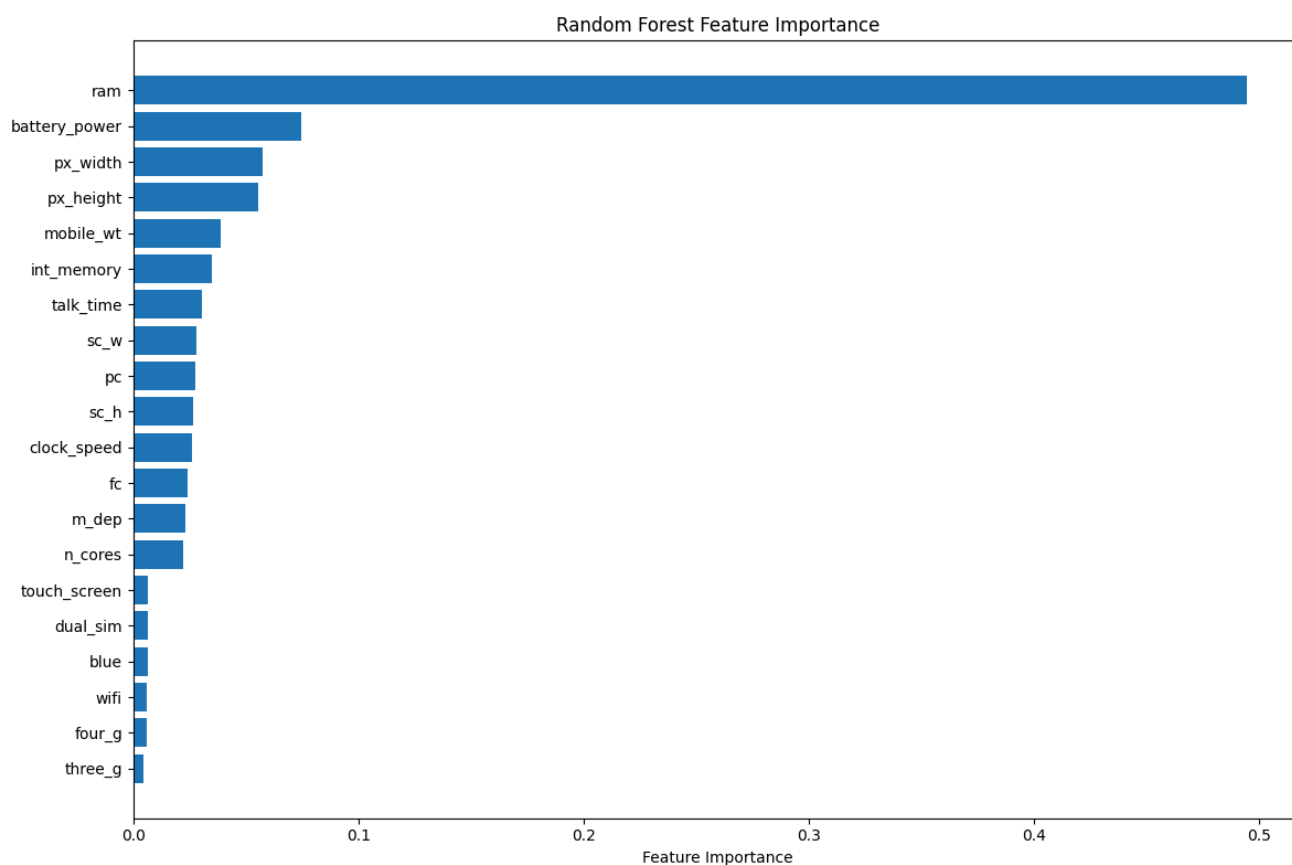
importance = model.feature_importances_

feature_names = X.columns if hasattr(X, "columns") else [f"feature {i}" for i in range(X.shape[1])]

train_importance = pd.DataFrame({
    "Feature": feature_names,
    "Importance": importance
})

train_importance = train_importance.sort_values(by = "Importance", ascending = False)

# Plot
plt.figure(figsize=(12, 8))
plt.barh(train_importance['Feature'][::-1], train_importance['Importance'][::-1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()
```



- 5. Apply machine learning models to the selected features and evaluate the models.

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

```
model = RandomForestClassifier(n_estimators = 100, random_state = 42)
```

```
# Fit the model
model.fit(X_train, y_train)
```

```
▼ RandomForestClassifier ⓘ ?
RandomForestClassifier(random_state=42)
```

```
# This line will produce percentage of importance for each value
importance = model.feature_importances_
importance
```

```
array([0.07456066, 0.00738241, 0.03033583, 0.00731484, 0.02605006,
       0.00722202, 0.03803845, 0.02541235, 0.04192694, 0.0235533 ,
       0.0294452 , 0.05730417, 0.0561254 , 0.4661135 , 0.02973062,
       0.02918127, 0.03089525, 0.00629718, 0.00626286, 0.00684769])
```

```
#This code return the index of feature according to importance
index = np.argsort(importance)[::-1]
index
```

```
array([13,  0, 11, 12,  8,  6, 16,  2, 14, 10, 15,  4,  7,  9,  1,  3,  5,
       19, 17, 18])
```

```
# 15 the number of feature I want to used
top_index = index[:15]
```

```
# obtain feature names
feature = X.columns[top_index]
```

```
new_X_train = X_train[feature]
new_X_test = X_test[feature]
```

```
# Here we compare two models to see what is the best
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100,
    random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
}
```

```
# Evaluate the model
metrics = []
for name, model in models.items():
    model.fit(new_X_train, y_train)
    y_pred = model.predict(new_X_test)
    print(f"Accuracy : {name}")
    print(accuracy_score(y_test, y_pred))
```

```
Accuracy : Random Forest
0.88
Accuracy : Decision Tree
0.8125
```

