

```
from google.colab import files
uploaded=files.upload()
print(uploaded)
```

لم يتم تحميل أي ملفات

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.csv

```
('train.csv': b'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\n842,0,2,2,0,1,0,7,0.6,188,2,2,2
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    confusion_matrix, accuracy_score,
    precision_score, recall_score, f1_score, classification_report
)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv('train.csv')
display(data.describe())
```

	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	t
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	1251.515500	2124.213000	12.306500	5.767000	11.011000	0.
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	432.199447	1084.732044	4.213245	4.356398	5.463955	0.
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	500.000000	256.000000	5.000000	0.000000	2.000000	0.
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	874.750000	1207.500000	9.000000	2.000000	6.000000	1.
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	1247.000000	2146.500000	12.000000	5.000000	11.000000	1.
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	1633.000000	3064.500000	16.000000	9.000000	16.000000	1.
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	1998.000000	3998.000000	19.000000	18.000000	20.000000	1.

8 rows × 21 columns

```
print(data.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
```

```
data.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
```

```
data.describe()
```

	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	t
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	1251.515500	2124.213000	12.306500	5.767000	11.011000	0.
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	432.199447	1084.732044	4.213245	4.356398	5.463955	0.
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	500.000000	256.000000	5.000000	0.000000	2.000000	0.
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	874.750000	1207.500000	9.000000	2.000000	6.000000	1.
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	1247.000000	2146.500000	12.000000	5.000000	11.000000	1.
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	1633.000000	3064.500000	16.000000	9.000000	16.000000	1.
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	1998.000000	3998.000000	19.000000	18.000000	20.000000	1.
8 rows × 21 columns																		

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

count
price_range
1      500
2      500
3      500
0      500

dtype: int64
```

```
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le
```

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.28, random_state=28, stratify=y)
```

```
def evaluate_model(model_phone, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
    report = classification_report(y_true, y_pred)

    # Output results
    metrics = {
        'Model Name': model_phone,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'Classification Report': report
    }

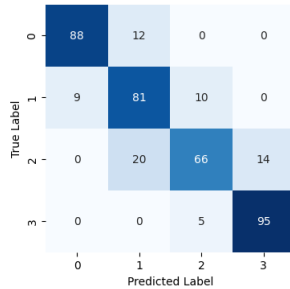
    # Plot Confusion Matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true),
                yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_phone}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
    return metrics
```

```
!RANDOMFOREST
rf_classifier = RandomForestClassifier(n_estimators=100,
                                     random_state=28)
# Train the model
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
                                    y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == "Classification Report":
        print(value)
    #Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: {value}")
```



```
#DECISIONTREE
DT = DecisionTreeClassifier()
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for DecisionTreeClassifier

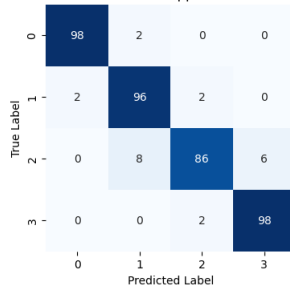


Model Name:
DecisionTreeClassifier
Accuracy: 0.8250
Precision: 0.8276
Recall: 0.8250
F1 Score: 0.8231

	precision	recall	f1-score	support
0	0.91	0.88	0.89	100
1	0.72	0.81	0.76	100
2	0.81	0.66	0.73	100
3	0.87	0.95	0.91	100
accuracy			0.82	400
macro avg	0.83	0.82	0.82	400
weighted avg	0.83	0.82	0.82	400

```
#SVM
SVM = SVC()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for Support Vector Machine



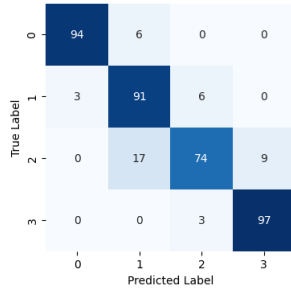
Model Name:
Support Vector Machine
Accuracy: 0.9450
Precision: 0.9459
Recall: 0.9450
F1 Score: 0.9445

