Project: Smartphone Price Prediction Using Machine Learning

Introduction:

In today's rapidly evolving smartphone market, accurately predicting a device's price range has become increasingly important for both consumers and businesses. This project aims to develop a robust machine learning model that can effectively predict the price category (low cost, medium cost, high cost, very high cost) of a smartphone based on its various technical specifications.
By leveraging the power of machine learning techniques, this project seeks to address the following:

- Improved Consumer Decision Making: Consumers can utilize the model's predictions to compare different smartphones within their budget and make informed purchase decisions.
- Enhanced Business Efficiency: Businesses can employ the model to optimize pricing strategies, target specific consumer segments, and gain valuable insights into market trends.

Summary:

This project utilizes the XGBoost classification algorithm to build a machine learning model for smartphone price prediction. The model is trained on a dataset containing various smartphone specifications and their corresponding price ranges. The project comprehensively covers data cleaning, exploratory data analysis, model training, hyperparameter tuning, evaluation, and prediction on unseen data.
The documented code outlines the following key steps:

1. Data Loading and Exploration: Import necessary libraries, load the training data, and gain initial insights into its structure and content.
2. Data Cleaning: Address missing values through iterative imputation using a Random Forest Regressor.
3. Exploratory Data Analysis (EDA): Perform visual analysis to understand the distribution of features, identify potential outliers, and explore relationships between variables.
4. Feature Engineering: Define the target variable (price range) and select relevant features for model training.
5. Model Training and Evaluation: Split data into training and testing sets, train an XGBoost classifier, and evaluate its performance using accuracy score and confusion matrix.
6. Hyperparameter Tuning (Optional): Employ GridSearchCV to identify the optimal hyperparameter configuration for the XGBoost model.
7. Prediction on New Data: Load and process unseen data (test set), make predictions using the best model, and save the predicted price categories.

The final outcome of this project is a documented and well-performing machine learning model capable of predicting smartphone price ranges based on technical specifications. This model can be further integrated into applications or web services to provide real-time price prediction for consumers or businesses.
Note: This code assumes you have the necessary libraries installed and the training and test data files ('train - train.csv' and 'test - test.csv') are available in the same directory.

The rest of the documented code :

1. Import Libraries:

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

# Import libraries for imputation
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor

# Import library for XGBoost model
from xgboost import XGBClassifier

# Import libraries for model evaluation
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Library to save models
import joblib
```

2. Load and Explore Data:

```
# Read the training data
train_df = pd.read_csv('train - train.csv')

# Get a quick overview of the data (first few rows)
print(train_df.head())
```

```
# Get information about the data (data types, missing values etc.)
print(train_df.info())
```

3. Data Cleaning - Imputation:

```
# Initialize IterativeImputer with RandomForestRegressor for imputation
imputer = IterativeImputer(estimator=RandomForestRegressor())

# Perform imputation on the training data
imputed_df = pd.DataFrame(imputer.fit_transform(train_df))

# Update column names after imputation
imputed_df.columns = train_df.columns

# Replace the original data with imputed data
train_df = imputed_df
```

4. Save Imputed Data (Optional):

```
# Save the imputed training data for future use
train_df.to_csv('imputed_train_data.csv')
```

5. Exploratory Data Analysis (EDA):

```
# Get summary statistics of the data
print(train_df.describe())

# Create a box plot to visualize the distribution of 'battery_power' with
outlier detection
plt.figure(figsize=(8, 6))
ax = sns.boxenplot(train_df['battery_power'])
```

```
# Identify outliers based on IQR (Interquartile Range)
outliers = train_df[train_df['battery_power'] >
ax.get_ylim()[1]]['battery_power']
ax.plot([1] * len(outliers), outliers, 'ro', alpha=0.6)  # Add red dots for
outliers

# Set plot title and labels
plt.title('Boxen Plot of Battery Power with Outliers')
plt.xlabel('Battery Power')
plt.ylabel('Value')
plt.show()

