# Machine Learning Lab: Smartphone Price Prediction

## 1. Introduction

In this lab, you'll build a machine learning model to predict smartphone prices using a sample dataset. The lab will guide you through data loading, cleaning, preprocessing, encoding, scaling, model building, and evaluation. By the end, you'll understand the entire machine learning pipeline.

## 2. Prerequisites

Ensure you have the following Python libraries installed:  
```bash  
pip install pandas numpy scikit-learn matplotlib seaborn  
```

## 3. Loading the Dataset

Download the sample dataset (`smartphone\_data.csv`) and load it using Pandas.  
```python  
import pandas as pd  
df = pd.read\_csv('smartphone\_data.csv')  
print(df.head())  
```

## 4. Data Cleaning

Check for missing values and clean the dataset if necessary.  
```python  
# Check for missing values  
print(df.isnull().sum())  
  
# Optionally fill or drop missing values if found  
df.fillna(df.median(), inplace=True)  
```

## 5. Data Preprocessing

Perform basic data preprocessing by removing duplicates and checking for outliers.  
```python  
# Remove duplicates  
df.drop\_duplicates(inplace=True)  
  
# Visualize outliers using boxplot  
import seaborn as sns  
import matplotlib.pyplot as plt  
sns.boxplot(data=df)  
plt.show()  
```

## 6. Feature Encoding

Convert categorical data to numerical values using encoding.  
```python  
from sklearn.preprocessing import LabelEncoder  
label\_encoder = LabelEncoder()  
df['Brand'] = label\_encoder.fit\_transform(df['Brand'])  
df['Processor'] = label\_encoder.fit\_transform(df['Processor'])  
```

## 7. Defining Feature Variables and Feature Scaling

Scale the features to bring them to a similar range using StandardScaler.  
```python  
  
scaler = StandardScaler()  
X = df.drop('Price (USD)', axis=1)  
y = df['Price (USD)']  
X\_scaled = scaler.fit\_transform(X)  
```

## 8. Splitting the Data

Split the dataset into training and testing sets.  
```python  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  
```

## 9. Model Building

Build a regression model using Random Forest.  
```python  
from sklearn.ensemble import RandomForestRegressor  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
model.fit(X\_train, y\_train)  
```

## 10. Model Evaluation

Evaluate the model using metrics like Mean Squared Error and R2 Score.  
```python  
from sklearn.metrics import mean\_squared\_error, r2\_score  
y\_pred = model.predict(X\_test)  
print('Mean Squared Error:', mean\_squared\_error(y\_test, y\_pred))  
print('R2 Score:', r2\_score(y\_test, y\_pred))  
```

## 11. Conclusion

# In this lab, you successfully built a smartphone price prediction model using Random Forest. You performed data cleaning, preprocessing, encoding, and scaling before building and evaluating the model. You can further improve the model by tuning hyperparameters or trying other algorithms. Line-by-Line Code Explanation for Smartphone Price Prediction