	precision	recall	f1-score	support
0	0.98	0.98	0.98	100
1	0.91	0.96	0.93	100
2	0.96	0.86	0.91	100
3	0.94	0.98	0.96	100
accuracy			0.94	400
macro avg	0.95	0.94	0.94	400
weighted avg	0.95	0.94	0.94	400

```
from sklearn.model_selection import GridSearchCV
# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=20)
# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4, 6],
    'max_features': [None, 'sqrt', 'log2']
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
```

```
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
          f"{key}: {value}")
# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for RandomForestClassifier



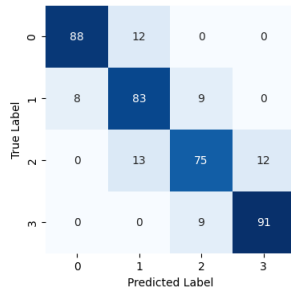
Model Name:
RandomForestClassifier
Accuracy: 0.8906
Precision: 0.8935
Recall: 0.8906
F1 Score: 0.8888

	precision	recall	f1-score	support
0	0.97	0.94	0.95	100
1	0.80	0.91	0.85	100
2	0.89	0.74	0.81	100
3	0.92	0.97	0.94	100
accuracy			0.89	400
macro avg	0.89	0.89	0.89	400
weighted avg	0.89	0.89	0.89	400

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

```
# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=20)
# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4, 6],
    'max_features': [None, 'sqrt', 'log2']
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5,
                           scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_DT = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
                                    y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: {value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for DecisionTreeClassifier



Model Name:
DecisionTreeClassifier
Accuracy: 0.8425
Precision: 0.8438
Recall: 0.8425
F1 Score: 0.8424

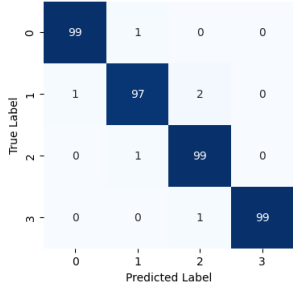
	precision	recall	f1-score	support
0	0.92	0.88	0.90	100
1	0.77	0.83	0.80	100
2	0.81	0.75	0.78	100
3	0.88	0.91	0.90	100
accuracy			0.84	400
macro avg	0.84	0.84	0.84	400
weighted avg	0.84	0.84	0.84	400

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_split': 10}

```
# Define the Support Vector Machine model
SVM = SVC(random_state=20)
# Define the hyperparameters for grid search
param_grid = {'C': [0.01, 0.1, 1, 10],
              'kernel': ['linear', 'rbf', 'poly'],
              'gamma': ['scale', 'auto'],
              'degree': [3, 4, 5],
              'class_weight': [None, 'balanced']}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
                           scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_
# Make predictions using the best model
```

```
y_pred = best_SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f'{key}: {value:.4f}" if isinstance(value, float) else
f'{key}: \n{value}')
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for Support Vector Machine



Model Name:
Support Vector Machine
Accuracy: 0.9850
Precision: 0.9851
Recall: 0.9850
F1 Score: 0.9850

	precision	recall	f1-score	support
0	0.99	0.99	0.99	100
1	0.98	0.97	0.97	100
2	0.97	0.99	0.98	100
3	1.00	0.99	0.99	100
accuracy			0.98	400
macro avg	0.99	0.99	0.99	400
weighted avg	0.99	0.98	0.99	400

Best hyperparameters found by GridSearchCV:
{'C': 0.01, 'class_weight': None, 'degree': 3, 'gamma': 'scale', 'kernel': 'linear'}

```
from sklearn.feature_selection import RFE
rf = RandomForestClassifier(n_estimators=200, random_state=42)
# Apply RFE (select top 5 features)
rfe = RFE(estimator=rf, n_features_to_select=5)
rfe.fit(X, y)
print("Selected Features Mask:", rfe.get_support())
print("Ranking of Features:", rfe.ranking_)
```

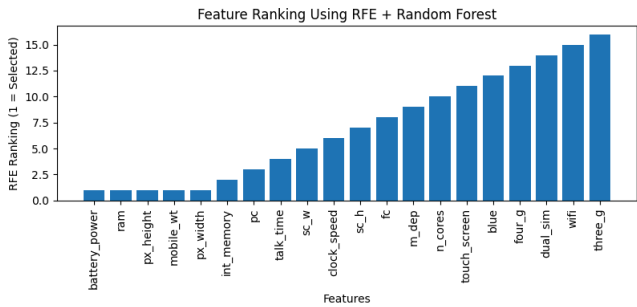
Selected Features Mask: [True False False False False False True False False True
True True False False False False]
Ranking of Features: [1 12 6 14 8 13 2 9 1 10 3 1 1 1 7 5 4 16 11 15]

```
import pandas as pd
import matplotlib.pyplot as plt

# Get feature names from the original DataFrame, excluding the target variable
feature_names = data.drop('price_range', axis=1).columns

ranking_data = pd.DataFrame({
    'Feature': feature_names,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')

plt.figure(figsize=(8, 4))
plt.bar(ranking_data['Feature'], ranking_data['RFE_Ranking'])
plt.xlabel("Features")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("Feature Ranking Using RFE + Random Forest")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