# Identify outliers for numerical features using z-scores
z_scores = np.abs(zscore(train_df))
outliers_mask = (z_scores > 3)
row_indices, column_indices = np.where(outliers_mask)

# Explore outliers but keep the data for now based on your assumption
outlier_values = train_df.values[row_indices, column_indices]
print("Outlier values with z-score > 3:")
print(list(zip(train_df.index[row_indices],
train_df.columns[column_indices])))

# Visualize the distribution of the target variable 'price_range'
plt.figure(figsize=(8, 6))
train_df['price_range'].value_counts().plot(kind='bar')
plt.title('Frequency of Values in Price Range')
plt.xlabel('Price Range')
plt.ylabel('Frequency')
plt.show()

# Visualize the relationship between 'fc' and 'pc' features
plt.figure(figsize=(8, 6))
sns.lineplot(data=train_df[['fc', 'pc']])
plt.title('Line Plot of Front Camera and Rear Camera')
plt.xlabel('Front Camera (MP)')
plt.ylabel('Rear Camera (MP)')
plt.show()
```

6. Define Target and Training Data:

```
# Define the target variable
target_data = train_df['price_range']

# Define the features (all columns except the last two)
X_data = train_df.iloc[:, :-2]
```

7. Train-Test Split and Model Training:

```
# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_data, target_data,
test_size=0.2, random_state=4
```

8. Train an XGBoost Classifier:

```
# Initialize XGBoost classifier
model = XGBClassifier()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate model performance using accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

9. Model Evaluation (Confusion Matrix):

```
# Calculate the confusion matrix to understand model performance for each
class
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')  # Adjust based
on your preference
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```python
plt.title('Confusion Matrix')
plt.show()
```

10. Model Optimization (Grid Search):

```python
# Define a parameter grid for XGBoost hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.3],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 0.1, 0.2],
}

# Initialize XGBoost classifier
model = XGBClassifier()

# Perform grid search using GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best hyperparameters from the grid search
best_params = grid_search.best_params_

# Train a new model with the best hyperparameters
best_model = XGBClassifier(**best_params)
best_model.fit(X_train, y_train)

# Make predictions on the test data using the best model
y_pred = best_model.predict(X_test)

# Evaluate the performance of the best model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy with Best Hyperparameters:", accuracy)

# Visualize the confusion matrix for the best model
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')  # Adjust based
on your preference
```

```
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix (Best Model)')
plt.show()
```

11. Save the Best Model:

```
# Save the best XGBoost model for future use (e.g., prediction on new data)
joblib.dump(best_model, 'best_xgb_model.pkl')
```

12. Load and Predict on Test Data:

```
# Read the test data
test_df = pd.read_csv('test - test.csv')

# Get a quick overview of the test data
print(test_df.head())

# Get information about the test data
print(test_df.info())

# Select features from the test data (assuming it has the same structure as
the training data)
test_df_model = test_df.iloc[:, 1:]

# Make predictions on the test data using the best model
predictedPriceRange = best_model.predict(test_df_model)
```

13. Process and Save Predictions:

```
# Create a DataFrame to store predictions
predictedPriceRange_df = pd.DataFrame(predictedPriceRange,
columns=['predictedPriceRange'])
```

```
# Define a mapping to convert predicted numeric values to price categories
category_mapping = {
    0: 'low cost',
    1: 'medium cost',
    2: 'high cost',
    3: 'very high cost'
}

# Add a new column for price categories based on the mapping
predictedPriceRange['price_category'] =
predictedPriceRange['predictedPriceRange'].map(category_mapping)
```

14. Saving the predicted file:

```
# Add a new column for price categories based on the mapping
predictedPriceRange_df['price_category'] =
predictedPriceRange_df['predictedPriceRange'].map(category_mapping)

# Get a glimpse of the predicted data with price categories
print(predictedPriceRange_df.head())

# Save the predicted price ranges and categories to a CSV file
predictedPriceRange_df.to_csv('predictValues.csv', index=False)

print("Successfully completed data cleaning, training, prediction, and
saved results!")
```