|  |  |  |
| --- | --- | --- |
| Code | Purpose | Explanation |
| # Libraries and Packages | Import necessary libraries for data manipulation, visualization, and model building | Import necessary libraries for data manipulation, visualization, and model building. |
| import pandas as pd | Pandas is used for data manipulation and analysis | Pandas is used for data manipulation and analysis. |
| import seaborn as sns | Seaborn is a statistical data visualization library built on top of Matplotlib | Seaborn is a statistical data visualization library built on top of Matplotlib. |
| import matplotlib.pyplot as plt | Matplotlib is used for plotting graphs and visualizing data | Matplotlib is used for plotting graphs and visualizing data. |
| from sklearn.model\_selection import train\_test\_split | Splits the dataset into training and testing subsets for model evaluation | Splits the dataset into training and testing subsets for model evaluation. |
| from sklearn.ensemble import RandomForestRegressor | Imports the Random Forest Regressor for predicting smartphone prices | Imports the Random Forest Regressor for predicting smartphone prices. |
| from sklearn.metrics import mean\_squared\_error, r2\_score | Evaluates model performance using Mean Squared Error and R2 Score | Evaluates model performance using Mean Squared Error and R2 Score. |
| from sklearn.preprocessing import StandardScaler | Scales the numerical columns |  |
| # Load the dataset | Reads the smartphone data from a CSV file using Pandas | Reads the smartphone data from a CSV file using Pandas. |
| df = pd.read\_csv('smartphone\_data.csv') | Loads the CSV data into a DataFrame named 'df' | Loads the CSV data into a DataFrame named 'df'. |
| print(df.head()) | Displays the first 5 rows of the dataset for a quick view | Displays the first 5 rows of the dataset for a quick view. |
| # Check for missing values | Identifies missing values in the dataset | Identifies missing values in the dataset. |
| print(df.isnull().sum()) | Prints the number of missing values in each column | Prints the number of missing values in each column. |
| # Fill missing values for categorical columns using mode | Replaces missing categorical values using the most frequent value (mode) | Replaces missing categorical values using the most frequent value (mode). |
| for col in df.select\_dtypes(include='object').columns: | Selects only the categorical columns | Selects only the categorical columns. |
| df[col] = df[col].fillna(df[col].mode()[0]) | Fills missing values in categorical columns with their mode | Fills missing values in categorical columns with their mode. |
| # Fill missing values for numeric columns using median | Replaces missing numeric values using the median value | Replaces missing numeric values using the median value. |
| df.fillna(df.select\_dtypes(include='number').median(), inplace=True) | Applies median values to numeric columns with missing data | Applies median values to numeric columns with missing data. |
| # Remove duplicates | Removes duplicate rows from the dataset | Removes duplicate rows from the dataset. |
| df.drop\_duplicates(inplace=True) | Drops duplicate records to prevent data redundancy | Drops duplicate records to prevent data redundancy. |
| # Visualize outliers using boxplot | Displays boxplots to identify outliers in numeric data | Displays boxplots to identify outliers in numeric data. |
| sns.boxplot(data=df) | Creates a boxplot using Seaborn to visualize data distribution | Creates a boxplot using Seaborn to visualize data distribution. |
| plt.show() | Displays the boxplot | Displays the boxplot. |
| # One-Hot Encoding for categorical features | Converts categorical columns into numerical format using One-Hot Encoding | Converts categorical columns into numerical format using One-Hot Encoding. |
| df = pd.get\_dummies(df, columns=['Brand', 'Processor'], drop\_first=True) | Creates binary columns for categorical data and drops the first column to prevent redundancy | Creates binary columns for categorical data and drops the first column to prevent redundancy. |
| # Define features and target | Separates the features (X) and target variable (y) | Separates the features (X) and target variable (y). |
| X = df.drop('Price (USD)', axis=1) | X contains all columns except 'Price (USD)', which is the target variable | X contains all columns except 'Price (USD)', which is the target variable. |
| y = df['Price (USD)'] | y stores the target variable 'Price (USD)' | y stores the target variable 'Price (USD)'. |
| # Scaling the numerical features | Scale the numerical features |  |
| scaler = StandardScaler() |  |  |
| X\_scaled = scaler.fit\_transform(X) |  |  |
| # Split the dataset | Splits the data into training and testing sets | Splits the data into training and testing sets. |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) | 80% of the data is used for training, and 20% for testing with a fixed random state for reproducibility | 80% of the data is used for training, and 20% for testing with a fixed random state for reproducibility. |
| # Tune and train Random Forest model | Initializes and trains a Random Forest model | Initializes and trains a Random Forest model. |
| model = RandomForestRegressor(n\_estimators=200, max\_depth=10, random\_state=42) | Random Forest with 200 trees, a max depth of 10, and a fixed random state | Random Forest with 200 trees, a max depth of 10, and a fixed random state. |
| model.fit(X\_train, y\_train) | Trains the model using the training dataset | Trains the model using the training dataset. |
| # Evaluate the model | Predicts test data and evaluates model performance using MSE and R2 score | Predicts test data and evaluates model performance using MSE and R2 score. |
| y\_pred = model.predict(X\_test) | Generates predictions using the test dataset | Generates predictions using the test dataset. |
| print('Mean Squared Error:', mean\_squared\_error(y\_test, y\_pred)) | Calculates and prints the Mean Squared Error | Calculates and prints the Mean Squared Error. |
| print('R2 Score:', r2\_score(y\_test, y\_pred)) | Calculates and prints the R2 Score | Calculates and prints the R2 Score. |
| # Identify important features | Evaluates which features contributed most to the model’s predictions | Evaluates which features contributed most to the model’s predictions. |
| feature\_importance = pd.Series(model.feature\_importances\_, index=X.columns).sort\_values(ascending=False) | Extracts and sorts feature importance values | Extracts and sorts feature importance values. |
| print('Feature Importance:', feature\_importance) | Displays the feature importance scores | Displays the feature importance scores. |