Predict Mobile Phone Pricing

Objective:
The goal of this project is to develop a machine learning model that predicts the price range of smartphones based on various features. The dataset contains features such as the number of cores, internal memory, RAM, camera quality, screen size, and other smartphone specifications. The task is to build a predictive model and evaluate it using multiple classification algorithms.

Data Preprocessing

1. **Data Exploration:**
   The dataset is first explored to identify any missing values, duplicate entries, and outliers. The dataset includes both categorical and continuous features, which are processed accordingly.

```
df_train.isna().sum()
df_train.duplicated().sum()
```

2. **Handling Missing Values:**
   Any missing or null values in the dataset are handled by either removing the rows with missing data or replacing the missing values with appropriate statistical measures, such as the mean for numerical columns.

```
df_train.drop(px_height_noiseIndex_train, inplace=True)
df_train["sc_w"].replace(sc_w_noise_train, round(df_train.sc_w.mean()), inplace=True)
```

3. **Outlier Detection and Removal:**
   Outliers are detected using the Interquartile Range (IQR) method and are removed if necessary, particularly in the columns that contain extreme values.

```
outlier_info.append({
    "Feature": col,
    "Outlier_index": indexes,
    "Outlier%": (len(indexes) / df_train.shape[0]) * 100
})
outlier_info = pd.DataFrame(outlier_info)
```

4. **Feature Scaling:**
   Continuous variables are standardized using *StandardScaler* to ensure that they are on the same scale, which is crucial for algorithms like SVM and Decision Trees.

```
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = scaler.feature_names_in_)
X_test = pd.DataFrame(scaler.transform(X_test), columns = scaler.feature_names_in_)
```

Model Development

1. **Model Selection:**
   Several classification models are considered for the task:
   - **Decision Tree Classifier**
   - **Random Forest Classifier**
   - **Support Vector Machine (SVM)**
2. **Hyperparameter Tuning Function:**

To optimize model performance, hyperparameter tuning is performed using *GridSearchCV* to test different combinations of hyperparameters.

```
def hyperparameter_tuning(model, params, X_train, y_train):
    kfold = StratifiedKFold(shuffle=True, random_state=0)
    grid_model = GridSearchCV(model, param_grid=params, cv=kfold, scoring="accuracy", n_jobs=-1)
    grid_model.fit(X_train, y_train)
    print("Best hyperparameters:\n", grid_model.best_params_, "\nBest accuracy score:\n", grid_model.best_score_)
    return grid_model.best_estimator_
```

3. **Model Evaluation Function:**

After training the models, their performance is evaluated using the following function, which includes the confusion matrix, classification report, and a bar chart comparison of train vs test scores.

```
def model_evaluation(model, X_train, X_test, y_train, y_test, model_name):
    sns.set_style("darkgrid")
    y_pred_test = model.predict(X_test)
    y_pred_train = model.predict(X_train)
    cm = confusion_matrix(y_test, y_pred_test)
    train_test_score = pd.DataFrame({
        "Train_score": [model.score(X_train, y_train)],
        "Test_score": [model.score(X_test, y_test)]
    }, index=[model_name])

    print("Test result".center(60, "*"), classification_report(y_test, y_pred_test), sep="\n")
    print("Train result".center(60, "*"), classification_report(y_train, y_pred_train), sep="\n")

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4), dpi=80)
    ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, ax=ax1)
    ax1.grid()

    train_test_score.plot.barh(ax=ax2, cmap="coolwarm")
    set_freq_label(ax2, "%.2f")
    plt.margins((0.09))
    plt.tight_layout()
    return train_test_score
```

Training and Hyperparameter Tuning

1. **Decision Tree Model:**

The Decision Tree model is trained with hyperparameter tuning for optimal accuracy.

```
tree_pipeline = make_pipeline(StandardScaler(), DecisionTreeClassifier())
params = {
    'decisiontreeclassifier__max_depth': [5, 10, 20, 25, 30],
    'decisiontreeclassifier__min_samples_leaf': [5, 10, 20, 50, 100, 150],
    'decisiontreeclassifier__min_samples_split': [5, 10, 20, 50, 100, 150]
}
grid_model_dts = hyperparameter_tuning(tree_pipeline, params, X_train, y_train)
train_test_score_dts = model_evaluation(grid_model_dts, X_train, X_test, y_train, y_test, "DTs")
```

2. **Random Forest Model:**

The Random Forest model is also tuned using the same approach.

```
rl = make_pipeline(StandardScaler(), RandomForestClassifier(n_jobs=-1))
params = {
    'randomforestclassifier__n_estimators': [10, 20, 30, 40],
    'randomforestclassifier__max_depth': np.arange(5, 10),
    'randomforestclassifier__min_samples_leaf': [1, 2, 3],
    'randomforestclassifier__min_samples_split': np.arange(2, 10)
}
grid_model_rl = hyperparameter_tuning(rl, params, X_train, y_train)
train_test_score_rl = model_evaluation(grid_model_rl, X_train, X_test, y_train, y_test, "Random Forest")
```

3. **Support Vector Machine Model:**

Finally, the Support Vector Machine (SVM) model is trained with hyperparameter tuning.

```
svm = SVC()
params = {'kernel': ['linear'], 'C': [0.1, 0.3, 1, 2, 2.5, 3, 3.5, 4, 10]}
grid_model_svm = hyperparameter_tuning(svm, params, X_train, y_train)
train_test_score_svm = model_evaluation(grid_model_svm, X_train, X_test, y_train, y_test, "SVM")
```

Model Comparison and Final Selection

1. **Model Comparison:**

A comparison of all models (Decision Tree, Random Forest, and SVM) is presented using a bar chart, showing their accuracy scores on the test set. This allows us to select the best-performing model.

```
models_comparison = pd.concat([train_test_score_dts, train_test_score_rl, train_test_score_svm])
ax = models_comparison.plot(kind="barh", cmap="coolwarm", figsize=(14, 8))
plt.axvline(x=.8, ls="--", color="y")
plt.title("Model Comparison", fontweight="bold", size=23, y=1.06)
set_freq_label(ax, fmt="%.2f%%")
```

2. **Final Model Training:**

After comparing the models, the SVM model is selected as the final model, and it is retrained using the entire dataset *(X, y)*.

```
Final_model = make_pipeline(StandardScaler(), SVC())
Final_model[1].set_params(**grid_model_svm.get_params())
Final_model.fit(X, y)
```

Prediction on Test Data

1. **Predicting Price Range on Test Set:**

The final model is now used to predict the price range of smartphones in the test dataset *(df_test)*.

```
df_test["price_range"] = Final_model.predict(df_test.drop("price_range", axis=1))
```