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

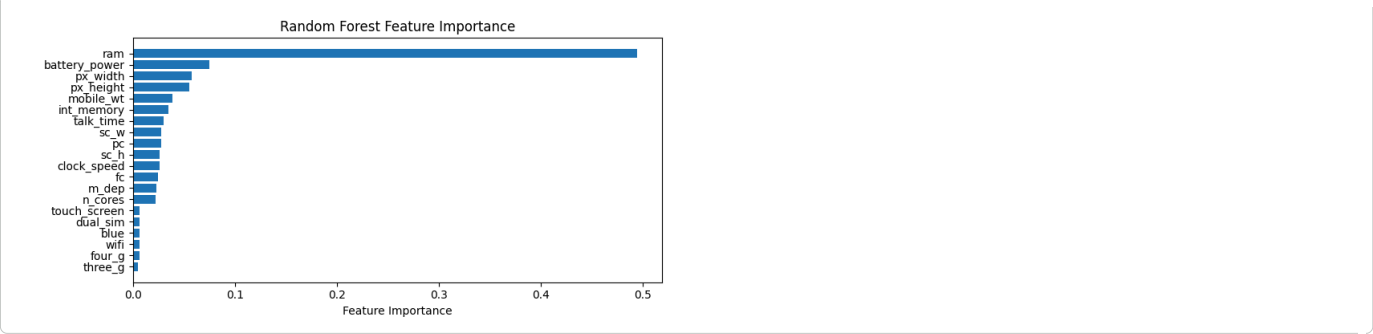
# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

# Extract importances
importances = rf.feature_importances_
feature_names = X.columns

# Create DataFrame
df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances})

# Sort by importance
df_importance = df_importance.sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(8, 4))
plt.barh(df_importance['Feature'][:-1], df_importance['Importance'][:-1])
plt.xlabel("Feature Importance")
plt.title("Random Forest Feature Importance")
```



```
#REF
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
k = 5
rf_selector = RandomForestClassifier(n_estimators=200,
    random_state=42)
rfe = RFE(estimator=rf_selector, n_features_to_select=k)
# Fit RFE on training data
rfe.fit(X_train, y_train)
# Transform datasets
X_train_k = rfe.transform(X_train)
X_test_k = rfe.transform(X_test)
# Get selected features
selected_features = X.columns[rfe.get_support()]
print("Selected Features by RFE:", list(selected_features))

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=20),
    "Decision Tree": DecisionTreeClassifier(random_state=20),
    "SVM": SVC(kernel='rbf')
}
metrics = []

for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
    average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted'),
        "F1 Score": f1_score(y_test, y_pred, average='weighted')
    })

results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RFE-Selected Features:")
results_df
print(results_df.to_string(index=False))
```

Selected Features by RFE: ['battery_power', 'mobile_wt', 'px_height', 'px_width', 'ram']

Model	Accuracy	Precision	Recall	F1 Score
Random Forest	0.9325	0.934017	0.9325	0.932830
Decision Tree	0.8525	0.858320	0.8525	0.853924
SVM	0.9650	0.965743	0.9650	0.965034

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

k = 5
rf = RandomForestClassifier(n_estimators=200, random_state=20)
rf.fit(X_train, y_train)
# Get importance scores
importances = rf.feature_importances_
# Sort indices by importance (descending)
sorted_idx = np.argsort(importances)[::-1]
# Select top-k feature indices
top_k_idx = sorted_idx[:k]
# Get feature names
selected_features = X.columns[top_k_idx]
print("Selected Features by RF Importance:", list(selected_features))
# Reduce datasets to selected features
X_train_k = X_train[selected_features]
X_test_k = X_test[selected_features]

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
    random_state=20),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}
metrics = []

for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
    average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted'),
        "F1 Score": f1_score(y_test, y_pred, average='weighted')
    })

results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
results_df
print(results_df.to_string(index=False))
```

Selected Features by RF Importance: ['ram', 'battery_power', 'px_height', 'px_width', 'mobile_wt']

Model	Accuracy	Precision	Recall	F1 Score
Random Forest	0.9325	0.933157	0.9325	0.932437
Decision Tree	0.8425	0.847403	0.8425	0.843609
SVM	0.9650	0.965743	0.9650	0.965034

Start coding or generate with AI.